

LA MAITRISE DE LA HP-48SX

PROGRAMMATION ET APPLICATIONS

Jean-Michel FERRARD



D3I DIFFUSION

GW - 6022

LA MAITRISE DE LA HP-48SX

TOME 2 : PROGRAMMATION ET APPLICATIONS

2^e EDITION

Jean-Michel FERRARD

Agrégé de Mathématiques
Professeur de Mathématiques Supérieures
au lycée Déodat de Séverac, Toulouse

D3I
BP 49
31528 RAMONVILLE ST-AGNE CEDEX

SOMMAIRE DU TOME 1

INTRODUCTION.....	
PRINCIPES DE BASE.....	
ELEMENTS DE PROGRAMMATION.....	
LA PILE OPERATIONNELLE.....	
Exercices:	
NOMBRES REELS OU COMPLEXES.....	
Exercices:	
LES LISTES.....	
Exercices:	
LES CHAINES DE CARACTERES.....	
Exercices:	
LES TABLEAUX.....	
Exercices:	
VARIABLES GLOBALES ET REPERTOIRES.....	
Exercices:	
EXPRESSIONS, CALCUL DIFFERENTIEL ET INTEGRAL.....	
Exercices:	
GRAPHISME, CLAVIER, SON.....	
Exercices:	
LA PILE OPERATIONNELLE, Corrigés.....	
NOMBRES REELS OU COMPLEXES, Corrigés.....	
LES LISTES, Corrigés.....	
LES CHAINES DE CARACTERES, Corrigés.....	
LES TABLEAUX, Corrigés.....	
VARIABLES GLOBALES ET REPERTOIRES, Corrigés.....	
EXPRESSIONS, CALCUL DIFFERENTIEL ET INTEGRAL, Corrigés.....	
GRAPHISME, CLAVIER, SON, Corrigés.....	
INDEX.....	

PREAMBULE

=====

La collection "**La maîtrise de la HP48SX**", comporte deux tomes:
HP48SX: Programmation et Exercices.
HP48SX: Programmation et Applications.

Les rôles sont ainsi partagés:

- * Le tome 1 vous permet de dominer la programmation de la HP48SX:
Pour cela il est organisé en chapitres traitant chacun d'un problème particulier.
Chaque chapitre est suivi d'exercices, qui sont corrigés et commentés dans la deuxième moitié du livre.
- * Le tome 2 est une collection de programmes couvrant les besoins (mathématiques en général) des utilisateurs de la HP48SX.

Les deux tomes sont donc complémentaires, et vous permettent de tirer le meilleur parti de votre calculatrice HP.

Les programmes contenus dans ce tome 2 sont issus du manuel: "**HP28S: Programmation et Applications**", paru il y a un an.
Les langages de programmation de la HP28S et de la HP48SX sont très proches l'un de l'autre. Néanmoins celui de la HP48SX contient de nombreuses améliorations dont je n'ai pas manqué de tirer parti dans les programmes de ce livre.

Pour prendre deux exemples:

- * Quand c'était nécessaire (ou simplement utile), j'ai utilisé les ressources qu'apportent les menus temporaires, ce qui donne à l'utilisateur un plus grand confort d'utilisation.
Le résultat est spectaculaire, par exemple, pour le chapitre "**Bases de données**".
- * De même, les programmes de tracés du répertoire '**STAT1**' sont considérablement améliorés par l'utilisation de l'instruction **LINE**, qui était absente dans le langage de la HP28 (les temps d'exécution de certains programmes sont divisés par 10).

Plus généralement, j'ai apporté le plus grand soin à l'écriture des programmes qui suivent, et dont j'espère qu'ils vous rendront de grands services.

J'ai plaisir à remercier **Sylvie Bagnéris**, de la Société **Action Informatique**, pour le rôle éminent qui est le sien dans la commercialisation de cet ouvrage, et qu'elle assure avec une gentillesse et une compétence remarquables.

Toulouse, Juillet 1990.

Jean-Michel FERRARD.

LA MAÎTRISE DE LA HP48SX.

Volume 2:

Programmation et Applications.

=====

Vous tenez le deuxième volume d'une collection ("La maîtrise de la HP48SX"), qui vous permettra d'exploiter toute la puissance de votre calculatrice HP.

Le premier volume ("Programmation et Exercices") vous a permis de comprendre - et je l'espère de dominer complètement - l'art de la programmation de la HP48SX.

J'ai voulu, dans ce volume 2, proposer une vaste bibliothèque de programmes "clefs en main" où chacun pourrait trouver son bonheur.

- Cet ouvrage devrait intéresser les étudiants et professeurs:
- * des classes scientifiques du second cycle des Lycées.
 - * des classes préparatoires aux écoles d'ingénieurs, de commerce, ou d'agronomie.
 - * des universités de sciences et de sciences économiques.
 - * des sections de techniciens supérieurs.
 - * des instituts universitaires de technologie.
 - * des écoles d'ingénieurs.

Tous les programmes ont été testés et utilisés "en situation", plus précisément devant des élèves de classes préparatoires.

Je pense pouvoir dire que tous les besoins (mathématiques bien sûr...) de ces élèves ont été passés en revue.

Bien entendu, un bon nombre des programmes peuvent être utilisés dès les classes du second cycle des Lycées.

Je souhaite que la lecture du premier volume vous ait apporté l'aisance nécessaire pour éventuellement modifier, selon votre goût, les programmes qui suivent.

J'ai personnellement fait le maximum pour aboutir à des algorithmes courts et rapides.

Néanmoins, la perfection n'existe pas en ce domaine, et les remaniements incessants qu'ont pu subir mes programmes depuis leur création me conduisent à penser que vous aussi pourrez les améliorer sur tel ou tel point.

Avant que vous n'alliez plus avant, je vous souhaite donc d'éprouver comme moi le plaisir qu'il y a à "dompter" cette calculatrice exigeante, mais passionnante.

L'ORGANISATION DE CE LIVRE:

Cet ouvrage est divisé en 15 chapitres correspondant à une famille de programmes à installer dans un répertoire particulier.

- 1) **ARITHMETIQUE:** Pgcd, Ppcm, simplification de fraction, estimations d'expressions rationnelles, décomposition en produit de facteurs premiers, équation $ax+by=c$ dans $\mathbb{Z}$...
- 2) **NOMBRES REELS ET COMPLEXES:** produits infinis, fractions continues
Calculs itératifs. Calculs trigonométriques.
Approximations rationnelles des éléments d'un tableau.
Calculs avec le nombre j . Racines n -ièmes d'un complexe.
Intégration dans le plan, etc ...
- 3) **POLYNOMES:** Opérations sur les polynômes. Zéros d'un polynôme de degré ≤ 4 . Méthode de Bairstow. Arithmétique des polynômes etc..
- 4) **FRACTIONS RATIONNELLES:** Opérations diverses sur les fractions rationnelles, et surtout décomposition en éléments simples.
- 5) **MATRICES:** Changements de base. Puissances de matrices. Calcul du rang. Polynôme caractéristique, vecteurs et valeurs propres. Résolution symbolique d'un système d'équations. Méthode du pivot. Relations linéaires et équations de sous-espaces vectoriels.
- 6) **TECHNIQUES D'ANALYSE:**
Tangentes, extremums, points d'inflexion. Polynôme de Lagrange.
Approximation au sens des moindres carrés. Systèmes non linéaires.
Série de Fourier. Equa diff. , etc..
- 7) **DEVELOPPEMENTS LIMITES:** Opérations sur les développements limités
D.L. des fonction usuelles, compositions de D.L. , etc...
- 8) **GEOMETRIE AFFINE ET EUCLIDIENNE:**
Equations, distances , écarts angulaires.
- 9) **GEOMETRIE DIFERENTIELLE:** Différentielle, divergence, laplacien, gradient, rotationnel . Rectification d'une courbe plane ou gauche. Centre de courbure. Aire limitée par une courbe plane. Intégrales curvilignes. etc..
- 10) **GRAPHISME:** Tracé de courbes paramétrées, en polaires, familles de courbes. Enveloppe d'une famille de droites etc...
- 11) **ENTIERS LONGS:** Opérations sur les entiers longs ; arithmétique.
- 12) **PROBABILITES:** Lois de probabilité et fonction de répartition des lois usuelles. Dénombrements.
- 13) **STATISTIQUES SIMPLES:** Différentes moyennes et caractéristiques d'une statistique simple. Courbe et indice de Gini. Histogramme. Polygone des fréquences cumulées. Médiale, etc...
- 14) **STATISTIQUES DOUBLES:** moyennes , variances des statistiques marginales. Corrélation. droites des moindres carrés, etc.... ajustements exponentiels et puissances, etc...
- 15) **BANQUES DE DONNES:** Créer une banque de données et y ajouter des enregistrements. Lire, modifier, trier une banque de données.

LES NOMS DE PROGRAMMES ET DE REPERTOIRES:

Chaque répertoire possède un nom particulier. Il en est de même évidemment pour les programmes qui les composent.

Il est important que vous conserviez ces dénominations: En effet, certains programmes en appellent d'autres et ces appels se font par le nom du programme appelé. De même certains programmes contiennent des passages temporaires à un autre sous-répertoire. Là encore le nom de celui-ci apparaît dans le texte du programme appelant.

Voici d'ailleurs les noms que j'ai choisi pour chacun de ces répertoires.

- 1) Arithmétique: 'ARIT'
- 2) Nombres réels ou complexes: 'R.C'
- 3) Polynômes: 'POLY'
- 4) Fractions rationnelles: 'FRAC' (ce répertoire doit être installé dans le répertoire 'POLY')
- 5) Développements limités: 'DL'
- 6) Calcul matriciel: 'MATR'
- 7) Techniques d'analyse: 'ANLY'
- 8) Géométrie: 'GEOM'
- 9) Géométrie différentielle: 'GDIF'
- 10) Graphisme: 'GRPH'
- 11) Entiers longs: 'LONG'
- 12) Probabilités: 'PROBA'
- 13) Statistiques simples: 'STAT1'
- 14) Statistiques doubles: 'STAT2'
- 15) Banques de données: 'DATA'

QUELS PROGRAMMES CHOISIR ?

Si vous n'avez pas équipé votre HP48SX d'une extension mémoire, vous serez amenés à faire un choix dans les programmes que vous implanterez dans votre calculatrice. En effet ce livre contient environ pour 37 K-octets de programmes.

Le choix que vous serez donc amené à faire dépend de votre spécialité, de vos goûts, et de la place en mémoire que nécessitent les programmes de tel ou tel répertoire.

S'il fallait absolument choisir, je pense que les répertoires 'ARIT', 'R.C', 'POLY', 'FRAC', 'MATR', 'DL' sont très utiles pour tout le monde.

Les répertoires 'PROBA', 'STAT1', 'STAT2' seront indispensables aux étudiants des classes préparatoires aux écoles de commerce. Ceux qui préparent les écoles d'ingénieurs apprécieront plutôt 'ANLY' ou 'GDIF'....

A titre d'information voici à peu près la place qu'occupent en mémoire les programmes des différents répertoires: (en octets)

'ARIT' : 1865	'R.C' : 2120	'POLY' : 5591
'FRAC' : 951	'DL' : 3626	'MATR' : 4846
'ANLY' : 3265	'GEOM' : 1008	'GDIF' : 2715
'GRPH' : 1447	'LONG' : 1941	'PROBA' : 1791
'STAT1' : 3314	'STAT2' : 1509	'DATA' : 1547

COMMENT SONT PRESENTES LES PROGRAMMES:

Chaque chapitre de ce livre contient les programmes d'un répertoire particulier. Il débute par une courte introduction.

Chaque programme est brièvement présenté. Un schéma fonctionnel donne l'aspect de la pile avant l'appel du programme et à la sortie de celui-ci. Un certain nombre de remarques peuvent alors être notées (le programme en question en appelle-t-il d'autres? ...).

Il n'était pas possible de commenter ici les nombreux programmes présentés (la taille de ce livre aurait peut-être doublé). On se reportera au volume 1 ("Programmation et Exercices") pour trouver des programmes abondamment commentés.

J'ai voulu que les listings soient faciles à copier et à comprendre (dans la mesure du possible). Pour cela:

- * Les différents mots du texte sont suffisamment espacés.
- * La structure du programme apparaît clairement grâce à l'emploi de l'indentation qui permet de distinguer l'entrée et la sortie des boucles, structures conditionnelles, sous-programmes etc... Bien entendu il n'est pas utile de respecter cette écriture, et en particulier le nombre des espaces, lors de la recopie du listing.
- * Quand un programme utilise un nom de programme, ou de répertoire, ou de variable globale nécessaire à son bon fonctionnement, et si ce nom est spécifique à ce livre (c'est-à-dire n'est pas un identificateur connu de la HP48SX), alors ce nom est souligné dans le listing du programme qui l'utilise.
- * Les programmes sont illustrés d'exemples.

Quelques conseils:

* Créez le répertoire (en vous plaçant dans le répertoire 'HOME') avant d'y entrer et d'y introduire un à un les programmes qui vous intéressent.

* Vérifiez le bon fonctionnement de ces programmes sur les exemples que je donne. Il se peut qu'un programme 'A', par exemple, appelle un programme 'B'. Dans ce cas 'B' devra être implanté en premier si vous voulez voir fonctionner 'A'.

* En cas de mauvais fonctionnement, vérifiez bien la conformité avec le listing de référence.

* Si vous rencontrez -1 dans un programme, ce n'est pas la même chose que - 1. Pour entrer -1 (sans espace entre - et 1), tapez 1, puis CHS.

* Evitez d'encombrer vos répertoires de variables ou de programmes qui ne vous servent plus. Une idée logique consiste à créer un répertoire spécifique aux applications et aux variables provisoires, répertoire que vous purgez régulièrement (Instruction CLVAR).

* Dans un répertoire contenant plus de 6 entrées, veillez à ce que les programmes les plus utiles apparaissent en tête de liste, et qu'ils soient groupés par thème. Vous y gagnerez en efficacité. Pensez à l'utilisation des instructions VARS et ORDER pour corriger l'ordre dans lequel les programmes apparaissent dans un répertoire donné.

* Faites attention au signe → : il apparaît dans de nombreux mots du langage de la HP48SX qu'il vaut mieux écrire en utilisant les menus: passez par exemple dans le menu 'OBJ' pour obtenir les mots →ARRY et →OBJ. Le signe → est également utilisé lors de la création de variables locales. Dans ce cas, il doit être tapé depuis la touche correspondante de la HP48SX.

* J'ai utilisé des minuscules pour désigner les variables locales. Une confusion entre une minuscule et une majuscule entraînerait une erreur. On peut bien sûr n'utiliser que des majuscules. Il convient alors de vérifier qu'aucune ambiguïté n'existe entre vos variables locales et certains identificateurs de la HP. On peut ainsi fort bien créer une variable locale nommée 'SIN', mais la fonction SINUS de la HP deviendrait alors momentanément inaccessible. On peut définir une variable locale 'end' mais c'est impossible avec le nom 'END'.

* Certains programmes de ce livre utilisent l'instruction IFERR...THEN. J'ai supposé que la fonction LASTARG était active sur votre machine (voir manuel de référence).

* Pour les programmes qui utilisent les fonctions trigonométriques, j'ai supposé que votre machine était en mode radians (sinon les résultats des programmes sont différents).

* Pour des raisons techniques, le caractère 0 (zéro) n'est pas imprimé en mode "slashé" dans la présente édition. Le risque de confusion entre la lettre majuscule O et le chiffre 0 existe donc mais il est minime: le contexte permet toujours de choisir, et je n'ai jamais dénommé O (la lettre..) une quelconque variable. Un 0 isolé est donc toujours le chiffre zéro. Soyez tout de même attentif...

* Pour chaque programme, j'ai indiqué la taille (en octets) et la somme de contrôle (Checksum), obtenues en appliquant l'instruction BYTES du menu MEMORY au nom de ce programme. La somme de contrôle vous permet de vérifier que le programme est correctement recopié.

Il se peut bien sûr qu'une erreur se soit glissée dans un listing, malgré mes vérifications. Je vous serais reconnaissant de me le signaler. Plus généralement j'accueillerai avec intérêt toute remarque et suggestion. Vous pouvez m'écrire au Lycée Déodat de Séverac (26, Bld Déodat de Séverac, 31076 Toulouse Cédex).

Jean-Michel FERRARD

ARITHMETIQUE

Le répertoire 'ARIT' regroupe les programmes consacrés aux entiers et aux nombres rationnels.

Certains des programmes qu'ils contient sont d'un usage courant et rendent de grands services. Citons par exemple celui qui permet d'évaluer une expression formée de nombres rationnels, telle que:

$$'1/25 + 27/11 - 3*51/4/9 + 13/7 - 9/5 + 22/17',$$

et qui donne, presque instantanément le résultat: (-52909/130900) sous forme d'une fraction simplifiée.

Dans le répertoire 'ARIT', vous pourrez donc installer quelques routines indispensables:

- 'FCTR' : décomposition d'un entier en produit de facteurs premiers.
- 'PGCD' : plus grand commun diviseur à deux entiers.
- 'PPCM' : plus petit commun multiple à deux entiers.
- 'SIMP' : simplification de fraction.
- 'ABXY' : résolution de l'équation en nombres entiers $ax+by=c$.
- 'CALC' : évaluation d'expressions rationnelles.

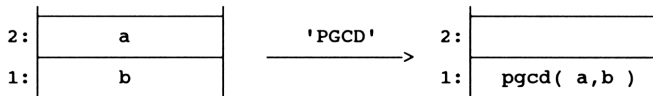
A coté, et si le coeur vous en dit, vous placerez les programmes suivants, qui sont plus spécialisés:

- 'LPRM' : liste des diviseurs premiers d'un entier.
- 'MOEB' : fonction de Moebius.
- 'EULER' : indicateur d'Euler.
- 'CYCLO' : calcul des polynômes cyclotomiques.

CALCUL DU P.G.C.D.

=====

'PGCD' calcule le plus grand commun diviseur de deux entiers a et b suivant le schéma fonctionnel suivant:



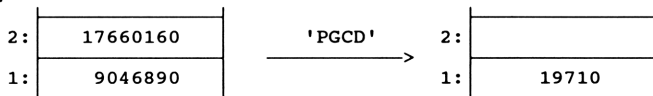
'PGCD': (checksum: # 31925d, taille: 46 octets)

```

«   WHILE   DUP
      REPEAT   SWAP   OVER   MOD   END
      DROP   ABS
»

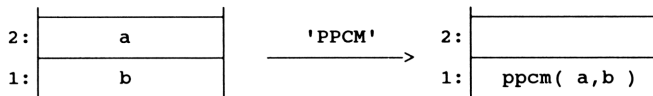
```

Exemple:

**CALCUL DU P.P.C.M.**

=====

'PPCM' calcule le plus petit commun multiple de deux entiers a et b suivant le schéma fonctionnel suivant:



N.B: le programme 'PPCM' appelle le programme 'PGCD':

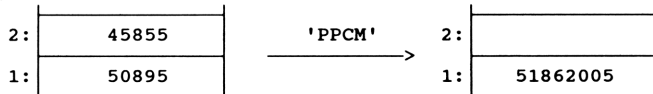
'PPCM': (checksum: # 21850d, taille: 36 octets)

```

«   DUP2   PGCD   /   *   ABS   »

```

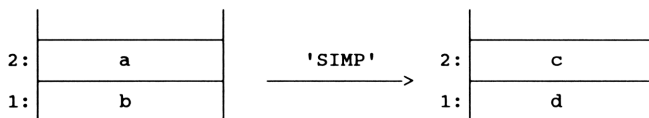
Exemple:



SIMPLIFICATION DE FRACTION.

=====

'SIMP' effectue la simplification d'une fraction a/b (où a et b sont deux entiers) selon le schéma suivant (c/d représentant la fraction après cette simplification) :

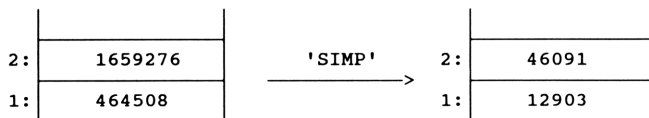


N.B: 'SIMP' appelle le programme 'PGCD'.

'SIMP': (checksum: # 14128d, taille: 43.5 octets)

« DUP2 PGCD ROT OVER / 3 ROLLD / »

Exemple: (en moins d'une seconde)



RESOLUTION DE L'EQUATION

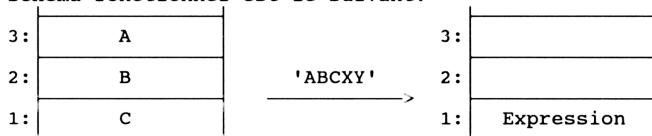
$$AX+BY=C \text{ (dans Z)}$$

=====

'ABCXY' permet de résoudre dans l'ensemble Z des entiers relatifs l'équation $AX+BY=C$, où A,B,C sont donnés, et où X,Y sont inconnus.

Une condition nécessaire et suffisante pour qu'il y ait des solutions (et il y en a alors une infinité) est que C soit divisible par le PGCD de A et B.

Le schéma fonctionnel est le suivant:



où 'Expression' décrit la solution générale du problème. Dans le cas où il n'y a aucune solution, le message d'erreur "Pas de solution" est affiché.

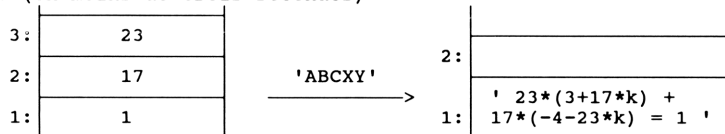
'ABCXY': (checksum: # 6640d, taille: 310 octets)

```

« → a b c
  « 0 1 b 3 →ARRY 1 0 a 3 →ARRY
    WHILE DUP 3 GET
      REPEAT
        SWAP DUP2 3 GET OVER
        3 GET / FLOOR * -
      END
      DROP c OVER 3 GET / DUP
      IF DUP FLOOR == THEN
        OVER 3 GET
        → p
        « * OBJ→ DROP2 a ROT b p /
          'k' * + * b ROT a p / 'k'
          * - * + c =
        »
      ELSE DROP2 a b c "Pas de solution" DOERR
      END
    »
  »

```

Exemple: (en moins de trois secondes)



Ce qui signifie que l'équation $23x+17y=1$ admet la solution particulière : $x=3$, $y=-4$, et que la solution générale est:

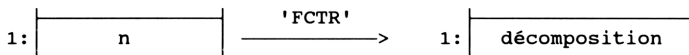
$$x = 3 + 17*k$$

$$y = -4 - 23*k, \text{ k étant un entier relatif quelconque.}$$

FACTORISATION D'UN ENTIER.

=====

'FCTR' effectue la décomposition d'un nombre entier n en un produit de facteurs premiers, selon le schéma fonctionnel:



où "décomposition" est une liste de nombres complexes (p,k) , chacun de ces nombres complexes (p,k) représentant un entier premier p figurant dans la décomposition de n , et l'exposant k correspondant.

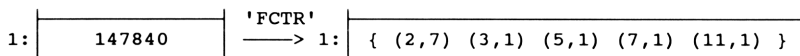
'FCTR': (checksum: # 50846d, taille: 231 octets)

```

«  ABS { } SWAP 1
   →  n  k
   «  WHILE n 1 >
       REPEAT
         IF k SQ n <
           THEN k DUP 2 > + 1 +
           ELSE n
         END
         'k' STO
         IF n k MOD NOT THEN
           k 0 n
           WHILE DUP k MOD NOT
             REPEAT
               k / SWAP 1 + SWAP
             END
           'n' STO R→C +
         END
       END
     »
  »

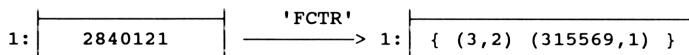
```

Exemple: (en moins de 2 secondes)



Car $147840 = 2^7 * 3 * 5 * 7 * 11$.

Exemple: (en 19 secondes)



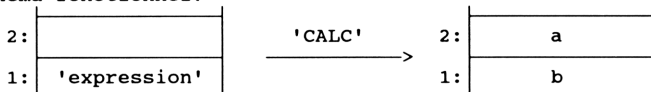
Car $2840121 = 3^2 * 315569$.

EVALUATION D'EXPRESSIONS RATIONNELLES.

=====

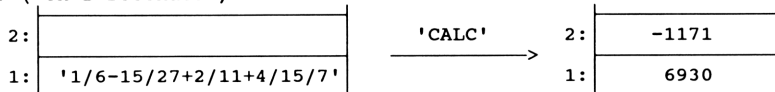
'CALC' évalue une expression algébrique constituée de sommes, différences, produits, quotients d'entiers, et donne le résultat sous forme d'une fraction simplifiée a/b. Le programme 'CALC' appelle d'ailleurs le programme 'SIMP'.

schéma fonctionnel:



```
'CALC': ( checksum: # 62365d, taille: 300 octets )
"  DUP  EVAL  SWAP  →STR  1
  → v  c  d
  "  WHILE  c  DUP  "/"  POS  DUP
    REPEAT
      1  +  OVER  SIZE  SUB  DUP  NUM
      IF  10  ==  THEN  2  OVER  SIZE  SUB  END
      'c'  STO  0
      WHILE  "0123456789"  c  1  1  SUB  POS  DUP
        REPEAT
          1  -  SWAP  10  *  +  c
          2  OVER  SIZE  SUB  'c'  STO
        END
      END  DROP  'd'  STO*
    END  DROP2
  v  d  *  .5  +  FLOOR  d  SIMP
"
"
```

Exemple: (en 2 secondes)



Attention !

Pour que le programme fonctionne correctement, chaque signe / de l'expression à évaluer doit être suivi d'un chiffre (et non d'une ouverture de parenthèse par exemple).

Ainsi '5/(11/3)' doit être remplacé par '5/11*3' ou par '15/11'.

De même, aucun dénominateur ne doit être élevé à une puissance quelconque.

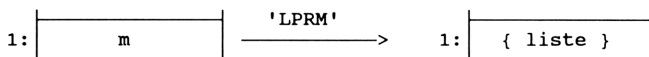
Ainsi '3/2^5' doit être remplacé par '3/32'.

Le programme 'CALC' effectue le produit de tous les dénominateurs présents dans l'expression. Une erreur d'arrondi se produira donc si ce produit est supérieur à 1E12.

DIVISEURS PREMIERS D'UN ENTIER.

=====

'LPRM' donne la liste de tous les diviseurs premiers positifs d'un entier m, suivant le schéma fonctionnel:



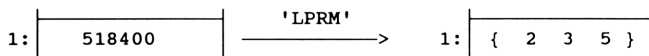
N.B: 'LPRM' appelle 'FCTR'.

'LPRM': (checksum: # 36961d, taille: 61 octets)

```

«   FCTR
    IF DUP    SIZE THEN
      OBJ→  →ARRY RE
      OBJ→  1   GET  →LIST
    END
»
  
```

Exemple: (une seconde)



(en fait, $518400 = (2^8) * (3^4) * (5^2)$).

N.B: Le programme 'LPRM' est appelé par 'CYCLO' et 'EULER'.

POLYNOMES CYCLOTOMIQUES.

=====

Le polynôme cyclotomique Φ_n d'ordre n est le polynôme unitaire dont les zéros sont les racines n -ièmes primitives de l'unité, c'est à dire les

$$w_k = \cos(2k\pi/n) + i \sin(2k\pi/n),$$

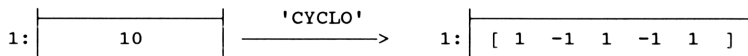
où k décrit les entiers compris entre 1 et n et premiers avec n .
Le degré du polynôme Φ_n est $\varphi(n)$, où φ est l'indicateur d'Euler.

Le schéma fonctionnel est le suivant:



Le résultat est obtenu sous la forme d'un vecteur représentant les composantes du polynôme, suivant les puissances décroissantes de l'indéterminée X .

Exemple: (en 4 secondes)



$$\text{car } \Phi_{10}(X) = X^4 - X^3 + X^2 - X + 1.$$

Exemple: le polynôme Φ_{30} , de degré 8, est obtenu en 13 secondes.
le polynôme Φ_{105} , de degré 48, est obtenu en 1 mn 5 s.

Le programme 'CYCLO' utilise la formule:

$$\Phi_n = \prod_{d|n} (X^d - 1)^{\mu(n/d)}$$
 où μ est la fonction de Moëbius, et où le produit est étendu à tous les diviseurs positifs de n .

N.B: Le programme 'CYCLO' appelle le programme 'LPRM'.

TEXTE DU PROGRAMME 'CYCLO'.

=====

```

'CYCLO': ( checksum: # 20046d, taille: 643 octets )
  " DUP { } 0 0
    → n m dp k pr
    " IF n 1 == THEN [ 1 -1 ]
      ELSE
        " 0
          1 dp SIZE FOR i
            k dp i GET MOD NOT +
          NEXT
          2 MOD
        "
        " → w
          " 1 OVER SIZE 1 GET n k / -
            IF w -1 == THEN SWAP END
            FOR i
              n k / i + DUP2 GET
              3 PICK i GET w * + PUT
            w STEP
          "
        "
        " OVER SIZE 1 GET n k /
          ROT * - 1 →LIST RDM
        "
        → ip dm rd
        " m LPRM DUP 'dp' STO
          OBJ→ 1 1 ROT START * NEXT
          'pr' STO
          [ 1 ]
          pr 1 FOR k
            IF pr k MOD NOT THEN
              IF ip EVAL NOT THEN
                -1 rd EVAL -1 dm EVAL
              END
            END
            -1 STEP
            1 pr FOR k
              IF pr k MOD NOT THEN
                IF ip EVAL THEN
                  1 dm EVAL 1 rd EVAL
                END
              END
            END
          END
        NEXT
      "
    "
  "

```

INDICATEUR D'EULER.

=====

L'indicateur d'Euler $\varphi(n)$ d'un entier naturel n est égal au nombre d'entiers naturels p compris entre 1 et n et qui sont premiers avec n .

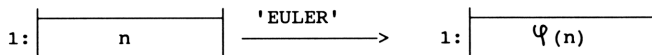
Si la décomposition de n en facteurs premiers s'écrit:

$n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$, alors $\varphi(n)$ est égal à

$$n \left(1 - \frac{1}{p_1}\right) \left(1 - \frac{1}{p_2}\right) \dots \left(1 - \frac{1}{p_k}\right).$$

En particulier, si n est premier, alors $\varphi(n) = n-1$.

'EULER' calcule l'indicateur d'Euler de l'entier n suivant le schéma fonctionnel:



N.B: Le programme 'EULER' appelle le programme 'LPRM'.

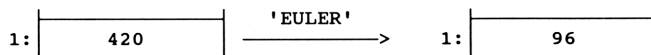
'EULER': (checksum: # 61654d, taille: 93 octets)

```

«   IF   DUP   ABS   1   >   THEN
    →   n
    «   n   LPRM   OBJ→   n   1   ROT   START
        DUP   ROT   /   -
    NEXT
    »
    END
»

```

Exemple: (1 seconde)



FONCTION DE MOEBIUS.

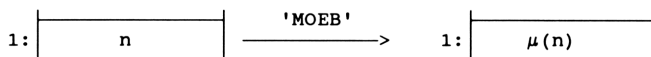
=====

La fonction de Moebius μ est définie sur l'ensemble des entiers naturels non nuls. Si n est un entier naturel non nul,

$\mu(n) = 0$ si n est divisible par le carré d'un entier premier.

si n n'est divisible par le carré d'aucun entier premier, alors μ est égal à 1 ou à -1 suivant que le nombre de diviseurs premiers de n est pair ou impair.

'MOEB' effectue le calcul de $\mu(n)$ suivant le schéma fonctionnel:



N.B: 'MOEB' appelle le programme 'FCTR'.

'MOEB': (checksum: # 27678d, taille: 101.5 octets)

```

«   FCTR  OBJ→
    → n
    «   IF   n
        THEN
            n →ARRY  IM  RNRM  2  <  1
            n  2  MOD  2  *  -  *
        ELSE  1
        END
    »
»
  
```

On trouve par exemple:

$\mu(1)=1,$	$\mu(2)=-1,$	$\mu(4)=0,$	$\mu(6)=1$
$\mu(30)=-1$ (en 1 seconde)	$\mu(210)=1$ (en 1 seconde).		

NOMBRES REELS ET NOMBRES COMPLEXES

Le répertoire consacré aux nombres réels ou complexes contient des programmes d'usage assez général, et dont le seul point commun est de ne pas pouvoir s'insérer facilement dans tel ou tel répertoire plus spécialisé.

Le répertoire 'R.C' contient les programmes suivants:

- 'PROD' : Produits partiels d'un produit infini.
- 'FRCN': calcul des fractions continues et des réduites d'un nombre réel donné.
- 'T+Q' : approximation, à la manière de l'instruction -Q, des éléments d'un tableau à coefficients réels ou complexes.
- 'I+J' : écriture d'un nombre complexe sous la forme $a+bj$.
- 'ITER' : permet d'effectuer des calculs récurrents (récurrence de pas quelconque).
- 'RCNC' : racines n-ièmes d'un nombre réel ou complexe.
- 'TRIG' : linéarisation des puissances de $\cos(t)$ et de $\sin(t)$, et opération inverse de la linéarisation.
- 'INTZ' : intégration le long d'un chemin du plan complexe.

CALCULS DE PRODUITS PARTIELS.

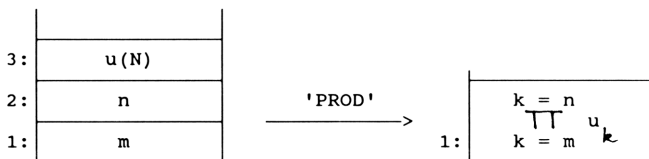
=====

'PROD' calcule le produit $\prod_{k=m}^{k=n} u_k$, où u_k est un réel dépendant de k.

m et n sont les deux entiers représentant les valeurs extrémales de l'indice de produit k.

Le calcul se fait à partir de la valeur k=m (celle qui est au niveau 1 de la pile) et se termine à la valeur k=n (celle qui est au niveau 2). (Il se peut que m > n, auquel cas les valeurs de k vont en décroissant. L'avantage est que l'utilisateur n'a pas à se soucier de l'ordre dans lequel il donne m et n. Un autre avantage est que l'on peut ainsi contrôler l'ordre du produit (pour que le calculateur utilise à plein sa précision, il vaut mieux lui faire multiplier d'abord les quantités les plus voisines de 1).

Le schéma fonctionnel est le suivant:



(où u(N) est une expression algébrique, ou un programme, permettant d'évaluer le terme u_N . Attention la majuscule "N" y est obligatoire).

'PROD': (Checksum: # 32428d, Taille: 108 octets)

```

«   DUP2   >   2   *   1   -
   "«   →   N"   5   ROLL   →STR   +   OBJ→
     →   s   u
   «   1   SWAP   ROT   FOR   N
       N   u   →NUM   *
       s   STEP
   »
»

```

Exemple: (en 6 secondes)



(si dans ce cas on inverse l'ordre de 100 et de 200 c'est-à-dire si on effectue le produit de k=200 jusqu'à k=100, on obtient le résultat 2.01506898486. Ce résultat est sans doute plus précis).

CALCULS DE FRACTIONS CONTINUES.

=====

Soit x un réel et a_1 sa partie entière. Si x n'est pas entier, on peut écrire: $x = a_1 + 1/x_1$ avec $x_1 > 1$.

Soit a_2 la partie entière de x_1 . Si x_1 n'est pas entier, on peut écrire: $x_1 = a_2 + 1/x_2$ avec $x_2 > 1$, et donc:

$$x = a_1 + \frac{1}{a_2 + \frac{1}{x_2}}$$

On poursuit ce processus... On construit ainsi une suite d'entiers $a_1, a_2, \dots, a_n, \dots$ et une suite de réels $x_1, x_2, \dots$ telles que:

$$x = a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{\dots a_n + \frac{1}{x_n}}}}$$

Le processus est fini si x est un rationnel (l'un des x_n est un entier), et infini sinon (aucun des x_n n'est entier).

La quantité ci-dessus est appelée une fraction continue de x .

Les a_n sont appelés quotients partiels de cette fraction continue.

La fraction continue:

$$a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{\dots a_{n-1} + \frac{1}{a_n}}}}$$

est notée $[a_1 / a_2 / \dots / a_n]$ et appelée n -ième réduite de x .

C'est un nombre rationnel r_n dont on notera $r_n = p_n / q_n$ l'expression sous forme de deux entiers premiers entre eux.

Les réduites r_n de x sont des approximations intéressantes de x

On montre que (si x n'est pas un rationnel):

- La suite des réduites de x converge vers x .
- Les suites p_n et q_n convergent vers l'infini.
- x est toujours compris entre deux réduites consécutives.
- l'écart entre x et sa n -ième réduite r_n est tel que:

$$|x - r_n| < \frac{1}{q_{n-1} q_n}$$

FRACTIONS CONTINUES (suite).

=====

Le schéma fonctionnel de 'FRCN' est:



où:

- * x est le réel dont on veut les réduites.
- * n est l'ordre jusqu'auquel on désire connaître les réduites.
- * liste1 est la liste des quotients partiels a_k de x
(jusqu'au rang n)
- * liste2 est la liste des réduites p_k / q_k (sous la forme (p_k, q_k)
jusqu'à p_n / q_n .

'FRCN': (Checksum: # 56779d, Taille: 206.5 octets)

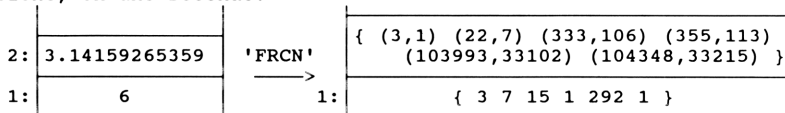
```

"      { }      (1,0)      (0,1)
  → x  n  k  r  s
    "  1  n  START
        x  FLOOR  'k'  OVER  STO+
        x  OVER   -  INV  'x'  STO  r  *
        s  +  r  's'  STO  'r'  STO  r
      NEXT
      n →LIST  k
    "
  "

```

Exemple:

On veut connaître les réduites de π , jusqu'à l'ordre 6. On obtient, en une seconde:

Ce qui signifie que les réduites successives de π sont:

```

[ 3 ] = 3
[ 3 / 7 ] = 3 + 1/7 = 22/7 ≈ 3.14285714286.
[ 3 / 7 / 15 ] = 3 + 1/(1 + 1/15) = 333 / 106 ≈ 3.14150943396.
[ 3 / 7 / 15 / 1 ] = 355 / 113 ≈ 3.14159292035.
[ 3 / 7 / 15 / 1 / 292 ] = 103993 / 33102 ≈ 3.14159265301.
[ 3 / 7 / 15 / 1 / 292 / 1 ] = 104348 / 33215 ≈ 3.14159265392.

```

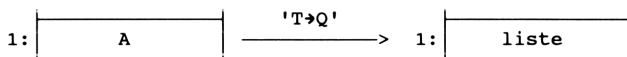
APPROXIMATION RATIONNELLE D'UN TABLEAU REEL OU COMPLEXE.

=====

'T→Q' calcule l'approximation à l'aide de rationnels des différents éléments d'un vecteur ou d'une matrice à coefficients réels ou complexes, à l'aide de l'instruction →Q.

La qualité de l'approximation est fonction du mode d'affichage:
 * Si le mode **n FIX** est actif, l'approximation est exacte sur **n** décimales.
 * Si le mode **STD** est actif, l'approximation est exacte sur les 12 chiffres significatifs)

Le schéma fonctionnel est:



où "liste" contient les éléments résultant des différentes approximations. Dans le cas d'une matrice A, "liste" est formée de sous-listes correspondant à chaque ligne de la matrice A.

Si vous placez la machine en mode **n FIX**, l'entier **n** doit être suffisamment grand pour que l'approximation rationnelle ne soit pas trop grossière.

Il doit cependant tenir compte des éventuelles erreurs d'arrondis qui pourraient affecter les nombres que l'on désire approcher par des rationnels.

Des valeurs comprises entre **n=8** et **n=10** donnent en général des résultats corrects.

'T→Q': (Checksum: # 25770d, Taille: 183.5 octets)

```

"  OBJ→  OBJ→  IF  1  ==  THEN  1  SWAP  END
  →  lig  col
  "  1  lig  FOR  i
      1  col  START
      →Q  col  ROLLD
      NEXT
      col  →LIST
      col  lig  i  -  *  i  +  ROLLD
      NEXT
      IF  lig  1  >  THEN  lig  →LIST  END
  "
  
```

PROGRAMME 'T→Q': EXEMPLES D'UTILISATION.

=====

Exemple 1:

On veut connaître la solution en nombres rationnels du système

$$\begin{cases} 2x + y - z = 1 \\ x - 3y + 5z = -3 \\ 3x + 5y - z = 0 \end{cases}$$

On place [1,-3,0] au niveau 2, et $\begin{bmatrix} 2 & 1 & -1 \\ 1 & -3 & 5 \\ 3 & 5 & -1 \end{bmatrix}$ au niveau 1.

On effectue l'opération / et on trouve la solution, en nombres réels, au niveau 1.

On passe en mode **10 FIX** et on appelle 'T→Q'.

On trouve, en deux secondes, au niveau 1:

{ '5/21' '-(13/42)' '-(5/6)' }

Ce qui prouve que la solution exacte du système ci-dessus est:

x = 5/21, y = -13/42, z = -5/6.

Exemple 2:

On veut connaître l'inverse de la matrice $A = \begin{bmatrix} 1 & 3 & -4 \\ 8 & -2 & 3 \\ -5 & 1 & 7 \end{bmatrix}$.

On place la matrice au niveau 1; on effectue l'opération '1/x'.

On passe en mode **9 FIX** et on appelle le programme 'T→Q'.

En trois secondes environ, on obtient la liste:

{ { '17/222' '25/222' '-1/222' }
{ '71/222' '13/222' '35/222' }
{ '1/111' '8/111' '13/111' } }.

L'inverse de A est donc $\frac{1}{222} * \begin{bmatrix} 17 & 25 & -1 \\ 71 & 13 & 35 \\ 1 & 16 & 26 \end{bmatrix}$.

Exemple 3:

Cet exemple utilise certains programmes du répertoire 'DL'.

On veut connaître de façon précise le développement limité, en zéro, à l'ordre 5, de $\tan(\ln(1+X))$.

On se place dans le répertoire 'DL'.

On place 5 sur la pile, puis on appelle successivement les programmes 'LG' et 'TG'. On obtient ainsi le développement cherché, mais sous forme réelle.

On revient dans le répertoire 'R.C', où on passe en mode **10 FIX**.

On appelle 'T→Q' et on obtient alors la liste:

{ 0 1 '-1/2' '2/3' '-3/4' '11/12' }

Ce qui prouve que le développement limité cherché s'écrit:

$\tan(\ln(1+X)) = X - \frac{1}{2}X^2 + \frac{2}{3}X^3 - \frac{3}{4}X^4 + \frac{11}{12}X^5 + o(X^5)$

ECRITURE D'UN COMPLEXE SOUS LA FORME $a+bj$.

=====

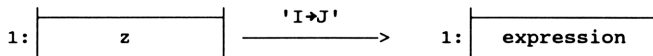
'I→J' permet d'écrire un nombre complexe z sous la forme $a+bj$, où a et b sont deux nombres réels et où j est le nombre complexe de module 1 et d'argument $2\pi/3$.

Le nombre complexe z peut être écrit sous forme d'expression algébrique. En fait, 'I→J' est surtout conçu pour effectuer rapidement des calculs utilisant le nombre j . A cet effet, il est fortement recommandé de créer, dans le répertoire 'R.C', une variable j .

Pour cela, exécutez en séquence:

1 ENTER CHS 3 √ R→C 2 / 'j' STO

Le schéma fonctionnel de 'I→J' est le suivant:



où 'expression' est de la forme $a+jb$, a et b étant deux nombres réels.

'I→J': (Checksum: # 8929d, Taille: 62 octets)

```

«   →NUM   C→R   3   √   /   DUP   2   *
   'j'   *   3   ROLL   +   SWAP   +
»

```

Exemples: (en une seconde).

Ces deux exemples supposent que l'on ait créé la variable j .

```

1: [ '(3+4*j)/(11-2*j)' ] --'I→J'--> 1: [ '.210884353742+.340136054422*j' ]

```

Si on passe en mode 10 FIX et si on exécute →Q, on obtient:
'31/147+50/147*j'

```

1: [ '(2+3*j)^5' ] --'I→J'--> 1: [ '149.000000002+87.0000000012*j' ]

```

Atrement dit: $(2+3*j)^5 = 149+87*j$.

CALCULS ITERATIFS.

=====

'ITER' permet d'effectuer des calculs itératifs d'ordre quelconque.

On se donne une relation de récurrence

$$U_n = F(U_{n-p}, U_{n-p+1}, \dots, U_{n-1})$$

permettant de calculer des "objets" U_n , la suite U étant initialisée par la donnée des p termes initiaux $U_0, U_1, \dots, U_{p-1}$.

Ceci suffit à calculer les termes suivants. C'est ce problème très général que résoud le programme 'ITER'.

Pour que 'ITER' fonctionne, il faut que le répertoire contienne une variable nommée 'LISTE' et ayant le format suivant:

```
{ N Un-p Un-p+1 .... Un-1 F(Un-p, Un-p+1, ..., Un-1) }.
```

Dès l'appel de 'ITER' celui-ci crée dans le répertoire les variables $N, U_0, U_1, \dots, U_{p-2}, U_{p-1}$.

'ITER' est alors suspendu avec présentation de la valeur de U_N .

Si on fait CONT, la valeur suivante U_{N+1} apparaît, le programme 'ITER' étant à nouveau suspendu. On continue ainsi à volonté pour obtenir les valeurs successives de la suite U .

Pour interrompre le programme, il est conseillé de vider la pile avant de faire CONT. Le programme 'ITER' comprend alors que l'utilisateur a terminé, et vide le répertoire de toutes les variables qui avaient été créés.

'ITER': (Checksum: # 15258d, Taille: 360 octets)

```

"  LISTE  SIZE
  "  ""U"  SWAP  →STR  +  OBJ→  "
  →  d  var
  "  LISTE  DUP  1  GET  'N'  STO  2
    0  d  2  -  FOR  i
      GETI  i  var  EVAL  STO
    NEXT
  DROP2
  DO
    d  2  -  var  EVAL  EVAL  EVAL  HALT
  IF  DEPTH  NOT  THEN
    'N'  PURGE
    0  d  2  -  FOR  i
      i  var  EVAL  PURGE
    NEXT
    0  DOERR
  ELSE
    'N'  1  STO+
    1  d  2  -  FOR  i
      i  var  EVAL  RCL  i  1  -  var  EVAL  STO
    NEXT
    d  3  -  var  EVAL  STO
  END
UNTIL  0  END
"
"
```

PROGRAMME 'ITER': EXEMPLES D'UTILISATION.

=====

Exemple 1:

On veut calculer les termes successifs de la suite de terme général u définie par la relation de récurrence:

$$u_n = \frac{u_{n-1}}{2} + \frac{2}{u_{n-1}}$$

et le terme initial $u = 1$.

On place { 1 1 'U0/2+2/U0' } dans 'LISTE' et on appelle 'ITER'.

Au bout d'une seconde (nécessaire aux créations des variables), le programme est suspendu et affiche la valeur de u_1 , c'est-à-dire 2.5; on fait CONT et on obtient la valeur de u_2 , c'est à dire 2.05; les valeurs suivantes sont: $u_3 = 2.0006097561$, $u_4 = 2.00000009292$, puis $u_5 \approx 2$.

On nettoie la pile avant de faire CONT si on veut interrompre le programme en nettoyant le répertoire des variables intermédiaires.

Exemple 2:

On veut calculer les termes successifs de la suite de matrice M définie par la relation de récurrence:

$$M_n = M_{n-1} - n^* \begin{pmatrix} M_{n-2} \end{pmatrix}.$$

et par les termes initiaux

$$M_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad M_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

On place { 2 [[1 0 0 [0 1 0 [0 0 1]]] [[0 1 0 [1 0 1 [0 1 0]]] « U1 U0 TRN N * - » } dans la variable 'LISTE' et on appelle 'ITER'.

On obtient rapidement la valeur de M_2 , c'est à dire $M_2 = \begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix}$

Des appuis successifs sur CONT donnent alors les matrices suivantes de la suite M .

Exemple 3:

On veut calculer les termes successifs de la suite de polynômes définie par la relation de récurrence:

$$P_n(x) = nP_{n-1}(x) - x^*P_{n-1}'(x) + x^2P_{n-2}'(x).$$

et les termes initiaux:

$$P_0(x) = 0, \quad P_1(x) = x \quad \text{et} \quad P_2(x) = x^2.$$

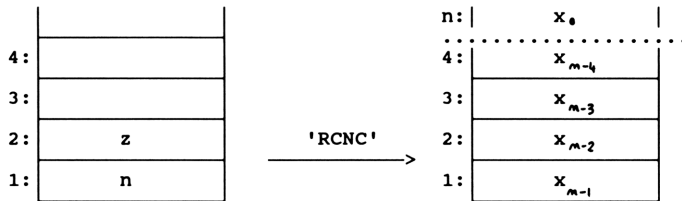
On place, dans la variable 'LISTE':

{ 3 [0] [1 0] [1 0 0] « N U2 U1 U0 POLY → n u2 u1 u0 « u2 n * u1 [-1 0] PRODP ADDP U0 DERIV [1 0 0] PRODP ADDP R.C » }
(on remarque le passage dans le répertoire 'POLY', où on crée des variables locales u0,u1,u2,u3,n et le retour dans le répertoire R.C).

RACINES NIÈMES D'UN NOMBRE COMPLEXE.

=====

'RCNC' calcule les racines n-èmes $x_0, x_1, \dots, x_{n-1}$ d'un nombre complexe z , suivant le schéma fonctionnel:



La formule utilisée est:

si $z = R \exp(i \theta)$, alors $x_k = \sqrt[n]{R} \exp(i (\theta/n + 2k\pi/n))$.

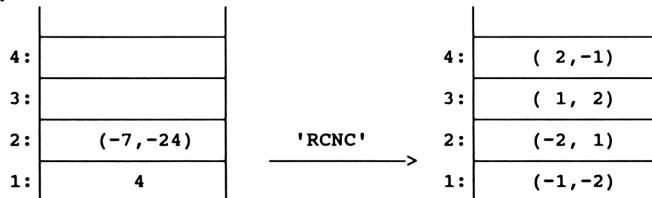
'RCNC': (Checksum: # 5849d, Taille: 81 octets)

```

«  -1  OVER  INV  2  *  ^
→   n  w
«    n  INV  ^
    2  n  START
      DUP  w  *
    NEXT
  »
»

```

Exemple:



LINEARISATION ET OPERATION INVERSE DE LA LINEARISATION.

=====

'TRIG' permet d'exprimer $\cos(t)^n$ et $\sin(t)^n$ en fonction de quantités du type $\cos(pt)$ et $\sin(pt)$: c'est le principe de la linéarisation.

Inversement, 'TRIG' permet d'exprimer $\cos(nt)$ et $\sin(nt)$ en fonction des puissances de $\cos(t)$ et de $\sin(t)$.

Attention: le programme 'TRIG' utilise les programmes 'TCHEB' et 'V→P' devant se trouver dans le répertoire 'POLY'.

Dès l'appel de 'TRIG', le menu devient:

COS [^]	SIN [^]	COS()	SIN()		EXIT
------------------	------------------	-------	-------	--	------

et le programme 'TRIG' est suspendu.

Un appui sur la touche "EXIT" permet de sortir du programme.

Après avoir placé un entier n au niveau 1, un appui sur une des 4 premières touches effectue l'opération qui correspond à:

- * linéarisation de $\cos(t)^n$.
- * linéarisation de $\sin(t)^n$.
- * expression de $\cos(nt)$.
- * expression de $\sin(nt)$.

Exemple: après avoir placé 3 au niveau 1 de la pile, on obtient, au niveau 1 de la pile:

'COS(T)[^]3=(COS(3*T)+3*COS(T))/4' si on a appuyé sur COS[^].

'SIN(T)[^]3=(-SIN(3*T)+3*SIN(T))/4' si on a appuyé sur SIN[^].

'COS(3*T)=4*COS(T)[^]3-3*COS(T)' si on a appuyé sur COS().

'SIN(3*T)=- (4*SIN(T)[^]3+3*SIN(T)' si on a appuyé sur SIN().

Exemple: avec n=15, les résultats sont obtenus en 3 à 9 s. on trouve par exemple:

'SIN(15*T)=- (16384*SIN(T)[^]15)+61440*SIN(T)[^]13-92160*SIN(T)[^]11
+70400*SIN(T)[^]9-28800*SIN(T)[^]7+6048*SIN(T)[^]5
-560*SIN(T)[^]3+15*SIN(T)'

TEXTE DU PROGRAMME 'TRIG'.

=====

```

'TRIG': ( Checksum: # 49045d, Taille: 984.5 octets )
  " 0 0 → n k
  " " n k COMB n k 2 * - "
  " IF DUP THEN 'T' * COS *
  " ELSE DROP 2 / END "
  " 2 n 1 - ^ / "
  " IF 2 MOD THEN NEG END "
  → p1 p2 p3 p4
  " { { "COS~"
    " → n " 'COS(T)' n ^ 0
    0 n 2 / FLOOR FOR k
    p1 EVAL p2 EVAL +
    NEXT p3 EVAL =
    " " }
  { "SIN~"
    " → n " 'SIN(T)' n ^ 0
    0 n 2 / FLOOR
    IF n 2 MOD THEN FOR k
    p1 EVAL 'T' * SIN * k n
    1 - 2 / + p4 EVAL +
    NEXT
    ELSE FOR k
    p1 EVAL p2 EVAL k n
    2 / + p4 EVAL +
    NEXT
    END p3 EVAL =
    " " }
  { "COS()"
    " → n " n 'T' * COS n POLY 1
    TCHEB 'COS(T)' V→P R.C =
    " " }
  { "SIN()"
    " → n " n 'T' * SIN
    IF n 2 MOD THEN
    n 1 POLY TCHEB n 1 - 2 /
    p4 EVAL 'SIN(T)' V→P R.C
    ELSE
    IF n DUP THEN
    1 - 2 POLY TCHEB n 2 /
    1 + p4 EVAL 'SIN(T)' V→P
    R.C 'COS(T)' *
    END
    END =
    " " }
  { }
  { "EXIT" " CONT " }
}
TMENU HALT 2 MENU
" " "

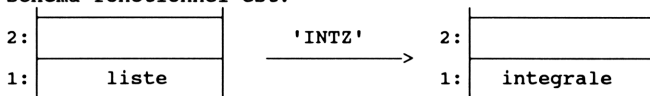
```

INTEGRATION LE LONG D'UN CHEMIN DU PLAN COMPLEXE.

=====

'INTZ' calcule $\int_{\Gamma} f(z)dz$, où f est une fonction de la variable complexe z et où Γ est un arc paramétré $x=x(t)$, $y=y(t)$, $a \leq t \leq b$.

Le schéma fonctionnel est:



où "liste" est une liste contenant les informations relatives au calcul à effectuer, "integrale" est une valeur approchée du résultat (c'est un nombre complexe).

La précision du calcul est donné par le mode d'affichage en cours:

En mode **STD**, le calcul est effectué avec toute la précision possible (Ça risque d'être long...).

En mode **n FIX**, il est effectué avec une précision de n décimales.

Le format de "liste" est le suivant: { F Z A B }, avec:

F: expression de la fonction f , par rapport à la variable 'Z'.

Z: expression symbolique, par rapport à la variable 'T', donnant la représentation paramétrique de l'arc Γ .

L'évaluation de Z doit donner un nombre complexe représentant les coordonnées du point $(x(t), y(t))$ de l'arc.

A,B: valeurs extrémales du paramètre T.

'INTZ': (Checksum: # 44859d, Taille: 133 octets)

```

"  EVAL  ROT  DUP  'Z'  STO
   'T'   DUP  PURGE  Ø  4  ROLL  'T'  SHOW
   *     3  DUPN  RE  'T'  ∫  →NUM  4  ROLLD
   IM  'T'  ∫  →NUM  R→C  'Z'  PURGE
"
  
```

EXEMPLES D'UTILISATION DU PROGRAMME 'INTZ'.

=====

Exemple 1:

Calculer $\int_{\Gamma} (1/Z) dZ$, où Γ est l'arc orienté
 $X(T) = \exp(T)$, $Y(T) = T \cdot \ln(T)$, $1 \leq T \leq 2$.

On passe par exemple en mode 6 FIX.

On met { '1/Z' 'EXP(T)+i*T*LN(T)' 1 2 } au niveau 1 de la pile et on appelle le programme 'INTZ'.

Au bout de 25 secondes, on obtient:

1: (1.017297 , 0.185459)

Exemple 2:

On sait que $\int_{\Gamma} f(Z) dZ$ ne dépend que des extrémités et de l'orientation du chemin Γ , à condition que la déformation de Γ soit continue et s'inscrive dans un domaine du plan complexe où f est holomorphe.

On le constate ici en intégrant $f(Z) = \exp(Z)$ entre le point A(1,0) et le point B(0,1):

- 1) le long du segment joignant A à B.
- 2) le long de l'arc du cercle unité qui joint A à B.

On passe par exemple en mode 5 FIX.

On met au niveau 1 de la pile:

Dans le premier cas: { 'EXP(Z)' '1-T+i*T' 0 1 },

Dans le 2ème cas: { 'EXP(Z)' 'EXP(i*T)' 0 'π/2' },

et on appelle 'INTZ'.

Dans le premier cas, on obtient, en 11 secondes:

1: (-2.17798, 0.84147)

Dans le second cas, on obtient, en 27 secondes:

1: (-2.17798, 0.84147)

Le résultat exact est:

$\exp(i) - \exp(1) \approx (-2.17797952259, .841470984808)$.

POLYNOMES

Le répertoire 'POLY' contient les programmes s'appliquant aux polynômes à une indéterminée (à coefficients réels ou complexes).

Les programmes qui suivent sont parmi les plus utiles, et ils libèrent l'utilisateur de tâches fastidieuses (et où les erreurs de calculs sont nombreuses).

Les polynômes seront ici représentés par le vecteur de leurs composantes suivant les puissances décroissantes de l'indéterminée:

Le polynôme $P = 3X^7 - 2X^6 + X^3 - 2X$ est ainsi représenté par le vecteur $[3 \ -2 \ 0 \ 0 \ 1 \ 0 \ -2 \ 0]$.

C'est sous cette forme que devront être écrits les polynômes qui sont arguments d'un programme, et c'est sous cette forme qu'ils seront obtenus s'ils sont résultats d'un programme.

Exceptions:

- Les polynômes arguments du programme 'DIVPC' (division suivant les puissances croissantes) devront être représentés par le vecteur de leurs composantes selon les puissances croissantes de l'indéterminée.
- Dans le programme 'RENV' (qui effectue la bascule puissances croissantes/ puissances décroissantes).
- Dans le programme $V \rightarrow P$ (transformation d'un polynôme écrit sous forme 'vecteur' en une notation plus algébrique).

Le répertoire 'POLY' est très complet. On y trouve:

- 'ADDP' : Addition de deux polynômes.
- 'PRODP' : Produit de deux polynômes.
- 'EXPO' : Elévation d'un polynôme à une puissance entière.
- 'DIV' : Division euclidienne de deux polynômes.
- 'DIVPC' : Division, suivant les puissances croissantes, de deux polynômes.
- 'COMP' : Composition de deux polynômes.
- 'TRNS' : Translation d'un polynôme.

.../...

'DEG2' : Racines réelles ou complexes d'un polynôme de degré 2.
'DEG3' : Racines réelles ou complexes d'un polynôme de degré 3.
'DEG4' : Racines réelles ou complexes d'un polynôme de degré 4.
'BRST' : Racines réelles ou complexes d'un polynôme de degré quelconque (méthode itérative de Bairstow).
'VALP' : Valeur d'un polynôme en un point.
'DERIV' : Dérivée d'un polynôme.
'PRIM' : Primitive (s'annulant en zéro) d'un polynôme.
'INTP' : Intégrale d'un polynôme sur un segment [a,b].
'PGDP' : PGCD de deux polynômes.
'PPMP' : PPCM de deux polynômes.
'TCHEB' : Calcule les polynômes de Tchébitchev de première ou de seconde espèce.
'V→P' : Transforme un polynôme (écrit sous forme de vecteur) en une notation algébrique traditionnelle.
'PPCS' : Primitive, sous forme symbolique, d'une expression du type: $P(x)\cos(ax)+Q(x)\sin(ax)$, où P et Q sont deux polynômes.
'PPEX' : Primitive, sous forme symbolique, d'une expression du type: $P(x)\exp(ax)$, où P est un polynôme.
'DEVLP' : Développement d'une expression polynômiale.
'DEVPP' : Développement d'un produit de puissances de polynômes.
'ABCUV' : Recherche d'une solution de l'équation $AU+BV=C$, où A,B,C sont trois polynômes donnés, et où U,V sont deux polynômes inconnus.
'ELMG','ELMD' sont 2 routines utilisées lors des opérations sur les polynômes et dont l'objet est de neutraliser certaines erreurs d'arrondi.

ADDITION DE DEUX POLYNÔMES.

=====

'ADDP' effectue la somme de deux polynômes P et Q suivant le schéma:



N.B: 'ADDP' est utile si les longueurs des vecteurs représentant P et Q sont différentes, ou si on ne connaît pas à priori ces longueurs (sinon il vaut mieux utiliser +).

N.B: 'ADDP' appelle le programme 'ELMG', qui supprime les éventuels zéros à gauche du vecteur résultat dans le cas où on a additionné des polynômes de même degré (voir exemple 2).

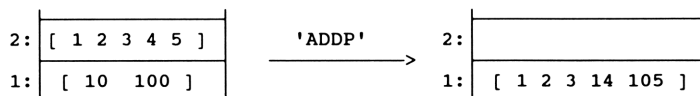
'ADDP': (Checksum: # 40188d, Taille: 151 octets)

```

«   IFERR + ELMG THEN
      OVER SIZE 1 GET OVER SIZE 1 GET -
      IF DUP 0 < THEN ABS ROT SWAP END
      → a d
      «   1 d START 0 NEXT
          a OBJ→ 1 GET d + →ARRY +
      »
    »
  END
»

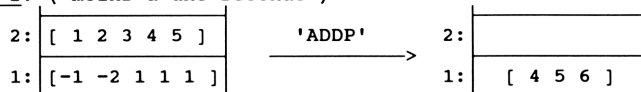
```

Exemple 1: (moins d'une seconde)



(car si $P = X^4 + 2X^3 + 3X^2 + 4X + 5$ et $Q = 10X + 100$, alors
 $P+Q = X^4 + 2X^3 + 3X^2 + 14X + 105$)

Exemple 2: (moins d'une seconde)



Ici on aurait pu tout simplement utiliser l'opération +, mais le résultat aurait été [0 0 4 5 6].

PRODUIT DE DEUX POLYNOMES.

=====

'PRODP' effectue le produit de deux polynômes A et B suivant le schéma:



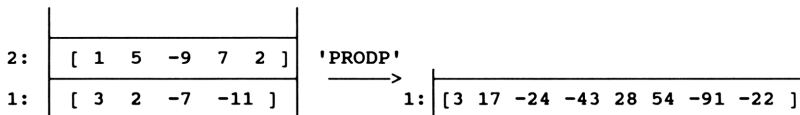
'PRODP': (Checksum: # 23183d, Taille: 197.5 octets)

```

«  DUP  SIZE  1  GET
  →  a  da
    «  DUP  DUP  0  CON  DUP  SIZE  1  GET  DUP
      da  +  1  -  1  →LIST
      →  b  c  db  dim
      «  { 1 }  db  +  RDM
        1  db  START
          a  V→  c  V→
          NEXT
          db  DROPN
          db  dim  +  →ARRY  *  dim  RDM
        »
      »
    »
  »

```

Exemple: (en 1 seconde).



car si $P = X^4 + 5X^3 - 9X^2 + 7X + 2$ et $Q = 3X^3 + 2X^2 - 7X - 11$,
alors

$PQ = 3X^7 + 17X^6 - 24X^5 - 43X^4 + 28X^3 + 54X^2 - 91X - 22$

PUISSANCES D'UN POLYNÔME.

=====

'EXPO' calcule la puissance n-ième du polynôme A (où n est un entier positif ou nul). Le schéma fonctionnel est:



N.B.: le programme 'EXPO' appelle le programme 'PRODP'.

'EXPO': (Checksum: # 7170d, Taille: 168 octets)

```

«   →   a   n
    «   1   DUP   →ARRY
        WHILE n   0   >
          REPEAT
            IF n   2   MOD   THEN a   PRODP   END
            n   2   /   FLOOR   'n'   STO
            IF n   THEN a   DUP   PRODP   'a'   STO   END
          END
        »
    »

```

Exemple:

On veut calculer (2X² -3X +7)⁵.

On place donc le vecteur [2 -3 7] au niveau 2, l'entier 5 au niveau 1, et on appelle 'EXPO'.

Le résultat est obtenu en 3 secondes. On obtient le vecteur:

[32 -240 1280 -4440 12290 -25443 43015 -54390 54880 -36015 16807],

ce qui signifie que:

(2X² -3X +7)⁵ = 32X¹⁰ - 240X⁹ + 1280X⁸ - 36015X + 16807.

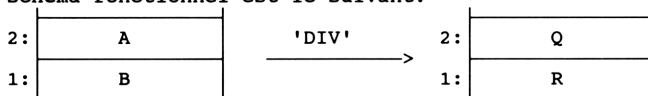
DIVISION DE DEUX POLYNOMES. (division euclidienne)

=====

'DIV' effectue la division euclidienne (c'est-à-dire suivant les puissances décroissantes) d'un polynôme A par un polynôme non nul B.

Cette division s'écrit $A = BQ + R$, où Q est le quotient de la division et où R est le reste (le degré de R est strictement inférieur à celui de B)

Le schéma fonctionnel est le suivant:



N.B: le programme 'DIV' appelle le programme 'ELMG'.

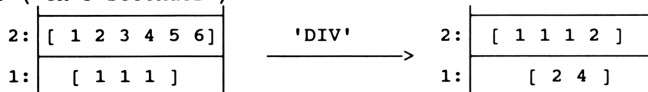
'DIV': (Checksum: # 19185d, Taille: 379 octets)

```

"  ELMG  DUP  1  GET
    "  DUP  IM  ABS  IF  NOT  THEN  RE  END  "
    →  d  reel
    "  DUP  SIZE  1  GET  ROT  DUP  SIZE  1  GET
    →  b  tb  a  ta
    "  a
      IF  tb  1  ==  THEN  d  /  0  1  →ARRY
      ELSE IF  ta  tb  <  THEN  0  1  →ARRY  SWAP
      ELSE
        ta  tb  FOR  i
          DUP  1  GET  d  /  SWAP  OVER
          b  i  {  }  +  RDM  *  -  OBJ→
          1  GET  1  -  →ARRY  SWAP  DROP
        -1  STEP
      ELMG
      →  r
      "  ta  tb  -  1  +  →ARRY  reel  EVAL  r
    "
    reel  EVAL
  END
END
"
"

```

Exemple: (en 3 secondes)



Ce qui signifie que dans la division du polynôme

$A = X^5 + 2X^4 + 3X^3 + 4X^2 + 5X + 6$ par le polynôme $B = X^2 + X + 1$, le quotient est

$Q = X^3 + X^2 + X + 2$, et le reste est $R = 2X + 4$.

DIVISION DE DEUX POLYNOMES SUIVANT LES PUISSANCES CROISSANTES.

=====

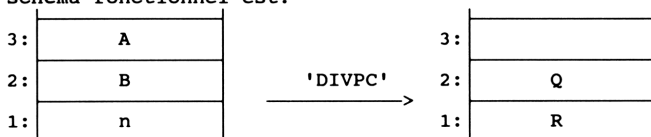
'DIVPC' effectue la division suivant les puissances croissantes, à l'ordre n (entier positif ou nul), d'un polynôme A par un polynôme B (le coefficient constant de celui-ci étant non nul).

Cette division s'écrit:

$$A = BQ + X^{m+1}R,$$

où Q est le quotient (de degré inférieur ou égal à n) et où R est le reste (On appelle également reste le polynôme $X^{m+1}R$).

Le schéma fonctionnel est:



ATTENTION:

Contrairement à la plupart des programmes du répertoire 'POLY', les polynômes A et B doivent être représentés par le vecteur de leurs coefficients suivant les puissances croissantes de l'indéterminée. Par exemple, [1 5 -4 7] représente $1 + 5X - 4X^2 + 7X^3$. Les polynômes Q et R sont obtenus dans le même format.

N.B. Le programme 'DIVPC' appelle le programme 'DIV'.

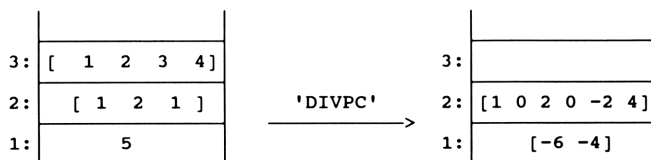
'DIVPC': (Checksum: # 6051d, Taille: 53.5 octets)

```

«   OVER   SIZE   1   GET   +   1   →LIST
   ROT    SWAP   RDM   SWAP   DIV
»

```

Exemple: (en 3 secondes)



Ce qui signifie que:

$$1 + 2X + 3X^2 + 4X^3 = (1 + 2X + X^2)(1 + 2X^2 - 2X^4 + 4X^5) + X^6(-6 - 4X)$$

COMPOSITION DE DEUX POLYNOMES.

=====

'COMP' effectue le calcul du composé P(Q) de deux polynômes P et Q selon le schéma fonctionnel:



N.B: 'COMP' appelle les programmes 'PRODP' et 'RENV'

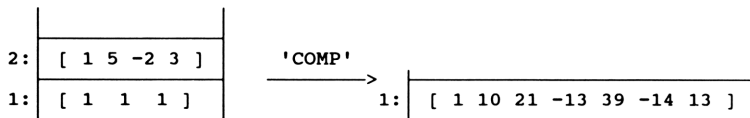
'COMP': (Checksum: # 29324d, Taille: 126 octets)

```

«   RENV  SWAP
  → P
  «   IF   DUP   SIZE  {1}  ≠  THEN
        OBJ→ 1  GET   SWAP  1  →ARRY
        2    ROT   START
          P  PRODP  DUP   SIZE
          DUP2 GET   4  ROLL  +  PUT
        NEXT
      END
    »
  »

```

Exemple: (en 2 secondes)



Ce qui signifie qu'en posant:

$P(X) = X^2 + X + 1$ et $Q(X) = X^3 + 5X^2 - 2X + 3$, alors

$P(Q(X)) = Q(X)^2 + Q(X) + 1$
 $= X^6 + 10X^5 + 21X^4 - 13X^3 + 39X^2 - 14X + 13$

TRANSLATE D'UN POLYNÔME.

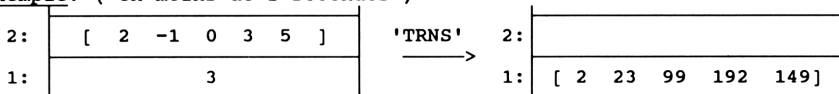
=====

'TRNS' transforme un polynôme $x \rightarrow P(x)$ en le polynôme $x \rightarrow Q(x)=P(x+a)$, suivant le schéma fonctionnel:



```
'TRNS': ( Checksum: # 40631d, Taille: 145.5 octets )
"  OVER  SIZE  1  GET
  →  a  n
    "  IF  n  1  ≠  THEN
      n  2  FOR  i
        2  i  FOR  j
          DUP  j  GET  OVER  j  1  -
          GET  a  *  +  j  SWAP  PUT
        NEXT
      -1  STEP
    END
  "
"
```

Exemple: (en moins de 2 secondes)



car si $P(X) = 2X^4 - X^3 + 3X + 5$,
alors $Q(X) = P(X+3) = 2X^4 + 23X^3 + 99X^2 + 192X + 149$

Remarque:

Le programme 'TRNS' donne également la décomposition du polynôme $x \rightarrow P(x)$ suivant les puissances de $(x-a)$.

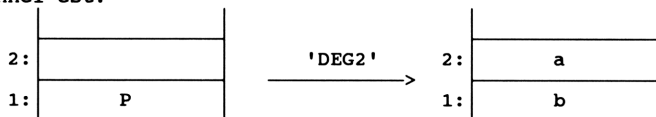
Ainsi l'exemple précédent peut-il être interprété en disant que:
si $P(X) = 2X^4 - X^3 + 3X + 5$,
alors $P(X) = 2(X-3)^4 + 23(X-3)^3 + 99(X-3)^2 + 192(X-3) + 149$

On peut encore dire que le programme 'TRNS' effectue le changement de variable $y=(x-a)$ dans le polynôme $P(x)$ et que le résultat est présenté comme un polynôme de la variable y .

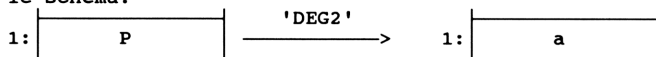
ZEROS D'UN POLYNOME DE DEGRE 2.

=====

'DEG2' calcule les deux zéros réels ou complexes a et b, d'un polynôme P de degré 2, à coefficients réels ou complexes. Le schéma fonctionnel est:



'DEG2' calcule également le zéro a d'un polynôme P de degré 1 suivant le schéma:



N.B: 'DEG2' appelle le programme 'ELMG'.

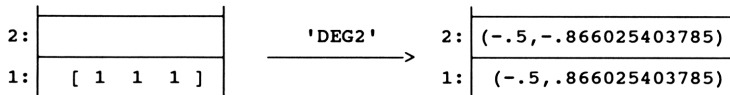
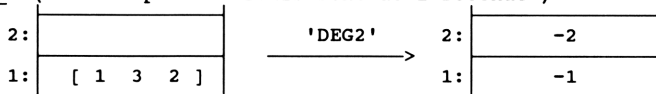
'DEG2': (Checksum: # 4415d, Taille: 351.5 octets)

```

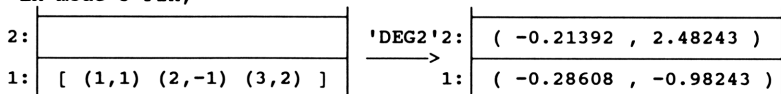
«      ELMG  DUP  SIZE
  →      p      s
  «      IF  s { 3 } == THEN
        'p(2)^2-4*p(1)*p(3)' EVAL ✓
        → rd
        «      '-(p(2)+rd)/2/p(1)' EVAL
          '(-p(2)+rd)/2/p(1)' EVAL
        »
      ELSE
        IF  s { 2 } == THEN
          p      v→ SWAP
          IF  DUP THEN / NEG ELSE DROP2 END
        END
      END
    »
  »

```

Exemples: (les temps de calculs sont de 1 seconde)



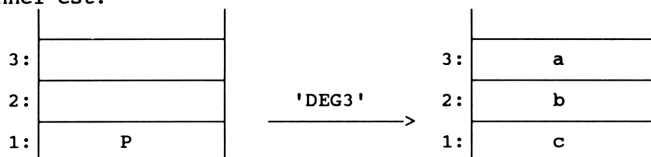
En mode 5 FIX,



ZEROS D'UN POLYNOME DE DEGRE 3.

=====

'DEG3' calcule les trois zéros réels ou complexes a b et c, d'un polynôme P de degré 3, à coefficients réels ou complexes. Le schéma fonctionnel est:



N.B: Celles des racines a,b ou c qui sont réelles sont cependant obtenues sous forme complexe (avec une partie imaginaire nulle ou presque compte tenu des erreurs d'arrondis).

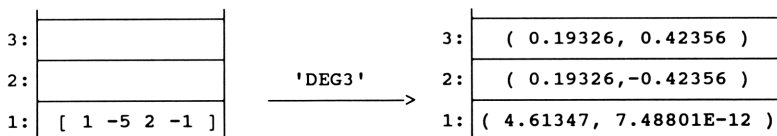
'DEG3': (Checksum: # 58445d, Taille: 401.5 octets)

```

«   DUP   1   GET   /   V→
→   a   b   c
«   DROP   'b-a^2/3'   EVAL   3   /
    '2*a^3/27-a*b/3+c'   EVAL   2   /
→   p   q
«   IF   p   ABS   THEN
    q   p   √   p   *   /   NEG   ASINH
→   z
    «   0   2   FOR   k
        '2*√p* SINH((z+2*i*k*π)/3)-a/3'   →NUM
    NEXT
    »
    ELSE
        0   2   FOR   k
            '(-2*q)^(1/3)*EXP(2*i*k*π/3)-a/3'   →NUM
        NEXT
    END
»
»
»

```

Exemple: (en Mode 5 FIX pour le résultat, et en 2 secondes)



Dans ce cas particulier, le polynôme P possède 2 racines complexes conjuguées, et une racine réelle égale sensiblement à 4.61347

ZEROS D'UN POLYNOME DE DEGRE 4.

=====

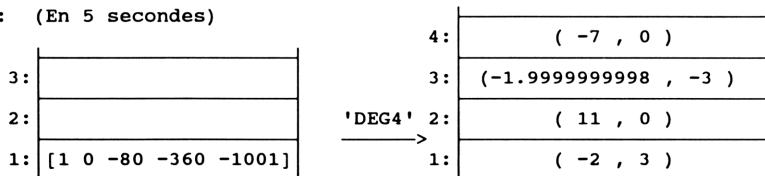
'DEG4' calcule les quatre zéros réels ou complexes a b c et d, d'un polynôme P de degré 4, à coefficients réels ou complexes. Le schéma fonctionnel est:



N.B: 'DEG4' appelle les programmes 'DEG2' et 'DEG3'. Les racines réelles sont données sous forme complexe, avec une partie imaginaire nulle (aux erreurs d'arrondi près).

```
'DEG4': ( Checksum: # 33018d, Taille: 627.5 octets )
«   DUP 1 GET / V→
  → a b c d
  «   DROP 1 b NEG 'a*c-4*d' EVAL 'd*(4*b-a*a)-c*c' EVAL
    4 →ARRY DEG3
    3 →ARRY DUP {3} 'a*a/4-b' EVAL CON + 1
    → t j
    « IF 'ABS(t(2)) > ABS(t(1))' THEN 2 'j' STO END
      IF 'ABS(t(3)) > ABS(t(j))' THEN 3 'j' STO END
      t j GET √ SWAP j GET
      → w y
      « IF w (0,0) == THEN
        a 4 / NEG DUP DUP2
      ELSE
        0 w 'a*y/2-c' EVAL w / 2 / 3
        →ARRY 1 a 2 / y 2 / 3 →ARRY
        « v1 v2 + DEG2
          v1 v2 - DEG2
        »
      »
    »
  »
»
```

Exemple: (En 5 secondes)

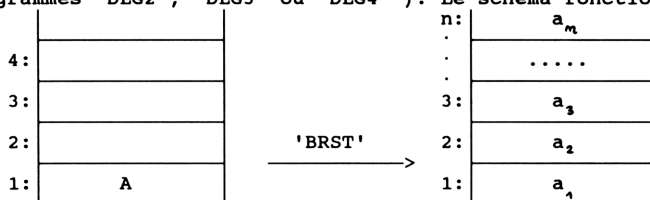


Les zéros de P sont ici -7, 11, -2-3i, et -2+3i.

ZEROS D'UN POLYNOME PAR LA METHODE DE BAIRSTOW.

=====

'BRST' calcule une valeur approchée des zéros réels ou complexes, $a_1, a_2, a_3, \dots, a_m$, d'un polynôme $A(x)$ à coefficients réels ou complexes, de degré n (si $n < 5$, 'BRST' se contente d'appeler l'un des programmes 'DEG2', 'DEG3' ou 'DEG4'). Le schéma fonctionnel est:



N.B: 'BRST' appelle les programmes 'DEG2', 'DEG3', 'DEG4', et 'ELMG'.

Le principe est le suivant:

On cherche deux complexes s et p tels que $A(x)$ soit divisible par $x^2 - sx - p$. La recherche de s et de p utilise un algorithme itératif de Newton.

On part donc d'une valeur initiale $[s_m, p_m]$ puis on construit une suite $[s_m, p_m]$, qui en principe converge vers une solution $[s, p]$.

Si $[s, p]$ est trouvé, le polynôme A s'écrit $A(x) = (x^2 - sx - p)B(x)$, où B est un polynôme de degré $n-2$. On place alors les deux zéros de $x^2 - sx - p$ sur la pile (utilisation de 'DEG2') puis on applique le même procédé au polynôme B .

Le schéma s'arrête dès que le polynôme obtenu est de degré inférieur ou égal à 4, auquel cas il vaut mieux appeler 'DEG2', 'DEG3' ou 'DEG4'.

Le programme 'BRST' est suspendu à chaque fois que deux nouveaux zéros de A sont placés sur la pile. L'utilisateur fait CONT s'il veut relancer la recherche d'autres zéros.

Lorsque le programme 'BRST' est suspendu, on peut modifier la variable '**eps**' qui contient par défaut $1E-6$ et qui sert à arrêter plus ou moins vite les itérations, et la variable '**max**' qui contient par défaut 20 (nombre d'itérations maximum à faire avant que le processus ne soit considéré comme divergent).

Le processus peut en effet diverger (ou converger lentement). On s'en rend compte lorsque le programme est suspendu sans que deux nouveaux zéros de A n'aient été déposés sur la pile. On peut alors modifier la variable '**v**' qui contient la valeur actuelle de $[s_m, p_m]$ (et qui est un **vecteur** de deux éléments) avant de relancer les itérations par CONT. On peut très bien relancer les itérations sans rien modifier.

Attention: si on effectue des modifications de variables pendant que le programme 'BRST' est suspendu, on doit prendre garde à ce que la pile retrouve son aspect au moment de la suspension, avant de faire CONT. Pour les noms de variables '**eps**', '**max**', '**v**', les minuscules sont obligatoires.

TEXTE DU PROGRAMME 'BRST'.

=====

'BRST': (Checksum: # 16338d, Taille: 723.5 octets)

```

«   DUP SIZE 1 GET 0 0 0 0 0 0 0 0 .000001 20
    → a n v s p b d k eps max

«   IF n 6 ≥ THEN

      1 DUP 2 →ARRY 'v' STO

    DO
      0 'k' STO

    DO
      v V→ 'p' STO 's' STO 0
      a 1 GET
      2 n FOR i
        DUP s * 3 PICK p
        * + a i GET +
      NEXT
      n →ARRY 'b' STO DROP v 0 0
      b 1 GET
      2 n 1 - FOR i
        ROT DROP DUP s * 3 PICK
        p * + b i GET +
      NEXT
      SWAP DUP 4 ROLL { 2 2 } →ARRY
      INV b n 1 - GET b n GET 2 →ARRY *
      DUP RNRM v RNRM / 'd' STO
      - 'v' STO 'k' 1 STO+
    UNTIL d eps < k max ≥ OR END

    IF k max < THEN
      1 s NEG p NEG 3 →ARRY
      DEG2 'n' 2 STO-
      b n 1 →LIST RDM 'a' STO
    END

    HALT

    UNTIL n 6 < k max < AND END

  END

a ELMG "DEG" OVER SIZE 1 GET 1 -
2 MAX →STR + 1 4 SUB OBJ→
»
»

```

EXEMPLE D'UTILISATION DU PROGRAMME 'BRST'.

=====

On cherche les zéros du polynôme
 $P(X) = X^{10} + 2X^9 + 3X^8 + 4X^7 + 5X^6 + 6X^5 + 7X^4 + 8X^3 + 9X^2 + 10X + 11$.
 On place donc le vecteur [1 2 3 4 5 6 7 8 9 10 11] au
 niveau 1 de la pile et on appelle 'BRST'.

Au bout de 32 s, on obtient sur la pile deux zéros complexes
 conjugués de P, à savoir :
 $a_1 = (-1.26463096509, -0.357261654484)$
 et $a_2 = (-1.26463096509, 0.357261654484)$.

On fait alors CONT et au bout de 21 s, on trouve à nouveau
 deux zéros complexes conjugués:
 $a_3 = (0.442765764928, -1.17374073066)$
 et $a_4 = (0.442765764928, 1.17374073066)$.

On poursuit par CONT et au bout de 23 s, on trouve encore deux
 zéros complexes conjugués:
 $a_5 = (-0.246722626138, -1.26288540248)$
 et $a_6 = (-0.246722626138, 1.26288540248)$.

On poursuit par CONT et au bout de 7 s, on obtient les quatre
 derniers zéros de P, qui sont deux à deux complexes conjugués. Le
 programme 'BRST' est alors terminé. Les quatre derniers zéros ont été
 obtenus par 'DEG4' (appelé par le programme 'BRST').
 Ces zéros sont
 $a_7 = (-0.88465843109, -0.959966681655)$
 $a_8 = (-0.88465843109, 0.959966681655)$
 $a_9 = (0.95324625739, 0.72511051529)$
 et $a_{10} = (0.95324625739, -0.72511051529)$

Il est important de pouvoir contrôler la précision des résultats
 obtenus.

Dans le cas de zéros réels (s'il y en a), on peut améliorer la
 précision en transformant le polynôme P (écrit sous forme de vecteur)
 en sa notation algébrique traditionnelle ' $X^{10} + 2X^9 + 3X^8 + \dots + 10X + 11$ ' (à l'aide du programme V→P) et en utilisant le programme
 SOLVR de la HP48 en partant du zéro approché qu'on a obtenu par
 'BRST'.

Dans le cas général, on peut utiliser l'approximation (où a est
 un zéro exact de P et où $\bar{a}$ est la valeur approchée trouvée, et à
 condition que a soit un zéro simple):

$$| P(a) - P(\bar{a}) | \approx | a - \bar{a} | | P'(\bar{a}) |$$

et donc:

$$| a - \bar{a} | \approx | P(\bar{a}) | / | P'(\bar{a}) |$$

Pour effectuer ce calcul, on utilise les programmes 'DERIV'
 (dérivation de P) et 'VALP' (calcul des valeurs de P et de P' en $\bar{a}$).

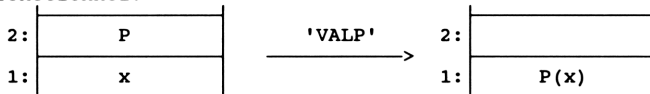
Pour a_{10} , on trouve ainsi $| a_{10} - \bar{a}_{10} | \approx 3.18E-9$ et donc
 certainement $| a_{10} - \bar{a}_{10} | \leq 5E-9$

Pour a_8 , on trouve	$ a_8 - \bar{a}_8 $	$\approx 6.01E-9$
Pour a_6 , on trouve	$ a_6 - \bar{a}_6 $	$\approx 9.05E-9$
Pour a_4 , on trouve	$ a_4 - \bar{a}_4 $	$\approx 6.33E-9$
Pour a_2 , on trouve	$ a_2 - \bar{a}_2 $	$\approx 1.48E-10$

VALEUR D'UN POLYNOME EN UN POINT.

=====

'VALP' calcule la valeur du polynôme P au point x, suivant le schéma fonctionnel:



N.B: le programme 'VALP' appelle le programme 'RENV'.

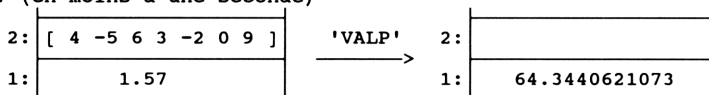
'VALP': (Checksum: # 47185d, Taille: 67.5 octets)

```

"      →  X
      "  RENV  OBJ→  1  GET  0  1  ROT
      START  X  *  +  NEXT
"

```

Exemple: (en moins d'une seconde)



On peut aussi utiliser la fonction EVAL du HP48 en plaçant '4*X^6 -5*X^5 +6*X^4 +3*X^3 -2*X^2 +9' au niveau 1, en mettant 1.57 dans X, et en appelant EVAL. EVAL est alors plus rapide que 'VALP', mais ça ne compense pas le fait d'avoir à rentrer le polynôme sous forme algébrique.

INVERSION DE L'ORDRE DES COMPOSANTES D'UN VECTEUR.

=====

'RENV' inverse l'ordre des composantes du vecteur situé au niveau 1 de la pile. Par exemple (en moins d'une seconde):



'RENV': (Checksum: # 55757d, Taille: 68.5 octets)

```

"      OBJ→  1  GET
      →  n
      "  1  n  FOR  i  i  ROLL  NEXT  n  →ARRY
"

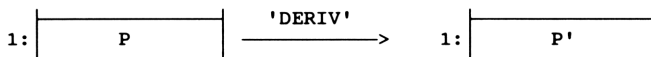
```

'RENV' est utile quand on veut présenter un polynôme suivant ses puissances ou croissantes, ou décroissantes (programme DIVPC).

DERIVATION D'UN POLYNOME.

=====

'DERIV' calcule le polynôme dérivé P' de P. Le schéma fonctionnel est:



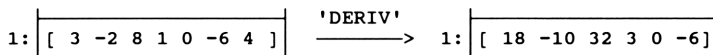
'DERIV': (Checksum: # 27163d, Taille: 113.5 octets)

```

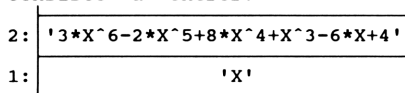
«   OBJ-> 1   GET  1   -
    → n
    «   DROP
        IF  n   THEN
            1   n   FOR  i   i   *  n   ROLLD  NEXT  n
        ELSE  0   1   END
        →ARRY
    »
»

```

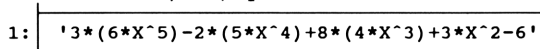
Exemple: (en moins d'une seconde)

Remarque:

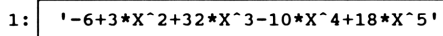
Cette méthode est beaucoup plus rapide que celle qui consiste à entrer:



puis à faire 'd/dx', qui donne au bout de 5 secondes:



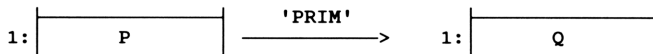
Il faut alors passer dans le menu ALGEBRA, faire COLCT pour obtenir au bout de 8 nouvelles secondes:



PRIMITIVATION D'UN POLYNOME.

=====

'PRIM' calcule la primitive Q s'annulant en 0 du polynôme P. Le schéma fonctionnel est:



'PRIM': (Checksum: # 61049d, Taille: 83 octets)

```
«   OBJ→  1   GET
    →  n
    «   1   n   FOR   i   i   /   n   ROLLD   NEXT
        0   n   1   +   →ARRY
    »
»
```

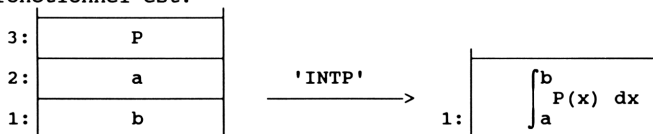
Exemple: (en moins d'une seconde)



INTEGRALE D'UN POLYNOME SUR UN SEGMENT.

=====

'INTP' calcule l'intégrale du polynôme P sur le segment [a,b]. Le schéma fonctionnel est:

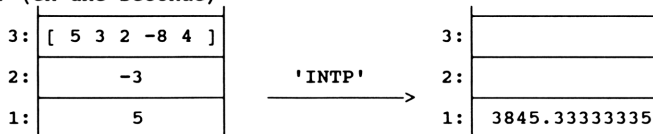


N.B: Le programme 'INTP' appelle les programmes 'PRIM' et 'VALP'

'INTP': (Checksum: # 28489d, Taille: 74 octets)

```
«   →  a   b
    «   PRIM   DUP   b   VALP   SWAP   a   VALP   -
    »
»
```

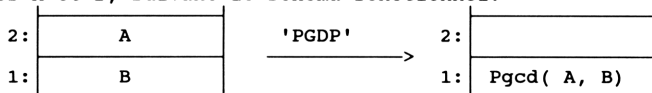
Exemple: (en une seconde)



P.G.C.D. DE DEUX POLYNOMES.

=====

'PGDP' calcule le pgcd (plus grand commun diviseur) de deux polynômes A et B, suivant le schéma fonctionnel:



Le polynôme obtenu est normalisé (terme de plus haut degré = 1).

N.B: le programme 'PGDP' appelle le programme 'DIV'.

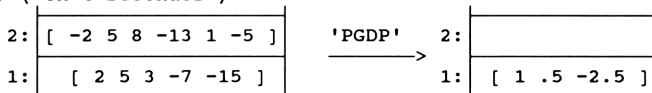
'PGDP': (Checksum: # 54781d, Taille: 65 octets)

```

"  WHILE  DUP  ABS
    REPEAT
      SWAP  OVER  DIV  SWAP  DROP
    END
  DROP  DUP  1  GET  /
"

```

Exemple: (en 4 secondes)

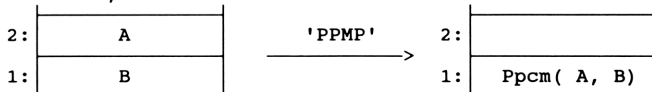


(sur cet exemple le coeff .5 du pgcd est obtenu avec une erreur d'arrondi égale à 2E-12)

P.P.C.M. DE DEUX POLYNOMES.

=====

'PPMP' calcule le ppcm (plus petit commun multiple) de deux polynômes A et B, suivant le schéma fonctionnel:



Le polynôme obtenu est normalisé (terme de plus haut degré = 1).

N.B: 'PPMP' appelle les programmes 'PGDP', 'DIV', 'PRODP'.

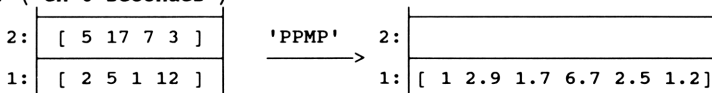
'PPMP': (Checksum: # 28888d, Taille: 56 octets)

```

"  DUP2
    PGDP  DIV  DROP  PRODP  DUP  1  GET  /
"

```

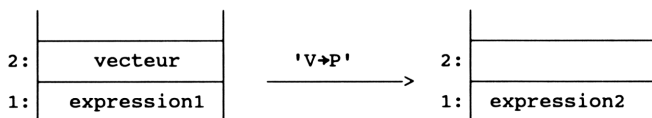
Exemple: (en 6 secondes)



PASSAGE DE LA FORME VECTEUR A LA FORME ALGEBRIQUE.

=====

'V→P' transforme un vecteur (représentant un polynôme P écrit suivant les puissances décroissantes) en une expression algébrique. Le schéma est le suivant:



où "vecteur" est le vecteur représentant le polynôme P, où "expression1" est une expression algébrique devant se substituer à l'indéterminée du polynôme P, et où "expression2" est l'expression algébrique ainsi obtenue pour le polynôme P.

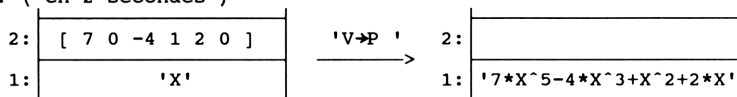
Si par exemple "expression1" est égale à 'X', on obtient l'écriture algébrique traditionnelle du polynôme P.

'V→P': (Checksum: # 44891d, Taille: 117.5 octets)

```

"      → v
  "      DUP SIZE { } + 1 GET
        → p n
        " 0 1 n FOR i
          p i GET v n i - ^ * +
          NEXT
        "
      "
    "
  
```

Exemple: (en 2 secondes)



On peut remplacer 'X' par d'autres expressions algébriques, comme par exemple 'Y', 'x', 'COS(T)', etc Dans ce dernier cas, et avec le polynôme ci-dessus, on obtient l'expression algébrique:

'7*COS(T)^5-4*COS(T)^3+COS(T)^2+2*COS(T)'

'V→P' est utile pour visualiser plus aisément un polynôme écrit sous forme de vecteur (et obtenu par exemple comme résultat d'un des programmes du répertoire) ou pour utiliser les programmes SOLVR ou DRAW du calculateur.

'V→P' fonctionne aussi si l'argument au niveau 2 est une liste. On peut ainsi l'utiliser conjointement avec le programme 'T→Q' du répertoire 'R.C'.

POLYNOMES DE TCHEBICHEV.

=====

'TCHEB' calcule les polynômes de Tchebichev de première espèce T_n , et de seconde espèce U_n .

Les polynômes T_n sont définis par:

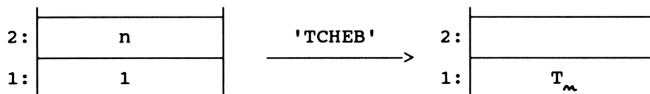
$T_0(x)=1$, $T_1(x)=x$, et pour tout $n \geq 2$, $T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x)$.

Les polynômes U_n sont définis par:

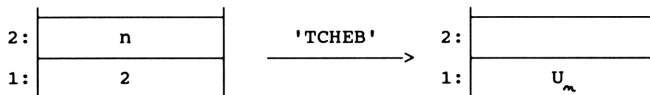
$U_0(x)=1$, $U_1(x)=2x$, et pour tout $n \geq 2$, $U_n(x) = 2xU_{n-1}(x) - U_{n-2}(x)$.

Le schéma fonctionnel est le suivant:

* pour obtenir les polynômes de Tchebichev de première espèce:



* pour obtenir les polynômes de Tchebichev de seconde espèce:



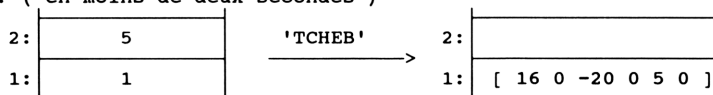
'TCHEB': (Checksum: # 24506d, Taille: 231.5 octets)

```

«   IF { 1 2 } OVER POS THEN 0 2 →ARRY 1 DUP →ARRY
    → n b a
    «   IF n THEN
          IF n 1 == THEN b ELSE
            3 n 1 + FOR k
              b DUP k 1 →LIST RDM 2 * 0 0
              a V→ k →ARRY - 'b' STO 'a' STO
            NEXT b
          END
        ELSE a END
    »
  »
END

```

Exemple: (en moins de deux secondes)

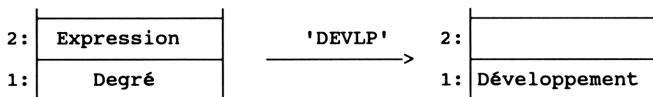


Car $T(X) = 16X^5 - 20X^3 + 5X$.

DEVELOPPEMENT D'EXPRESSIONS POLYNOMIALES

=====

'DEVLP' permet de développer une expression polynômiale de la variable X, suivant le schéma fonctionnel suivant:



'Expression' doit être écrite par rapport à la variable 'X'.

'Degré' est le degré du polynôme représenté par 'Expression'.

'Développement' est la forme développée de 'Expression', écrite suivant les puissances décroissantes de 'X'.

Attention: Les coefficients obtenus sont exacts dans la mesure où le degré est "raisonnable" (inférieur à 11). Il faut faire en sorte que, si n est le degré du polynôme P, le calcul de P(n) ne crée pas d'erreurs d'arrondi trop importantes.

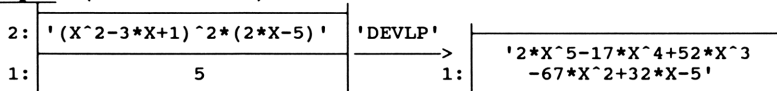
'DEVLP': (Checksum: # 20417d, Taille: 214.5 octets)

```

«   → p  n
    «   0  n  FOR i
        i  'X'  STO p  EVAL
    NEXT
    n  1  +  →ARRY
    0  n  FOR i
        0  n  FOR j
            i  j  ^
        NEXT
    NEXT
    n  1  +  DUP  2  →LIST
    →ARRY  /  V→  0
    n  0  FOR i
        SWAP 'X'  i  ^  *  +
    -1  STEP
    'X'  PURGE
    »
»

```

Exemple: (En 5 secondes)



DEVELOPPEMENT D'UN PRODUIT DE PUISSANCES DE POLYNOMES

=====

'DEVPP' permet de développer un produit de puissances de polynômes c'est-à-dire un polynôme s'écrivant:

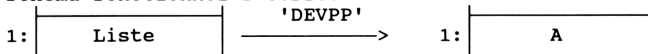
$$A = B_1^{\alpha_1} B_2^{\alpha_2} \dots B_n^{\alpha_n}$$

La forme factorisée doit être entrée sous la forme d'une liste constituée de la manière suivante:

$$\text{Liste} = \{ B_1 \alpha_1 B_2 \alpha_2 \dots B_n \alpha_n \}$$

Tous les exposants doivent être présents, même égaux à 1.

Le schéma fonctionnel s'écrit:



où A est le polynôme (écrit sous forme de vecteur) résultant du développement.

N.B: 'DEVPP' appelle les programmes 'EXPO' et 'PRODP'.

'DEVPP': (Checksum: # 31499d, Taille: 87 octets)

```

« 1  DUP  →ARRY
  1  3  PICK  SIZE  FOR  i
    OVER  i  GETI  3  ROLLD  GET  EXPO  PRODP
  2  STEP
    SWAP  DROP
»

```

Exemple: (en cinq secondes)

```

1: [ { [ 1 5 ] 2 [ 1 -2 4 ] 3 } ]

'DEVPP'
-----> 1: [ 1 4 -11 34 136 -536 1504 -1760 1600 ]

```

Autrement dit: $(X+5)^2(X^2-2X+4)^3$ est égal à $X^8 + 4X^7 - 11X^6 + 34X^5 + 136X^4 - 536X^3 + 1504X^2 - 1760X + 1600$

Remarque:

Par rapport à 'DEVLP' (développement d'expressions polynômiales sous forme algébrique), 'DEVPP' a l'avantage de ne pas présenter de limitations en terme de degré.

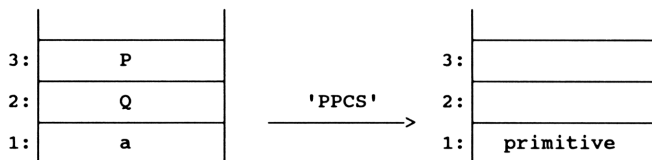
Mais 'DEVLP' permet de développer des expressions polynômiales contenant des sommes de polynômes.

PRIMITIVATION DE **$P(x)\cos(ax) + Q(x)\sin(ax)$**

=====

'PPCS' permet de calculer une primitive, sous forme symbolique, d'une fonction s'écrivant $P(x)\cos(ax) + Q(x)\sin(ax)$, où P et Q sont deux polynômes et où a est un réel non nul.

Le schéma fonctionnel est:



Ici les polynômes P et Q sont écrits avec le format habituel (vecteurs des composantes suivant les puissances décroissantes), et 'primitive' est une expression algébrique représentant une primitive symbolique de l'application $P(x)\cos(ax) + Q(x)\sin(ax)$. Cette primitive est obtenue sous la forme $A(x)\cos(ax) + B(x)\sin(ax)$, où A et B sont deux expressions polynômiales par rapport à la variable 'X'.

N.B: 'PPCS' appelle les programmes 'DERIV', 'ADDP' et 'V→P'.

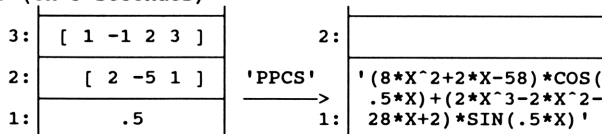
'PPCS': (Checksum: # 26977d, Taille: 311 octets)

```

" → P  Q  a
  " P  DERIV  Q  a  *  NEG  ADDP  DUP  SIZE  1  GET
    → r  n
    " 0  0  n  1  FOR  k
      r  n  k  -  1  +  GET  3  PICK
      k  1  +  *  k  *  -  a  SQ  /
    -1 STEP
    n →ARRY  3  ROLL  DROP2  DUP  'X'  V→P  a
    'X' * COS * P ROT DERIV NEG ADDP a
    / 'X' V→P a 'X' * SIN * +
  "
"
"

```

Exemple: (en 5 secondes)



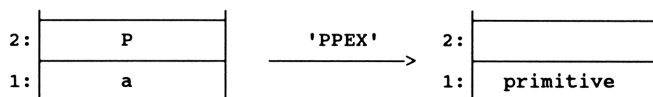
En effet: $\int [(X^3 - X^2 + 2X + 3)\cos(X/2) + (2X^2 - 5X + 1)\sin(X/2)] dx$
 est égal à $(8X^2 + 2X - 58)\cos(X/2) + (2X^3 - 2X^2 - 28X + 2)\sin(X/2)$
 (à la constante d'intégration près).

PRIMITIVATION DE $P(x)\exp(ax)$

=====

'PPEX' permet de calculer une primitive, sous forme symbolique, d'une fonction s'écrivant $P(x)\exp(ax)$, où P est un polynôme et où a est un réel non nul.

Le schéma fonctionnel est:



Ici le polynôme P est écrit avec le format habituel (vecteur des composantes suivant les puissances décroissantes), et 'primitive' est une expression algébrique représentant une primitive symbolique de l'application $P(x)\exp(ax)$. Cette primitive est obtenue sous la forme $A(x)\exp(ax)$, où A est une expression polynomiale par rapport à la variable 'X'.

N.B: 'PPEX' appelle le programme 'V→P'.

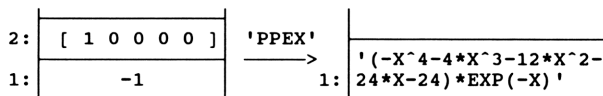
'PPEX': (Checksum: # 44523d, Taille: 163 octets)

```

«   OVER   SIZE   1   GET
  →  p     a     n
  «   0     n     1   FOR   k
      p     n     k   -   1   +   GET   OVER   k   *   -   a   /
      -1   STEP
      n   →ARRY   SWAP   DROP   'X'   V→P   a   'X'   *   EXP   *
  »
»

```

Exemple: En deux à trois secondes:



En effet: $\int X^4 \exp(-X) dx = (-X^4 - 4X^3 - 12X^2 - 24X - 24) \exp(-X)$
(à la constante d'intégration près).

SOLUTION DE L'EQUATION

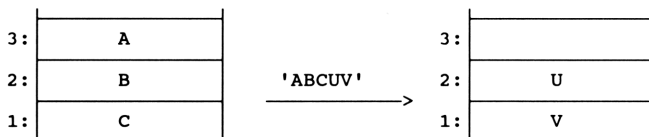
$$AU + BV = C.$$

=====

Le programme 'ABCUV' permet d'obtenir la meilleure solution (U,V) (c'est-à-dire la solution minimisant les degrés de U et V) pour l'équation : $AU+BV=C$, où A,B,C sont trois polynômes donnés et où U,V sont deux polynômes cherchés.

Pour qu'une telle équation admette au moins une solution (et elle en admet alors une infinité) il est nécessaire et suffisant que le polynôme C soit un multiple du PGCD des polynômes A et B.

Le schéma fonctionnel est le suivant:



où les polynômes A et B sont écrits sous la forme habituelle (vecteur des composantes suivant les puissances décroissantes).

Dans le cas où l'équation $AU+BV=C$ n'a pas de solution, le programme s'achève avec le message "Pas de solution".

N.B: Le programme 'ABCUV' appelle les programmes 'DIV', 'PRODP', 'ADDP' et 'ELMG'

'ABCUV': (Checksum: # 6635d, Taille: 362 octets)

```

« 0 1 →ARRY 1 DUP →ARRY DUP2
→ a b c u v y x
« WHILE a b DIV DUP ABS
  REPEAT
    b 'a' STO 'b' STO
    u DUP 3 PICK PRODP NEG
    x ADDP 'u' STO 'x' STO
    v DUP ROT PRODP NEG
    y ADDP 'v' STO 'y' STO
  END
  DROP2 c b DIV
  IF ABS THEN
    DROP "Pas de solution"
  ELSE
    u OVER PRODP ELMG
    v ROT PRODP ELMG
  END
»
»

```

EXEMPLES D'UTILISATION DU PROGRAMME 'ABCUV'

=====

Exemple 1: (temps de calcul = 8 secondes, en mode 5 FIX)

3:	[1 -1 -2 3]		3:	
2:	[1 3 4]	'ABCUV'		[-0.03681 0.00613]
1:	[1]	→	1:	[0.03681 -0.15337 .24540]

Une utilisation de 'T→Q' dans le répertoire 'R.C' montre que les coefficients obtenus sont :

* { '-6/163' '1/163' } au niveau 2.
* { '6/163' '-25/163' '40/163' } au niveau 1.

Autrement dit on vient d'obtenir le résultat: $AU+BV=C$, avec:

$$\begin{aligned} A &= X^3 - X^2 - 2X + 3, & B &= X^2 + 3X + 4, & C &= 1, \\ U &= (-6X^2 + 1)/163, & V &= (6X^3 - 25X^2 + 40)/163 \end{aligned}$$

Exemple 2: (en 5 secondes)

3:	[1 -1 -1 -2]		3:	
2:	[1 1 -6]	'ABCUV'		
1:	[1 1]	→	1:	"Pas de Solution"

Ce résultat signifie que l'équation $AU+BV=C$, avec:

$$\begin{aligned} A &= X^3 - X^2 - X - 2, & B &= X^2 + X - 6, & C &= X + 1, \\ & & & & & \text{n'a pas de solution.} \end{aligned}$$

La raison en est que les deux polynômes A et B ont pour PGCD le polynôme $X-2$, et que C n'est pas divisible par $X-2$.

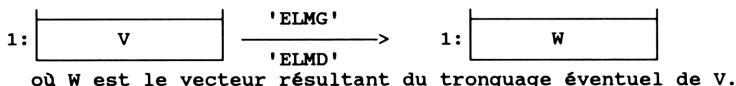
ELIMINATION DES COEFFICIENTS NULS A DROITE OU A GAUCHE.

=====

'ELMG' et 'ELMD' sont deux routines qui éliminent les coefficients nuls qui pourraient débiter (dans le cas de 'ELMG') ou terminer (dans le cas de 'ELMD') un vecteur V.

Pour combattre les erreurs d'arrondis, est considéré comme nul tout coefficient dont la valeur absolue est inférieure à 1E-5 (cette valeur peut être modifiée dans le texte des programmes 'ELMG' et 'ELMD').

Le schéma fonctionnel de 'ELMG' et de 'ELMD' est le suivant:



'ELMG': (Checksum: # 8202d, Taille: 134.5 octets)

```

«   DUP   RNRM
    IF .00001 < THEN DROP 0 1 →ARRY ELSE
      OBJ→ 1 GET 1 +
      WHILE DUP ROLL DUP ABS .00001 <
        REPEAT DROP 1 - END
      OVER ROLL 1 - →ARRY
    END
  »
  
```

'ELMD': (Checksum: # 22420d, Taille: 117 octets)

```

«   DUP   RNRM
    IF .00001 < THEN DROP 0 1 →ARRY
      ELSE OBJ→ 1 GET
      WHILE OVER ABS .00001 <
        REPEAT 1 - SWAP DROP END
      →ARRY
    END
  »
  
```

Exemple:

le vecteur V= [0 1E-11 1 -2 3 4 0 1E-7 0] est transformé
 en W= [1 -2 3 4 0 1E-7 0] par 'ELMG'
 et en W= [0 1E-11 1 -2 3 4] par 'ELMD'

FRACTIONS RATIONNELLES

Le répertoire 'FRAC' doit être placé dans le répertoire 'POLY'.

Il contient un certain nombre de programmes s'appliquant aux fractions rationnelles, c'est-à-dire aux fonctions s'écrivant comme le quotient $R=A/B$ de deux fonctions polynômes A et B.

'SMFF' : Simplification d'une fraction rationnelle.

'ADDF' : Addition de deux fractions rationnelles.

'PRODF' : Produit de deux fractions rationnelles.

'F+Q' : Conversion en nombres rationnels des coefficients d'une fraction rationnelle.

'V+F' : Ecriture algébrique d'une fraction rationnelle.

'VALF' : valeur d'une fraction rationnelle en un point.

'EXPOF' : Puissances d'une fraction rationnelle.

'DERVF' : Dérivée d'une fraction rationnelle.

Et surtout:

'DECPF' : Décomposition d'une fraction rationnelle en éléments simples.

Dans la plupart des cas ci-dessus, une fraction rationnelle est représentée sur la pile par la superposition de deux vecteurs représentant l'un le numérateur, l'autre le dénominateur.

Par exemple, la fraction rationnelle: $R = \frac{X^3 - 2X^2 + 5X - 1}{11X^2 - 15X + 21}$

est représentée sur la pile par:

2:	[1 -2 5 -1]
1:	[11 -15 21]

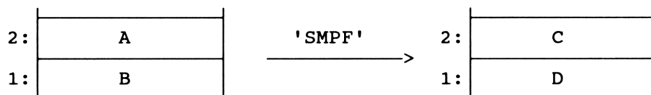
SIMPLIFICATION D'UNE FRACTION RATIONNELLE.

=====

Le programme 'SMPF' permet de simplifier (si possible), la fraction rationnelle $R=A/B$. On obtient la fraction rationnelle C/D .

Si aucune simplification n'est possible, la fraction A/B reste inchangée.

Le schéma fonctionnel est:



N.B: 'SMPF' appelle les programmes 'PGDP' et 'DIV' du répertoire 'POLY'.

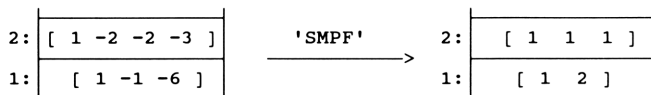
'SMPF':

```

«   DUP2   PGDP
   IF DUP   [ 1 ] ≠ THEN
     ROT   OVER   DIV   DROP   3   ROLLD   DIV
   END
   DROP
»

```

Exemple: (en six secondes)



En effet:

$$\frac{X^3 - 2X^2 - 2X - 3}{X^2 - X - 6} \text{ est égale à } \frac{X^2 + X + 1}{X + 2} .$$

ADDITION ET PRODUIT DE FRACTIONS RATIONNELLES.

=====

Les programmes 'ADDF' et 'PRODF' permettent respectivement d'additionner et de multiplier deux fractions rationnelles A/B et C/D.

Si on note E/F la fraction rationnelle représentant le résultat, le schéma fonctionnel est:



N.B: 'PRODF' appelle 'PRODP' dans le répertoire 'POLY'.
'ADDF' appelle 'PRODP', 'ELMG', et 'ADDP'.

Remarque: après addition ou multiplication, aucune tentative de simplification n'est réalisée.

'ADDF':

```

"  →  a  b  c  d
    "  a  d  PRODP  b  c  PRODP  ADDP  ELMG
      b  d  PRODP
    "
  "

```

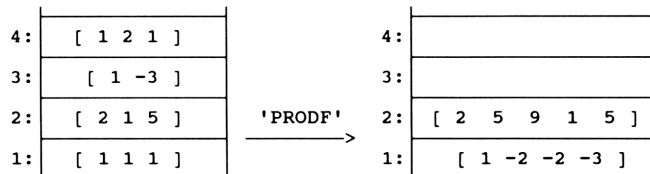
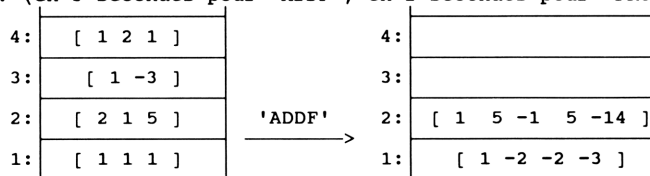
'PRODF':

```

"  ROT  PRODP  3  ROLL  PRODP  SWAP  "

```

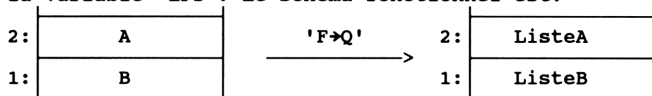
Exemple: (en 3 secondes pour 'ADDF', en 2 secondes pour 'PRODF').



CONVERSION DES COEFFICIENTS D'UNE FRACTION RATIONNELLE EN NOMBRES RATIONNELS.

=====

'F→Q' transforme en leur approximation rationnelle les coefficients d'une fraction rationnelle A/B. Cette transformation se fait par appel du programme 'T→Q' dans le répertoire 'R.C' et elle y utilise la variable 'EPS'. Le schéma fonctionnel est:

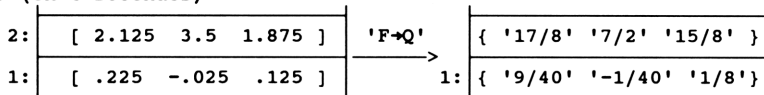


où ListeA et ListeB sont les listes contenant les approximations rationnelles des coefficients des polynômes A et B.

N.B: 'F→Q' passe dans le répertoire 'R.C', y appelle le programme 'T→Q' avant de revenir dans le répertoire 'FRAC', via le répertoire 'POLY'.

'F→Q': « R.C SWAP T→Q SWAP T→Q POLY FRAC »

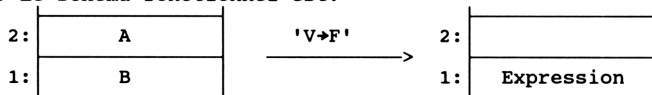
Exemple: (en 4 secondes)



ECRITURE ALGEBRIQUE D'UNE FRACTION RATIONNELLE.

=====

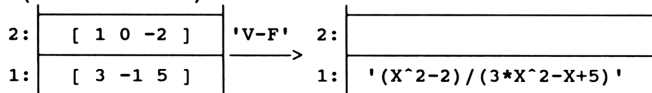
'V→F' permet d'obtenir l'expression, par rapport à la variable X, d'une fraction rationnelle A/B, A et B étant donnés sous la forme vecteur. Le schéma fonctionnel est:



N.B: 'V→F' appelle 'V→P' dans le répertoire 'POLY'.

'V→F': « SWAP 'X' V→P SWAP 'X' V→P / »

Exemple: (en 2 secondes)



VALEUR EN UN POINT D'UNE FRACTION RATIONNELLE.

=====

'VALF' calcule la valeur au point x de la fraction rationnelle $R=A/B$, suivant le schéma fonctionnel:



N.B: 'VALF' appelle le programme 'VALP' du répertoire 'POLY'

'VALF': " ROT OVER VALP 3 ROLLD VALP / "

Exemple: (en une seconde)

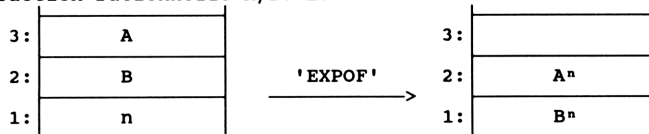


Le résultat obtenu est en fait égal à 25/13.

PUISSANCES ENTIERES D'UNE FRACTION RATIONNELLE.

=====

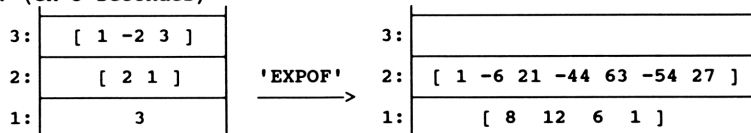
'EXPOF' calcule la puissance n-ème (où n est un entier positif) d'une fraction rationnelle A/B. Le schéma fonctionnel est:



N.B: 'EXPOF' appelle 'EXPO' dans le répertoire 'POLY'.

'EXPOF': " ROT OVER EXPO 3 ROLLD EXPO "

Exemple: (en 5 secondes)



DERIVATION D'UNE FRACTION RATIONNELLE.

=====

'DERVF' calcule la dérivée C/D d'une fraction rationnelle A/B, avec donc:

$$C = A'B - AB' \quad \text{et} \quad D = B^2.$$

Le schéma fonctionnel est le suivant:



N.B: Bien sûr, le programme 'DERVF' appelle le programme 'DERIV' du répertoire 'POLY'. On y appelle également les programmes 'PRODP', 'ADDP' et 'ELMG'.

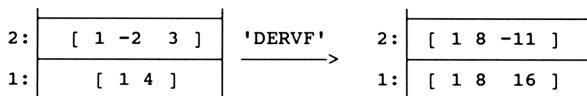
'DERVF':

```

«   →   a   b
«   «   a   DERIV   b   PRODP
      a   b   DERIV   PRODP   NEG
      ADDP   ELMG
      b   DUP   PRODP
«
»

```

Exemple: (en 3 secondes)



Effectivement, la dérivée de:

$$R(X) = \frac{X^2 - 2X + 3}{X + 4} \quad \text{est} \quad R'(X) = \frac{X^2 + 8X - 11}{X^2 + 8X + 16}$$

DECOMPOSITION D'UNE FRACTION RATIONNELLE EN ELEMENTS SIMPLES

=====

'DECPF' permet de procéder à la décomposition en éléments simples d'une fraction rationnelle $R=A/B$, à coefficients réels ou complexes.

Le numérateur A doit être donné sous forme d'un vecteur.

Le dénominateur B doit être donné sous forme d'une liste. Plus précisément, si la décomposition de B en produits de facteurs irréductibles s'écrit:

$$B = B_1^{\alpha_1} B_2^{\alpha_2} \dots B_m^{\alpha_m} \quad (\text{où les } \alpha_i \text{ sont des entiers } \geq 1),$$

alors B doit être donné sous la forme:

$$B = \{ B_1 \alpha_1 B_2 \alpha_2 \dots B_m \alpha_m \}.$$

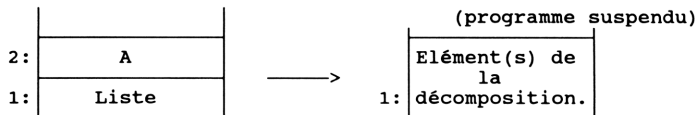
Dans cette liste, tous les exposants α_i doivent être présents, même égaux à 1.

Les différents polynômes B_i doivent être premiers entre eux deux à deux, et irréductibles (donc de degré 1, ou de de degré 2 mais à coefficients réels et avec un discriminant négatif)

Le programme 'DECPF' n'effectue aucune vérification de la validité de la liste donnant B.

Une non observation des conditions décrites ci-dessus conduit inévitablement à des erreurs d'exécution ou à des résultats ne pouvant pas être interprétés valablement.

Le schéma fonctionnel est le suivant:



Le programme fournit peu à peu tous les éléments de la décomposition en éléments simples.

A chaque présentation d'un résultat intermédiaire, le programme est suspendu.

Il peut être repris par CONT (on n'est pas tenu de laisser la pile en l'état) pour obtenir le résultat intermédiaire suivant.

Supposons donc que la fraction rationnelle R s'écrive:

$$R = \frac{A}{B_1^{a_1} B_2^{a_2} \dots B_m^{a_m}}. \text{ Les résultats sont donnés dans l'ordre suivant:}$$

Premier résultat: La partie entière E (même si elle est nulle), sous forme d'un vecteur (notation traditionnelle des polynômes dans le répertoire 'POLY').

Deuxième résultat: La partie principale relative à la présence du facteur $B_k^{a_k}$ au dénominateur de la fraction rationnelle R: On sait qu'une telle partie principale s'écrit:

$$\frac{A_k}{B_k^{a_k}} + \frac{A_{k-1}}{B_k^{a_k-1}} + \frac{A_{k-2}}{B_k^{a_k-2}} + \dots + \frac{A_1}{B_k}$$

où $A_k, A_{k-1}, \dots, A_1$ sont des polynômes de degré strictement inférieur au degré de B_k .

Le résultat est ici obtenu sous forme d'une liste constituée de la manière suivante:

Liste = { $A_k, A_{k-1}, \dots, A_1$ }, les différents polynômes A_i étant écrits sous forme de vecteur.

Résultats suivants: On obtient successivement les parties principales relatives à la présence des autres facteurs B_k au dénominateur de la fraction R, comme expliqué ci-dessus.

Remarques:

'DECPF' appelle 'DIV' et 'ABCUV' dans le répertoire 'POLY'.

'DECPF' a été optimisé pour être le plus court et surtout le plus précis possible. Il est possible de l'améliorer en tenant compte de certaines particularités (notamment si le dénominateur contient au plus deux facteurs différents).

Une particularité de 'DECPF' est que toutes les parties principales sont calculées avec la même précision. En fait on n'utilise pas les résultats déjà obtenus pour progresser dans la décomposition. Ce serait possible (l'idée étant d'accélérer les calculs) mais la propagation des erreurs d'arrondis pourrait vite devenir catastrophique.

'DECPF':

```

"  DUP  SIZE  →  a  L1  n
"  { } [ 1 ]
  1  n  FOR  i
    L1  i  GETI  3  ROLL  GET  EXPO  ROT
    OVER  +  0  +  3  ROLL  PRODP
  2  STEP
  →  L2  b
"  a  b  DIV  'a'  STO  HALT
  1  n  FOR  i
    L2  i  GET  b  OVER  DIV  DROP  SWAP  a
    ABCUV  DROP  L1  i  GETI  3  ROLL  GET
  →  d  k
    "  1  k  START  d  DIV  SWAP  NEXT
    DROP  k  →LIST
  "
  HALT  2  STEP
"  "  "

```

PROGRAMME 'DECPF': EXEMPLE D'UTILISATION.

=====

Soit à décomposer en éléments simples la fraction rationnelle:

$$R = \frac{X^{13}}{(X-1)^2 * (X^2+X+1) * (X^2+1)^3}$$

On crée la pile :

2:	[1 0 0 0 0 0 0 0 0 0 0 0 0]
1:	{ [1 -1] 2 [1 1 1] 1 [1 0 1] 3 }

et on appelle le programme 'DECPF':

Au bout de dix secondes, le programme est suspendu et on trouve le vecteur [1 1 -2 -1] au niveau 1 de la pile. Par conséquent, la partie entière est égale à X^3+X^2-2X-1 .

On poursuit l'exécution du programme (CONT): après 43 secondes, on obtient, au niveau 1 de la pile:

1:	{ [.041666666661] [.374999999959] }
----	---

(une utilisation du programme R→Q dans le répertoire R.C montre que les nombres obtenus valent 1/24 et 3/8)

Ainsi, la partie principale correspondant à $(X-1)^2$ est:

$$\frac{1}{24*(X-1)^2} + \frac{3}{8*(X-1)}$$

On poursuit par CONT: on trouve en 25 secondes:

1:	{ [.333333333344] }	Le terme correspondant dans la décomposition est donc :
----	-----------------------	---

$$\frac{1}{3*(X^2+X+1)}$$

On poursuit par CONT: on trouve en 49 secondes:

1:	{ [.500000000001 .000000000004] [-2.500000000003 -.250000000016] [4.625000000008 1.250000000036] }
----	--

Par conséquent, on vient d'obtenir la partie suivante de la décomposition:

$$\frac{X}{2*(X^2+1)^3} + \frac{-10*X-1}{4*(X^2+1)^2} + \frac{37*X+10}{8*(X^2+1)}$$

On refait CONT et le programme est terminé.

On a ainsi obtenu la totalité de la décomposition en éléments simples de la fraction rationnelle de départ. On remarque la très bonne qualité (erreurs d'arrondi faibles) des résultats obtenus.

CALCUL MATRICIEL

Le répertoire 'MATR' contient les programmes consacrés aux calculs portant sur les matrices ou les systèmes linéaires.

Dans ce domaine, la HP48S donne toute sa mesure: il est possible de résoudre des problèmes lourds au moyen de programmes relativement simples. Les calculs fastidieux consistant en les produits de matrices ou de vecteurs sont ici ignorés du programmeur puisque la machine gère elle-même tous ces produits.

Néanmoins, on rencontrera ici des programmes dont le temps de calcul n'est pas négligeable: les calculs sur les tableaux sont en effet relativement longs, au moins pour les raisons suivantes:

- Obligation d'utiliser un adressage indexé pour accéder à tel ou tel coefficient d'une matrice donnée.
- Recopie fréquente de tableaux sur la pile.

Le répertoire 'MATR' contient les programmes suivants:

- * 'CB' effectue le changement de matrice d'une application linéaire (connaissant la matrice initiale et la matrice de changement de base).
- * 'TR' calcule la trace d'une matrice carrée.
- * 'PUISS' permet de calculer les puissances quelconques d'une matrice carrée.
- * 'INVN' donne l'inverse d'une matrice carrée A à coefficients entiers, en donnant son déterminant d et la matrice (à coefficients entiers) $B = d * A^{-1}$.
- * 'ALU' permet de décomposer une matrice carrée en le produit d'une matrice triangulaire inférieure à diagonale unité et d'une matrice triangulaire supérieure.
- * 'PUTL' place une ligne dans une matrice.
- * 'PUTC' place une colonne dans une matrice.
- * 'GETL' extrait une ligne quelconque d'une matrice.
- * 'GETC' extrait une colonne quelconque d'une matrice.
- * 'ECHL' échange deux lignes.
- * 'EHC' échange deux colonnes.
- * 'CALCL' permet d'effectuer toutes sortes de calculs sur les lignes d'une matrice.
- * 'CALCC' permet d'effectuer toutes sortes de calculs sur les colonnes d'une matrice.
- * 'CRTAB' permet de créer un tableau (matrice ou vecteur) dont le terme général répond à une formule donnée.

.../...

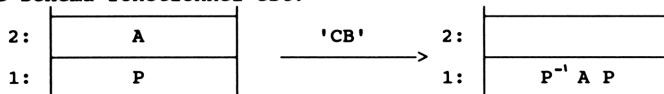
- * 'PCAR' donne le polynôme caractéristique d'une matrice carrée.
- * 'VP234' donne les valeurs propres d'une matrice carrée d'ordre 2,3, ou 4.
- * 'DEFL' permet d'approcher les valeurs propres et vecteurs propres dans le cas de matrices de dimension supérieure à 4.
- * 'RANG' calcule le rang d'une matrice quelconque (par utilisation de la méthode du pivot de Gauss).
- * 'SYST' permet de donner l'expression symbolique de la solution générale d'un système de n équations à p inconnues. Le cas où le système possède une infinité de solutions est ici particulièrement intéressant.
- * 'VECTP' donne la ou les équations du sous-espace propre d'une matrice carrée pour une valeur propre donnée.
- * 'DIVVR' permet de diviser un tableau, par une matrice carrée, avec une précision améliorée par rapport à '/ '.
- * 'INVVR' permet d'inverser une matrice carrée , avec une précision supérieure à celle de la commande INV.
- * 'ADDID' consiste à border à droite une matrice A (à n lignes) par la matrice identité d'ordre n (utile dans les programmes 'POLM', 'PIVOT' et 'EQRL'.
- * 'POLM' calcule le polynôme minimal d'une matrice carrée.
- * 'PIVOT' décrit la méthode du pivot de Gauss appliquée à une matrice.
- * 'EQRL' permet de connaître les éventuelles relations linéaires existant entre n vecteurs, ou les équations de l'espace vectoriel engendré par ces n vecteurs.

CHANGEMENT DE BASE AU MOYEN D'UNE MATRICE DE PASSAGE.

=====

'CB' calcule la nouvelle matrice $B = P^{-1} A P$ d'une application linéaire f , l'ancienne matrice étant A , et la matrice de passage de l'ancienne base à la nouvelle base étant P .

Le schéma fonctionnel est:



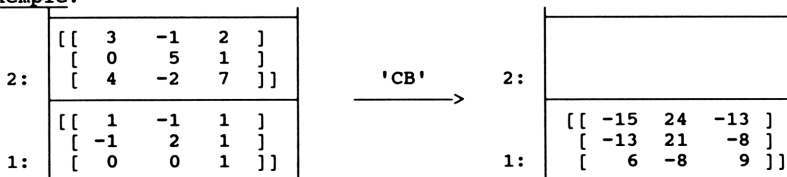
'CB': (Checksum: 31267# d, Taille: 42.5 octets)

```

"      → P
"      " P / P *
"

```

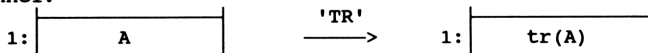
Exemple:



CALCUL DE LA TRACE D'UNE MATRICE CARREE

=====

'TR' calcule la trace $\text{tr}(A)$ (c'est à dire la somme des coefficients diagonaux) d'une matrice carrée A , suivant le schéma fonctionnel:



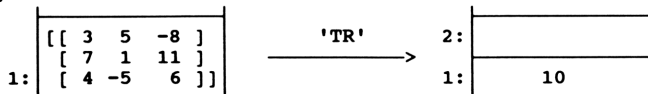
'TR': (Checksum: # 16159d, Taille: 68 octets)

```

"
0
1 3 PICK SIZE 2 GET FOR i
OVER i DUP 2 -LIST GET +
NEXT
SWAP DROP
"

```

Exemple:



PUISSANCES D'UNE MATRICE CARREE.

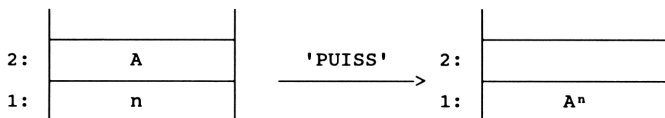
=====

'PUISS' Calcule les puissances A^n (l'exposant n étant un entier positif ou négatif) d'une matrice carrée A.

Si l'exposant est négatif, la matrice A doit être inversible.

Si n est nul, le résultat est la matrice identité de même dimension que A.

Le schéma fonctionnel est:



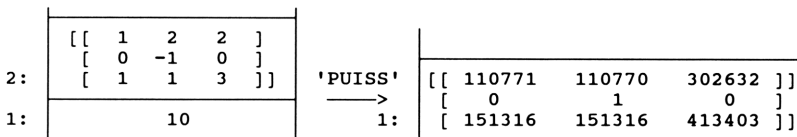
'PUISS': (Checksum: # 53967d, Taille: 158 octets)

```

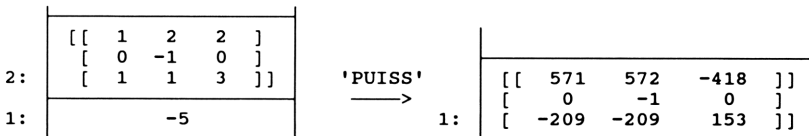
«   IF   DUP   0   <   THEN
      SWAP   INV   SWAP   NEG
    END
    OVER   IDN
    →   n   p
    «   WHILE   n
      REPEAT
        IF   n   2   MOD   THEN
          DUP   'p'   STO*
        END
        SQ   n   2   /   FLOOR   'n'   STO
      END
      DROP   p
    »
  »

```

Exemple 1: (en moins de 2 secondes)



Exemple 2: (en 2 secondes)

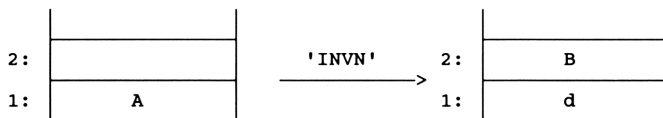


INVERSION D'UNE MATRICE A COEFFICIENTS ENTIERS.

=====

'INVN' permet de calculer de façon précise l'inverse d'une matrice carrée A à coefficients entiers.

Le schéma fonctionnel est le suivant:



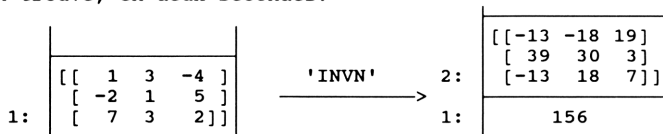
où "d" est le déterminant de A (c'est un entier), et où B est la matrice carrée à coefficients entiers telle que $A^{-1} = (1/d) * B$.

'INVN': (Checksum: # 24340d, Taille: 56.5 octets)

```
«  DUP   DET   .5  +   FLOOR  SWAP
   INV   OVER  *   0   RND   SWAP
»
```

Exemple: Soit à calculer l'inverse de la matrice $A = \begin{bmatrix} 1 & 3 & -4 \\ -2 & 1 & 5 \\ 7 & 3 & 2 \end{bmatrix}$

On trouve, en deux secondes:



L'inverse de A est donc la matrice $\frac{1}{156} \begin{bmatrix} -13 & -18 & 19 \\ 39 & 30 & 3 \\ -13 & 18 & 7 \end{bmatrix}$

Répertoire 'MATR'.

Programme 'ALU'.

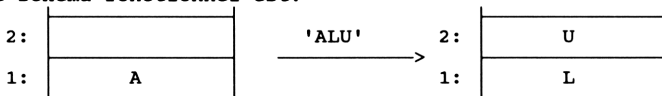
DECOMPOSITION "A=LU" D'UNE MATRICE CARREE A.

=====

'ALU' décompose une matrice carrée A en le produit LU d'une matrice carrée L (triangulaire inférieure à diagonale unité) et d'une matrice U (triangulaire supérieure). (L pour "Low" , U pour "Up").

N.B: Le programme 'ALU' écrase la variable 'L' qui se trouve éventuellement dans le répertoire. Il appelle les programmes 'GETL' et PUTL'.

Le schéma fonctionnel est:



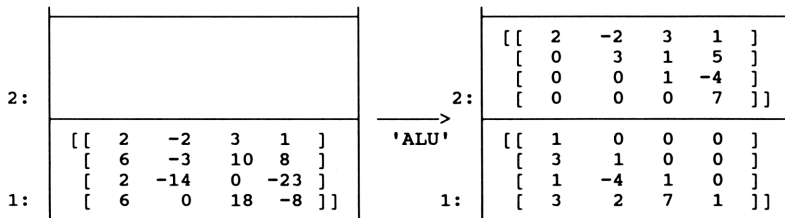
'ALU': (Checksum: # 43868d, Taille: 249.5 octets)

```

«  DUP  SIZE  1  GET  DUP  IDN  'L'  STO
  → d
  «  1  d  1  -  FOR  i
    DUP  i  GETL  DUP  i  GET
    → Lp  p
    «  i  1  +  d  FOR  j
      'L'  j  i  2  →LIST  3  PICK  OVER
      GET  p  /  DUP  4  ROLLD  PUT  OVER
      j  GETL  Lp  ROT  *  -  j  PUTL
    NEXT
  »
  NEXT
  L
  'L'  PURGE
»
  »

```

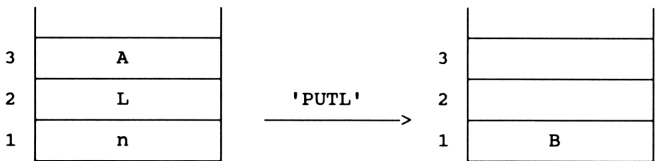
Exemple: (en 8 secondes)



PLACER UNE LIGNE DONNEE
DANS UNE MATRICE DONNEE.

=====

'PUTL' place la ligne L dans la matrice A, au niveau de la ligne de numéro n, transformant ainsi A en une matrice B, suivant le schéma fonctionnel suivant:

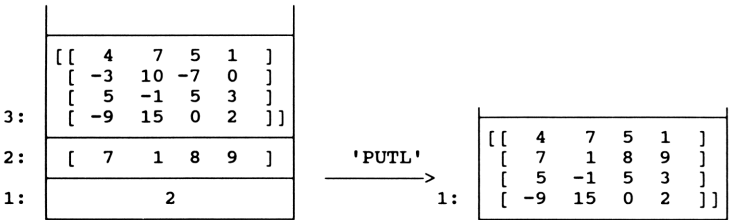


N.B: L peut aussi bien être un vecteur qu'une matrice uni-ligne.

'PUTL': (Checksum: # 8768d, Taille: 84 octets)

```
“ { 1 } + SWAP
  → L
  “ 1 3 PICK SIZE 2 GET FOR i
    L i GET PUTI
    NEXT
    DROP
  ”
”
```

Exemple:



PLACER UNE COLONNE DONNEE DANS UNE MATRICE DONNEE.

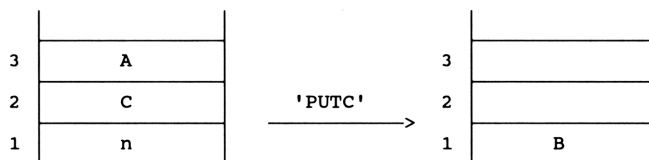
=====

'PUTC' Place la colonne C dans la matrice A, au niveau de la colonne de numéro n, transformant ainsi A en une matrice B.

N.B: 'PUTC' appelle le programme 'PUTL'.

La colonne C peut être écrite comme vecteur ou comme matrice unicolonne.

schéma fonctionnel:

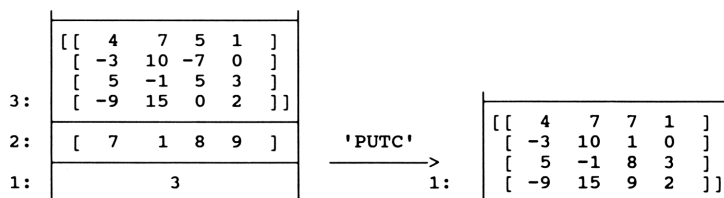


'PUTC': (Checksum: # 48548d, Taille: 53.5 octets)

```

«   ROT   TRN   ROT
   DUP  SIZE   1   1   SUB   RDM
   ROT   PUTL   TRN
»
  
```

Exemple:



EXTRAIRE UNE LIGNE DONNEE
D'UNE MATRICE DONNEE.
=====

'GETL' extrait la ligne de numéro n de la matrice A. Le résultat est un vecteur L.

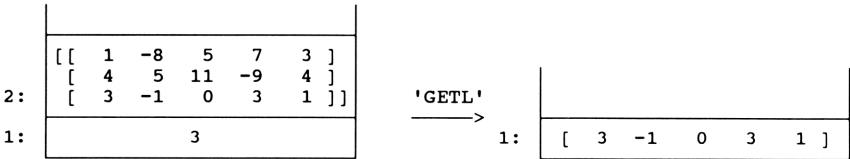
schéma fonctionnel:



'GETL': (Checksum: # 60021d, Taille: 51 octets)

« SWAP TRN DUP SIZE 2 2 SUB
 0 CON ROT 1 PUT *
»

Exemple:



EXTRAIRE UNE COLONNE DONNEE D'UNE MATRICE DONNEE.

=====

'GETC' extrait la colonne de numéro n de la matrice A. Le résultat est une matrice unicolonne C.

schéma fonctionnel:

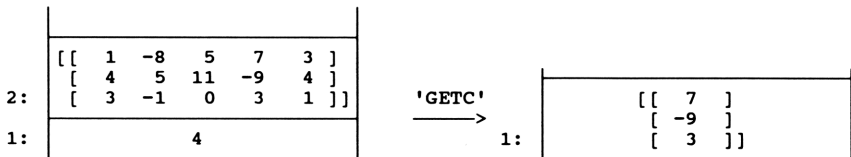


'GETC': (Checksum: # 41771d, Taille: 48.5 octets)

```

«   OVER   SIZE   1   1   PUT   0   COM
   TRN   SWAP   1   PUT   *
»
  
```

Exemple:

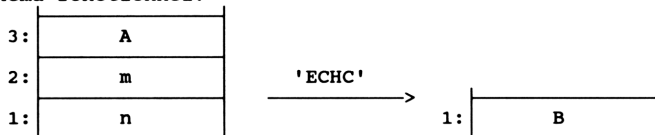


ECHANGE DE DEUX COLONNES D'UNE MATRICE.

=====

'EHC' échange les colonnes de numéros n et m de la matrice A, transformant ainsi A en une matrice B.

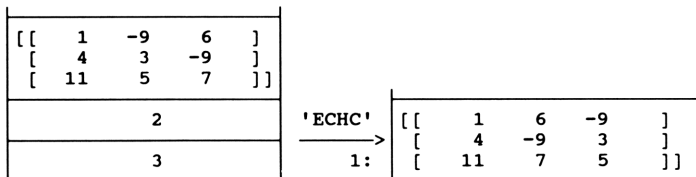
schéma fonctionnel:



'EHC': (Checksum: # 47058d, Taille: 126 octets)

```
«   →   m   n
«   DUP SIZE 2 GET IDN m m 2 →LIST 0 PUT
      n n 2 →LIST 0 PUT m n 2 →LIST 1 PUT
      n m 2 →LIST 1 PUT *
«
```

Exemple:



ECHANGE DE DEUX LIGNES D'UNE MATRICE.

=====

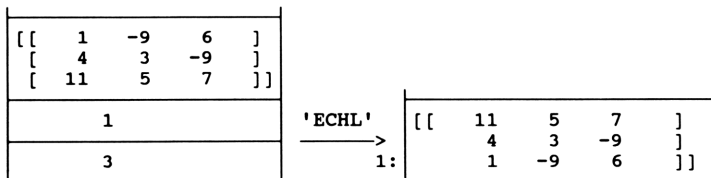
'EHL' échange les lignes de numéros n et m de la matrice A, transformant ainsi A en une matrice B.

Le schéma fonctionnel est analogue à celui de 'EHC'. D'ailleurs 'EHL' appelle 'EHC'.

'EHL': (Checksum: 27242# d, Taille: 38.5 octets)

```
«   ROT   TRN   3   ROLLD   EHC   TRN   »
```

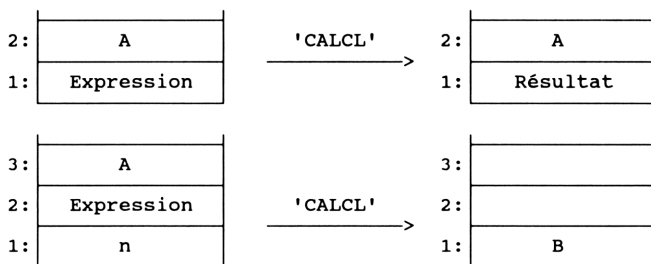
Exemple:



CALCULS SUR LES LIGNES D'UNE MATRICE.

=====

'CALCL' permet d'effectuer toutes sortes de calculs sur les lignes d'une matrice A, et éventuellement de modifier telle ou telle de ces lignes. Il y a deux schémas fonctionnels possibles:



Dans les deux cas, 'Expression' est une expression algébrique ou un programme (écrit en notation RPN) où les lignes de A sont désignées par les noms 'L1', 'L2', 'L3', etc...

Dans le premier cas, 'Résultat' désigne le résultat de l'évaluation de 'Expression'. Il peut s'agir d'un vecteur (au format des différentes lignes de A) ou de tout autre objet (notamment si 'Expression' est en fait un programme RPN). Si 'Expression' est un programme RPN et ne se comporte pas vis-à-vis de la pile comme une expression algébrique, l'aspect final de la pile peut être différent de celui qui est indiqué ci-dessus.

Dans le second cas, n désigne le numéro de la ligne de A qu'il faut modifier: le programme 'CALCL' place alors le résultat de l'évaluation de 'Expression' dans la n-ème ligne de A (il faut bien sûr que ce résultat soit un vecteur au format des différentes lignes de A), et 'B' désigne la matrice ainsi modifiée.

N.B: 'CALCL' crée des variables globales 'L1', 'L2', etc...
(autant que de lignes dans la matrice) avant de les purger.
'CALCL' appelle les programmes 'GETL' et 'PUTL'.

```
'CALCL': ( Checksum: # 64556d, Taille: 218.5 octets)
"  DUP  TYPE  NOT  DUP  DROPN  3  PICK  SIZE  1  GET
  "    " "L"  SWAP  →STR  +  OBJ→  "    RCLF
  →    n  t  p  f
  "    STD
    1  t  FOR  i
      OVER  i  GETL  i  p  EVAL  STO
  NEXT
  EVAL
    1  t  FOR  i  i  p  EVAL  PURGE  NEXT
  IF  n  THEN  n  PUTL  END
  f  STOP
"
"
```

PROGRAMME 'CALCL':
EXEMPLES D'UTILISATION.

=====

Exemple 1: en quatre secondes,

2:	$\begin{bmatrix} 1 & -1 & 3 & 0 \\ 0 & 5 & 2 & 4 \\ 6 & -2 & 8 & -7 \end{bmatrix}$		2:	$\begin{bmatrix} 1 & -1 & 3 & 0 \\ 0 & 5 & 2 & 4 \\ 6 & -2 & 8 & -7 \end{bmatrix}$
1:	'L1+2*L3-L2'	'CALCL' →	1:	$\begin{bmatrix} 13 & -10 & 17 & -18 \end{bmatrix}$

Dans cet exemple, on s'est contenté d'évaluer l'expression:
'L1+2*L3-L2', avec $L1 = \begin{bmatrix} 1 & -1 & 3 & 0 \end{bmatrix}$
 $L2 = \begin{bmatrix} 0 & 5 & 2 & 4 \end{bmatrix}$
et $L3 = \begin{bmatrix} 6 & -2 & 8 & -7 \end{bmatrix}$,
et de placer le résultat au niveau 1.

Exemple 2: en cinq secondes,

3:	$\begin{bmatrix} 1 & -1 & 3 & 0 \\ 0 & 5 & 2 & 4 \\ 6 & -2 & 8 & -7 \end{bmatrix}$			
2:	'L1+2*L3-L2'			
1:	1	'CALCL' →	1:	$\begin{bmatrix} 13 & -10 & 17 & -18 \\ 0 & 5 & 2 & 4 \\ 6 & -2 & 8 & -7 \end{bmatrix}$

Ici, on a repris les données précédentes, en demandant au programme 'CALCL' de placer le résultat dans la ligne 1 de la matrice.

Exemple 3: en quatre secondes,

2:	$\begin{bmatrix} 1 & -1 & 3 & 0 \\ 0 & 5 & 2 & 4 \\ 6 & -2 & 8 & -7 \end{bmatrix}$		4:	$\begin{bmatrix} 1 & -1 & 3 & 0 \\ 0 & 5 & 2 & 4 \\ 6 & -2 & 8 & -7 \end{bmatrix}$
			3:	1
			2:	32
1:	« L1 L2 DOT L1 L3 DOT L2 L3 DOT »	'CALCL' →	1:	-22

Sur cet exemple, on a évalué un programme qui calcule successivement les produits scalaires de L1 et L2, de L1 et L3, puis de L2 et L3.

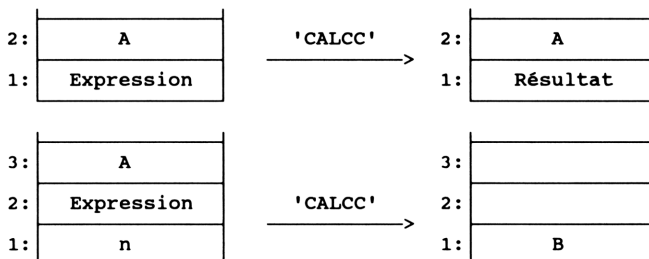
On voit qu'ici l'utilisation d'une expression algébrique est impossible (car l'instruction DOT ne peut y apparaître).

D'autre part, on voit bien que le schéma de la pile après exécution de 'CALCL' peut dépendre de ce que vous mettez dans le programme à évaluer.

CALCULS SUR LES COLONNES D'UNE MATRICE.

=====

'CALCC' permet d'effectuer toutes sortes de calculs sur les colonnes d'une matrice A, et éventuellement de modifier telle ou telle de ces colonnes. Il y a deux schémas fonctionnels possibles:



Dans les deux cas, 'Expression' est une expression algébrique ou un programme (écrit en notation RPN) où les colonnes de A sont désignées par les noms 'C1', 'C2', 'C3', etc...

Dans le premier cas, 'Résultat' désigne le résultat de l'évaluation de 'Expression'. Il peut s'agir d'une matrice uni-colonne (au format des différentes colonnes de A) ou de tout autre objet (notamment si 'Expression' est en fait un programme RPN). Si 'Expression' est un programme RPN et ne se comporte pas vis-à-vis de la pile comme une expression algébrique, l'aspect final de la pile peut être différent de celui qui est indiqué ci-dessus.

Dans le second cas, n désigne le numéro de la colonne de A qu'il faut modifier: le programme 'CALCC' place alors (programme PUTC) le résultat de l'évaluation de 'Expression' dans la n-ème colonne de A (il faut alors que ce résultat soit un vecteur ou une matrice uni-colonne), et 'B' désigne la matrice ainsi modifiée.

N.B: 'CALCC' crée des variables globales 'C1', 'C2', etc...

(autant que de colonnes dans la matrice) avant de les purger.

'CALCC' appelle les programmes 'GETC' et 'PUTC'.

```
'CALCC': ( Checksum: # 39066d, Taille: 218.5 octets)
"  DUP   TYPE  NOT  DUP   DROPN 3   PICK  SIZE  2   GET
  "  "C"  SWAP  +STR  +  OBJ→  "  RCLF
→  n  t  p  f
  "  STD
    1  t  FOR  i
      OVER  i  GETC  i  p  EVAL  STO
  NEXT
  EVAL
    1  t  FOR  i  i  p  EVAL  PURGE  NEXT
  IF  n  THEN  n  PUTC  END
  f  STOF
"
"
```

PROGRAMME 'CALCC': EXEMPLES D'UTILISATION.

=====

Exemple 1: en cinq secondes,

	1 -1 3 0 0 2 2 4 6 -2 8 -7																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
--	---	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

CREATION D'UN TABLEAU DONT LE TERME GENERAL REPOND A UNE FORMULE DONNEE.

=====

'CRTAB' crée un tableau (vecteur ou matrice) A dont le terme général (A(I) dans le cas d'un vecteur, A(I,J) dans le cas d'une matrice) est donné par une formule F(I) (cas d'un vecteur) ou F(I,J) (cas d'une matrice).

La taille du tableau à créer doit être placée au niveau 1.

La formule de calcul doit être placée au niveau 2; elle consiste en une expression algébrique ou en un programme, et pour une valeur de I (indice de ligne) et de J (indice de colonne) données, elle fournit la valeur de l'élément générique du tableau.

schéma fonctionnel (création d'un vecteur):



schéma fonctionnel (création d'une matrice):



'CRTAB': (Checksum: # 3883d, Taille: 147.5 octets)

```

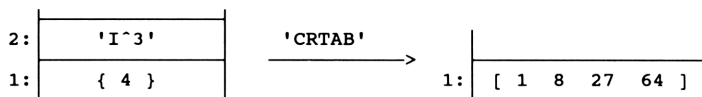
"  "  →  I  J"  ROT  +STR  +  OBJ→
→  d  f
"  d  OBJ→  DUP  1  ≠  DROPN
    →  L  c
    "  1  L  FOR  i
        1  c  FOR  j
            i  j  f  EVAL
        NEXT
    NEXT
    d  +ARRY
    "
"

```

EXEMPLES D'UTILISATION DU PROGRAMME 'CRATB'.

=====

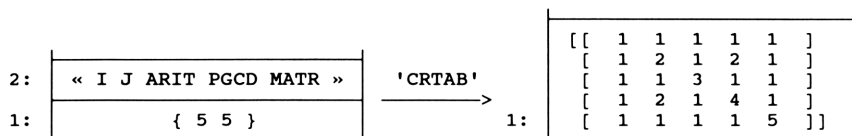
Exemple 1: Création d'un vecteur de longueur 4 de terme général $A(I)=I^3$. (temps de calcul inférieur à deux secondes).



Exemple 2: Création d'une matrice à 3 lignes et 4 colonnes dont le terme général $A(I,J)$ vaut I si $I=J$ et J^2 si $I \neq J$. (temps de calcul 4 secondes)



Exemple 3: Création de la matrice carrée d'ordre 5 dont le terme général est $A(I,J)=PGCD(I,J)$. Dans cet exemple le terme général est calculé au moyen d'un programme qui passe dans le répertoire 'ARIT', y calcule le PGCD de I et J , et revient dans le répertoire 'MATR'. (Temps de calcul: environ 4 secondes)



CALCUL DU POLYNOME CARACTERISTIQUE D'UNE MATRICE CARREE.

=====

'PCAR' calcule le polynôme caractéristique $P(x) = \det(xI - A)$ d'une matrice carrée A.

Si A est une matrice carrée d'ordre n, alors P(x) est un polynôme de degré n:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_k x^k + \dots + a_1 x + a_0.$$

En particulier:

$$a_n = 1, \quad a_{n-1} = -\text{tr}(A) \quad (\text{tr}(A) = \text{trace de } A), \quad a_0 = (-1)^n \det(A).$$

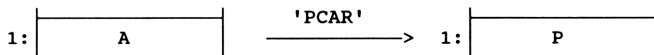
Les racines du polynôme caractéristique P de A sont aussi les valeurs propres de A.

Le polynôme P est obtenu sous la forme du vecteur:

$$[a_n \quad a_{n-1} \quad \dots \quad a_1 \quad a_0].$$

N.B: Le programme 'PCAR' appelle le programme 'TR' (calcul de la trace d'une matrice carrée).

schéma fonctionnel:

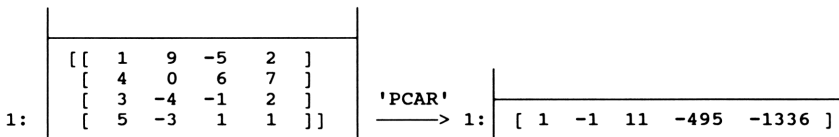


'PCAR': (Checksum: # 20812d, Taille: 137 octets)

```

«   DUP   SIZE   2   GET   DUP   IDN
  →   a   n   id
  «   1   id   1   n   FOR   k
      a   *   DUP   TR   k   /   NEG
      DUP   3   ROLL   id   *   +
      NEXT
      DROP   n   1   +   →ARRY
  »
»
  
```

Exemple: (temps de calcul 4 secondes)



VALEURS PROPRES D'UNE MATRICE CARREE D'ORDRE 2, 3, OU 4, A COEFFICIENTS REELS OU COMPLEXES.

=====

'VP234' Calcule les valeurs propres d'une matrice carrée A, d'ordre 2,3, ou 4, et à coefficients réels ou complexes.

Les valeurs propres de A sont les coefficients k tels que le système $A(X)=kX$ admette d'autres solutions que la solution nulle. Ce sont également les racines du polynôme caractéristique de A.

'VP234' procède de la manière suivante:

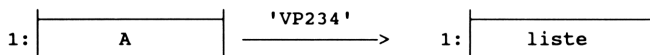
- calcul du polynôme caractéristique (programme 'PCAR').
- passage dans le répertoire 'POLY'.
- calcul des racines du polynôme caractéristique (on utilise 'DEG2', 'DEG3', 'DEG4' du répertoire 'POLY').
- retour dans le répertoire 'MATR'.

Les valeurs propres sont déposées dans une liste au niveau 1 de la pile. Une valeur propre multiple apparaît autant de fois que son ordre de multiplicité.

Les temps de calcul sont environ de:

- 2 secondes pour une matrice carrée d'ordre 2.
- 4 secondes pour une matrice carrée d'ordre 3.
- 10 secondes pour une matrice carrée d'ordre 4.

schéma fonctionnel:

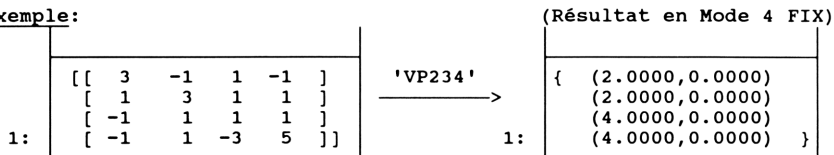


'VP234': (Checksum: # 4001d, Taille: 130.5 octets)

```

"  DUP   SIZE  2   GET
  →  n
  "  IF    { 2 3 4 }  n  POS  THEN
      PCAR  POLY  "DEG"  n  →STR  +
      1  4  SUB  OBJ→  MATR
  END
  n  →LIST
"
  
```

Exemple:



La matrice A ci-dessus admet donc 4 valeurs propres toutes réelles:

2 est une valeur propre double.

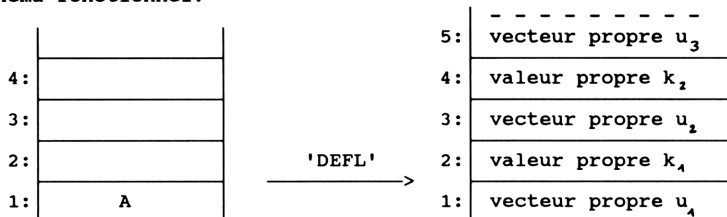
4 est une valeur propre double.

VALEURS PROPRES ET VECTEURS PROPRES D'UNE MATRICE CARREE REELLE (METHODE DE DEFLATION).

=====

'DEFL' permet de calculer, de manière approchée, les valeurs propres et vecteurs propres associés d'une matrice A, carrée d'ordre n, à coefficients réels, dans la mesure où A possède n valeurs propres réelles et distinctes $k_1, k_2, \dots, k_n$ telles que $|k_1| < |k_2| < \dots < |k_n|$.

Schéma fonctionnel:



'DEFL' est suspendu à chaque fois qu'une valeur propre k est obtenue. celle-ci est alors affichée au niveau 2 de la pile, alors qu'au niveau 1 apparaît un vecteur propre unitaire pour la valeur propre k. On poursuit alors la recherche par 'CONT', jusqu'à obtention de la dernière valeur propre.

'DEFL' utilise un algorithme itératif de recherche des valeurs propres. Le test d'arrêt des itérations est donné par une valeur 'eps' égale par défaut à 1E-8 (voir cette valeur dans le texte du programme). On peut modifier cette donnée, en plaçant une nouvelle valeur de 'eps' au niveau 1 de la pile (avant d'appeler 'DEFL'), la matrice A remontant alors au niveau 2. Plus 'eps' est petit, meilleure est l'approximation, mais les temps de calculs augmentent.

N.B: les valeurs propres sont calculées dans l'ordre de leurs valeurs absolues décroissantes. Le calcul d'une valeur propre est d'autant plus rapide qu'est grande sa valeur absolue, relativement aux valeurs propres de moindre valeur absolue.

Exemple:

avec la matrice A égale à:

```
[ [ 1  -9  -9  -9 ]
  [-13 19  0  22 ]
  [-13 22  -3  22 ]
  [ 26 -31 13 -34 ] ]
```

On trouve successivement (en mode 8 FIX):

```
En 12 secondes: k =-24.9999989 et
u = [ -2.0116359E-9  0.40824829  0.40824829  -0.81649658 ].
En 24 secondes: k =10.00000001 et
u = [ -0.500000000  0.500000000  0.500000000  -0.500000000 ].
En 23 secondes: k =-3.00000000 et
u = [ 3.97471899E-9  -0.70710678  -3.58306366E-9  0.70710678 ].
En 15 secondes: k =1.00000000 et
u = [ 0.63245553  -0.316222777  -0.316222777  0.63245553 ].
```

On peut donc dire que les valeurs propres de A sont -25, 10, -3, 1, et que des vecteurs propres respectivement associés à ces valeurs propres sont: [0,1,1,-2], [-1,1,1,-1], [0,1,0,-1], [2,-1,-1,2].

TEXTE DU PROGRAMME 'DEFL'

=====

'DEFL': (Checksum: # 56093d, Taille: 485 octets)

```

«   IF DUP TYPE THEN .00000001 END
    SWAP DUP SIZE 2 2 SUB DUP 0 CON
    DUP ROT 1 GET

→   eps a old new p
«
  « → mat
    « old 1
      1 p START RAND PUTI NEXT DROP
      DO
        'old' STO mat old * DUP ABS /
        IF DUP old DOT 0 < THEN NEG END
        DUP 'new' STO
        UNTIL 'ABS(new-old)<eps' END
        mat new * DOT new ABS / new
      »
    »
→   vmax
«   1 p FOR k
    a vmax EVAL
    IF k p < THEN
      HALT
      DUP2 a TRN vmax EVAL
      ROT DUP2 DOT /
      DUP SIZE 1 + RDM SWAP
      1 OVER SIZE + RDM *
      3 ROLL + 2 / *
      'a' SWAP STO-
    END
    NEXT
  »
»
»

```

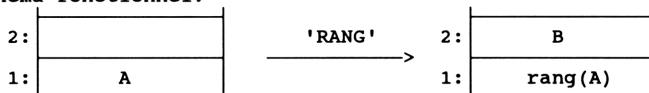
N.B: dans l'exemple d'utilisation donné plus haut, les résultats dépendent du générateur de nombres aléatoires de la HP48, et peuvent donc différer de ceux que j'ai personnellement obtenu.

CALCUL DU RANG D'UNE MATRICE.

=====

'RANG' calcule le rang d'une matrice A (c'est à dire le nombre de colonnes ou de lignes linéairement indépendantes dans A) après avoir (par la méthode dite "du pivot de Gauss") transformé A en une matrice triangulaire supérieure B.

schéma fonctionnel:



N.B: le programme 'RANG' appelle le programme 'ECHL'.

'RANG' utilise un arrondi à la 7-ème décimale, pour éviter d'être trompé par des erreurs d'arrondis et d'utiliser un pivot trop petit.

De même la forme finale de la matrice est arrondie à la 9ème décimale, pour que les coefficients nuls en théorie n'apparaissent pas.

Les instructions "7 RND" et "9 RND" peuvent être modifiées si vous jugez que ces arrondis sont trop ou pas assez sévères.

'RANG': (Checksum: # 62920d, Taille: 582 octets)

```

"  DUP TYPE NOT DUP DROPN OVER SIZE EVAL
  1 1 0 0 → f nl nc i j rg lp
"    " 0 i nl FOR ii
      OVER ii j 2 →LIST GET ABS DUP2
      IF < THEN ii 'lp' STO SWAP END
      DROP NEXT
"
"  DUP i j 2 →LIST GET → piv
"    nl IDN
      1 f NOT i * + nl FOR ii
      IF ii i ≠ THEN
        ii i 2 →LIST 3 PICK
        ii j 2 →LIST GET piv
        / NEG PUT
      END
      NEXT SWAP *
"
" → cp pv
" WHILE i nl ≤ j nc ≤ AND
  REPEAT cp EVAL
    IF 7 RND THEN
      'rg' 1 STO+
      IF i lp < THEN i lp ECHL END
      IF i nl < f OR THEN pv EVAL END
      'i' 1 STO+
    END
    'j' 1 STO+
  END
  9 RND rg
"  "  "

```

EXEMPLES D'UTILISATION DU PROGRAMME 'RANG'.

=====

Exemple 1: (en environ 5 secondes)

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Exemple 2: (en environ 13 secondes)

La matrice A égale à

[	[	18	-12	10	11	-9	-19	10	]
	[	4	5	9	5	2	1	3	]
	[	1	8	5	2	4	6	1	]
	[	10	5	3	4	1	2	5	]
	[	-6	0	6	1	1	-1	-2	]

est de rang 3, et la matrice triangulaire B obtenue s'écrit,
(en mode 2 FIX):

[	[	18	-12	10	11	-9	-19	10	]
	[	0	11.67	-2.56	-2.11	6	12.56	-.56	]
	[	0	0	8.46	3.94	.06	-3.03	1.14	]
	[	0	0	0	0	0	0	0	]
	[	0	0	0	0	0	0	0	]

Variante:

Si, avant d'appeler 'RANG', on place un réel non nul au niveau 1 de la pile, (la matrice A passant alors au niveau 2), la matrice B obtenue à la sortie a subi une "triangulation" plus poussée: tous les coefficients situés au dessus d'un pivot non nul ont été annulés.

Cette variante est utilisée dans les programmes 'SYST' (résolution symbolique d'un système linéaire) et 'VECTP' (équations de sous-espaces propres). Dans l'exemple ci dessus, la matrice obtenue par cette variante est: (toujours en mode 2 FIX)

[	[	18	0	0	5.39	-2.88	-3.45	8.43	]
	[	0	11.67	0	-.92	6.02	11.64	-.21	]
	[	0	0	8.46	3.94	.06	-3.03	1.14	]
	[	0	0	0	0	0	0	0	]
	[	0	0	0	0	0	0	0	]

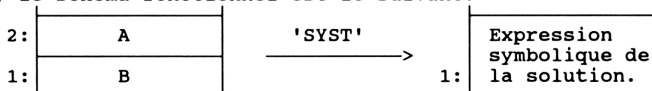
RESOLUTION SYMBOLIQUE D'UN SYSTEME D'EQUATIONS LINEAIRES.

=====

'SYST' permet d'obtenir la solution symbolique d'un système de n équations linéaires à p inconnues:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1p}x_p = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2p}x_p = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mp}x_p = b_m \end{cases}$$

Si on note A la matrice du système et B le vecteur des seconds membres, le schéma fonctionnel est le suivant:



L'expression symbolique de la solution est:

- Le message "Pas de solution" si le système est impossible.
- Une liste contenant la ou les égalités définissant la solution générale du système: ces égalités peuvent être de la forme :
'X3=-17' (par exemple) si -17 est la seule valeur de l'inconnue X3 qui satisfasse au système.

et/ou de la forme:

'X2=3*X4-2*X5' (par exemple) si la résolution du système conduit à cette expression de l'inconnue X2 en fonction des inconnues X4 et X5 (ces deux-ci étant alors indéterminées).
(Les inconnues du système sont notées X1, X2, X3, ...)

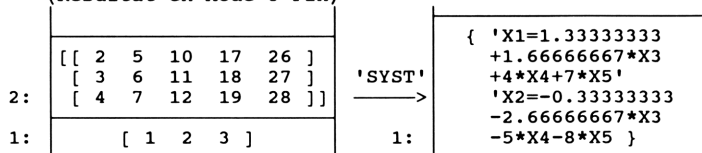
N.B.: le programme 'SYST' appelle le programme 'RANG'.

Exemple : (en 16 secondes)

Résolution du système

$$\begin{cases} 2X_1 + 5X_2 + 10X_3 + 17X_4 + 26X_5 = 1 \\ 3X_1 + 6X_2 + 11X_3 + 18X_4 + 27X_5 = 2 \\ 4X_1 + 7X_2 + 12X_3 + 19X_4 + 28X_5 = 3 \end{cases}$$

(Résultat en Mode 8 FIX)



Le système précédent admet donc une infinité de solutions: X3, X4, X5 ont des valeurs arbitraires et X1 et X2 sont respectivement données par:

$$\begin{aligned} X &= 4/3 + 5X_3/3 + 4X_4 + 7X_5 \\ X &= -1/3 - 8X_3/3 - 5X_4 - 8X_5 \end{aligned}$$

TEXTE DU PROGRAMME 'SYST'

=====

'SYST': (Checksum: # 34133d, Taille: 468 octets)

```

«   →STR  SWAP  TRN  →STR  1  OVER  SIZE  1  -  SUB
    SWAP  +  OBJ→  TRN  1  RANG  OVER  SIZE  2  GET
    «   RCLF  STD  "X"  ROT  →STR  +  OBJ→  SWAP  STOF  »

→   a  rg  nc  var

«   { }
    1  rg  FOR  i

        a  i  { 1 }  +  0
        DO  DROP  GETI  UNTIL  DUP  EVAL  END
        SWAP  2  GET
        IF  DUP  1  ==  THEN
            3  DROPN  "Pas de solution"  DOERR
        END
        1  -

→   piv  j

«   DROP  j  var  EVAL  'a(i,nc)/piv'  EVAL
    IF  j  1  +  nc  <  THEN
        j  1  +  nc  1  -  FOR  jj
            '-a(i,jj)/piv'  EVAL
            jj  var  EVAL  *  +
        NEXT
    END
    »
    =  +

NEXT

»
»
»

```

EQUATIONS DE SOUS-ESPACES PROPRES D'UNE MATRICE CARREE

=====

'VECTP' permet d'obtenir la ou les équations du sous-espace propre d'une matrice carrée A, relativement à une valeur propre k. Pour cela, le programme 'SYST' est appelé afin de résoudre symboliquement le système:

$(A - kI)X = 0$, où I est la matrice identité de même dimension que A, et où X est un vecteur dont les composantes sont notées X1, X2, X3,

schéma fonctionnel:



La ou les équations sont données dans une liste, en respectant la syntaxe du programme 'SYST'. Si k n'est pas réellement une valeur propre de A, le résultat est { 'X1=0' 'X2=0' 'X3=0' ... }

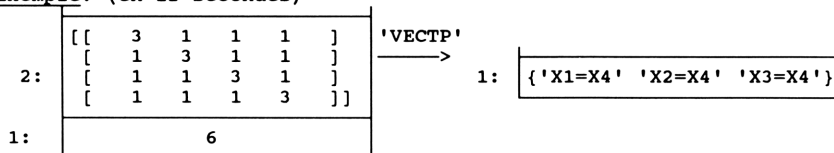
'VECTP': (Checksum: # 44125d, Taille: 54.5 octets)

```

«   OVER   IDN   *   -   DUP   SIZE
   1   1   SUB   0   CON   SYST
»

```

Exemple: (en 12 secondes)



Autrement dit, 6 est une valeur propre simple de la matrice et le sous-espace propre est la droite vectorielle engendré par le vecteur : [1 1 1 1] (obtenu en donnant la valeur 1 à la variable X4).

De même, avec la même matrice et avec k=2, le résultat est:

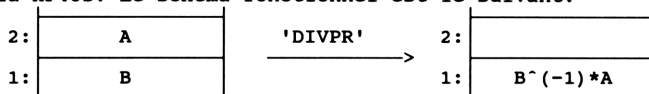
{ 'X1=-X2-X3-X4' },

ce qui signifie que 2 est une valeur propre triple, le sous-espace propre étant engendré par les vecteurs [-1 0 0 1], [-1 0 1 0], [-1 1 0 0] (obtenus en donnant la valeur 1 à l'une des variables X2, X3, X4, et la valeur 0 aux deux autres).

DIVISION DE MATRICES AVEC PRECISION ACCENTUEE.

=====

'DIVPR' permet d'augmenter la précision de la commande / (division d'un tableau par une matrice), en utilisant l'instruction RSD de la HP48S. Le schéma fonctionnel est le suivant:



où A est un tableau et où B est une matrice carrée.

'DIVPR': (Checksum: # 63411d, Taille: 44.5 octets)

« DUP2 / 3 DUPN RSD ROT / + SWAP DROP »

Exemple: Avec A = $\begin{bmatrix} 38 & 61 \\ 87 & 67 \\ -31 & -7 \end{bmatrix}$ et B = $\begin{bmatrix} 5 & -3 & 7 \\ 8 & 1 & 6 \\ 4 & -5 & -2 \end{bmatrix}$ on obtient

Par utilisation de / :

$B^{(-1)} * A = \begin{bmatrix} 5.999999999998 & 2.999999999999 \\ 9.000000000002 & .999999999994 \\ 5.000000000003 & 7.000000000001 \end{bmatrix}$

et par utilisation de 'DIVPR':

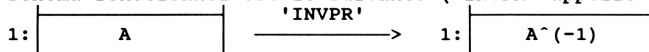
$B^{(-1)} * A = \begin{bmatrix} 6 & 3 \\ 9 & 1 \\ 5 & 7 \end{bmatrix}$ ce qui est le résultat exact.

INVERSION DE MATRICES AVEC PRECISION ACCENTUEE.

=====

'INVPR' permet d'inverser une matrice carrée A avec une meilleure précision que celle obtenue en utilisant le commande INV (doublée au clavier par la touche 1/x).

Le schéma fonctionnel est le suivant: ('INVPR' appelle 'DIVPR')



'INVPR': (Checksum: # 38687d, Taille: 35.5 octets)

« DUP IDN SWAP DIVPR »

Exemple: Avec A = $\begin{bmatrix} -2 & -3 & 2 \\ 6 & 1 & 6 \\ 4 & 5 & -2 \end{bmatrix}$, on trouve

En utilisant INV:

$A^{(-1)} = \begin{bmatrix} -4.000000000022 & .500000000027 & -2.500000000014 \\ 4.500000000026 & -.500000000031 & 3.000000000016 \\ 3.250000000018 & -.250000000021 & 2.000000000011 \end{bmatrix}$

et en utilisant 'INVPR':

$A^{(-1)} = \begin{bmatrix} -4 & .5 & -2.5 \\ 4.5 & -.5 & 3 \\ 3.25 & -.25 & 2 \end{bmatrix}$ ce qui est le résultat exact.

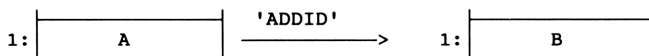
BORDER UNE MATRICE A DROITE PAR LA MATRICE IDENTITE.

=====

'ADDID' permet de border, à droite, une matrice A de format {n,p} par la matrice identité d'ordre n. Le résultat est une matrice B ayant donc le format { n, n+p }.

'ADDID' est appelé dans le programme 'EQRL'. Il est très utile également pour mener à bien l'inversion d'une matrice par la méthode du pivot de Gauss (voir programme 'PIVOT').

Le schéma fonctionnel est:

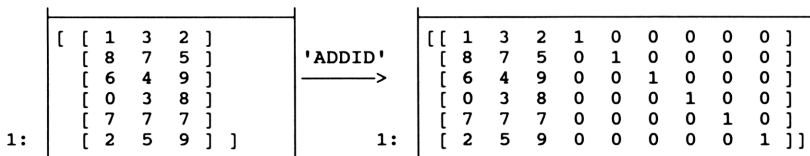


'ADDID': (Checksum: # 38199d, Taille: 135.5 octets)

```

“   TRN   DUP   SIZE   EVAL
  →   p     n
“   p     n   +   n   2   →LIST   RDM
   p     n   *   1   +   n   p   +   n   *   FOR   i
     i     1   PUT
   n     1   +   STEP
   TRN
”
”
  
```

Exemple: (en une seconde)

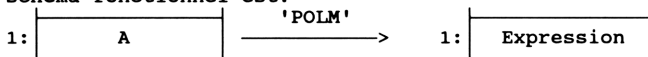


POLYNOME MINIMAL D'UNE MATRICE CARREE.

=====

'POLM' permet de calculer le polynôme minimal P d'une matrice carrée A, c'est à dire le polynôme unitaire de degré minimum tel que $P(A)=0$ (Le polynôme P divise le polynôme caractéristique de A. Voir à ce sujet le programme 'PCAR' dans le répertoire 'MATR').

Le schéma fonctionnel est:



où 'Expression' est une expression algébrique représentant l'égalité $P(A)=0$. Si par exemple le polynôme minimal de A est $P = X^3 - 3X + 2$, alors l'expression obtenue est:

'A^3-3*A+2*I=0',

où la lettre I est mise pour représenter la matrice identité de même format que A.

N.B: 'POLM' appelle 'ADDID', 'RANG', 'GETL' dans le répertoire 'MATR', et 'ELMG' dans le répertoire 'POLY'.

'POLM': (Checksum: # 24551d, Taille: 341.5 octets)

```

«  DUP      SIZE  1  GET      DUP  IDN
   →  a      n      b
   «  n      1  +      DUP  2  →LIST  0  CON
      n      1  +      DUP  SQ   n      -  FOR  i
      i      1  PUT
      n  STEP
      0  n  START
      b  OBJ→  DROP  'b'  a  STO*
      NEXT
      n  1  +  n  SQ  2  →LIST  →ARRY  *
      ADDID  RANG  DROP  n  1  +  GETL
      POLY  ELMG  MATR  DUP  1  GET  /  OBJ→
      1  GET  1  +  0
      SWAP  3  FOR  i
      i  ROLL  'A'  i  2  -  ^  *  +
      -1  STEP
      SWAP  'I'  *  +  0  =
   »
»
  
```

Remarque: les coefficients obtenus sont réels et susceptibles de comporter des erreurs d'arrondis. Si les coefficients de A sont rationnels, ceux du polynôme minimal P le sont aussi (Mieux: Si A est à coefficients entiers, P aussi).

Il semble judicieux dans ces cas là de forcer l'approximation rationnelle des coefficients obtenus, avec l'instruction →Q, après être passé en mode n FIX (l'entier n étant choisi de façon à juste contrer les erreurs d'arrondi).

PROGRAMME 'POLM': EXEMPLES D'UTILISATION.

=====

Remarque :

Le temps d'exécution du programme 'POLM', et la place qui lui est nécessaire pour fonctionner en mémoire augmente de façon importante avec l'ordre de la matrice carrée A dont on veut le polynôme caractéristique.

Ainsi le temps de calcul est approximativement de:

12 secondes pour une matrice carrée d'ordre 3.

25 secondes pour une matrice carrée d'ordre 4.

Exemple 1:

		(Résultat obtenu en mode STD)						
1:	<table border="1"> <tr><td>[[3 -1 1]</td></tr> <tr><td>[-1 3 1]</td></tr> <tr><td>[1 1 3]]</td></tr> </table>	[[3 -1 1]	[-1 3 1]	[1 1 3]]	<table border="1"> <tr><td></td></tr> <tr><td>'POLM'</td></tr> <tr><td>→ 1: 'A^2-5*A+4*I=0'</td></tr> </table>		'POLM'	→ 1: 'A^2-5*A+4*I=0'
[[3 -1 1]								
[-1 3 1]								
[1 1 3]]								
'POLM'								
→ 1: 'A^2-5*A+4*I=0'								

Exemple 2:

	(Résultat obtenu avec →Q en mode 8 FIX)			
1:	<table border="1"> <tr><td>[[1 1 1]</td></tr> <tr><td>[1 1 1]</td></tr> <tr><td>[1 1 1]]</td></tr> </table>	[[1 1 1]	[1 1 1]	[1 1 1]]
[[1 1 1]				
[1 1 1]				
[1 1 1]]				
	<table border="1"> <tr><td></td></tr> <tr><td>'POLM'</td></tr> <tr><td>→ 1: 'A^2-3*A=0'</td></tr> </table>		'POLM'	→ 1: 'A^2-3*A=0'
'POLM'				
→ 1: 'A^2-3*A=0'				

Exemple 3:

	(Résultat obtenu en mode STD)				
1:	<table border="1"> <tr><td>[[5 1 3 -2]</td></tr> <tr><td>[4 -1 6 -4]</td></tr> <tr><td>[3 1 1 -1]</td></tr> <tr><td>[-2 -1 -1 3]]</td></tr> </table>	[[5 1 3 -2]	[4 -1 6 -4]	[3 1 1 -1]	[-2 -1 -1 3]]
[[5 1 3 -2]					
[4 -1 6 -4]					
[3 1 1 -1]					
[-2 -1 -1 3]]					
	<table border="1"> <tr><td></td></tr> <tr><td>'POLM'</td></tr> <tr><td>→ 1: 'A^4-8*A^3-14*A^2+50*A+3*I=0'</td></tr> </table>		'POLM'	→ 1: 'A^4-8*A^3-14*A^2+50*A+3*I=0'	
'POLM'					
→ 1: 'A^4-8*A^3-14*A^2+50*A+3*I=0'					

Exemple 4:

	(Résultat obtenu en mode STD)				
1:	<table border="1"> <tr><td>[[-142 21 -217 213]</td></tr> <tr><td>[300 -47 463 -451]</td></tr> <tr><td>[190 -30 293 -284]</td></tr> <tr><td>[68 -12 107 -102]]</td></tr> </table>	[[-142 21 -217 213]	[300 -47 463 -451]	[190 -30 293 -284]	[68 -12 107 -102]]
[[-142 21 -217 213]					
[300 -47 463 -451]					
[190 -30 293 -284]					
[68 -12 107 -102]]					
	<table border="1"> <tr><td></td></tr> <tr><td>'POLM'</td></tr> <tr><td>→ 1: 'A^4-2*A^3+A^2=0'</td></tr> </table>		'POLM'	→ 1: 'A^4-2*A^3+A^2=0'	
'POLM'					
→ 1: 'A^4-2*A^3+A^2=0'					

METHODE DU PIVOT DE GAUSS.

=====

'PIVOT' permet d'appliquer la méthode du pivot à une matrice A.
Il y a deux possibilités, suivant que l'on souhaite pousser plus
ou moins loin la méthode:

Premier cas:

Chaque pivot rencontré sert à annuler tous les termes situés en-dessous de lui.
La matrice A sera ainsi peu à peu transformée en une matrice dont tous les coefficients situés en dessous de la diagonale sont nuls (plus exactement tous les coefficients dont l'indice de ligne est supérieur à l'indice de colonne).
Cette méthode est utile pour déterminer le rang de la matrice, ou pour transformer un système d'équations linéaires en système en "cascades".

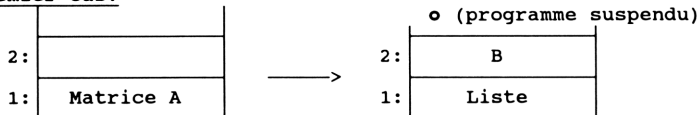
Deuxième cas:

Chaque pivot rencontré sert à annuler tous les termes situés en-dessous et au dessus de lui.
Dans chaque colonne où un pivot a été utilisé, ce pivot reste donc le seul coefficient non nul.
Cette méthode est employée pour inverser une matrice carrée A (au préalable, il conviendra de border A, à droite, par la matrice identité, au moyen du programme 'ADDID').
Par cette méthode, on peut ainsi résoudre un système d'équations en rendant diagonale la matrice du système.

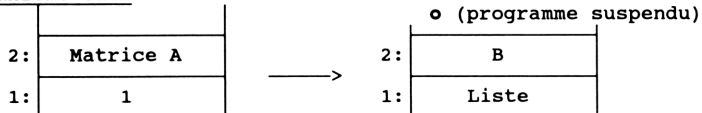
Supposons que A est une matrice de format {n,p} (n lignes, p colonnes):
à la k-ème étape de la méthode, le pivot utilisé est le terme à l'intersection de la k-ème ligne et de la k-ème colonne.

Il appartient à l'utilisateur de procéder au besoin à un échange de lignes (programme 'ECHL') si d'aventure le coefficient devant être utilisé comme pivot est nul (dans un tel cas un message apparaît pour vous permettre de remédier à cette situation).

Il y a deux schémas fonctionnels correspondant aux deux cas ci-dessus:

Premier cas:

(Dans ce cas on se contente d'annuler les coefficients situés sous la diagonale).

Deuxième cas:

(Dans ce cas tous les coefficients situés de part et d'autre du pivot, sur sa colonne, seront annulés).

Le programme 'PIVOT' est suspendu à chaque étape du calcul, c'est-à-dire après qu'un pivot ait été complètement utilisé.

Les différentes opérations réalisées avec ce pivot sont alors présentées dans une liste, sous forme d'expressions algébriques.

Pour prendre un exemple, l'expression:

'L4=2*L4-3*L2'

signifie que la 4-ème ligne (notée L4) a été remplacée par une ligne obtenue en effectuant l'opération suivante:

"2 fois la ligne L4 - 3 fois la ligne L2".

Une telle situation signifie que la ligne du pivot est la 2-ème ligne, et que par exemple les lignes N° 2 et 4 de la matrice avaient la forme suivante:

L1 ->	[[* * * * *]	L'opération décrite ci-dessus se
L2 ->	[0 2 * * *]	traduit donc par l'annulation
L3 ->	[0 * * * *]	du coefficient égal à 3 dans
L4 ->	[0 3 * * *]	la ligne L4.
	 etc	

Le programme étant suspendu à l'issue d'une étape de la méthode, on a donc la liste des opérations effectuées au niveau 1, et la matrice modifiée au niveau 2.

Il suffit alors de recopier sur papier cette liste et cette matrice modifiée, avant de faire DROP (pour supprimer cette liste) puis CONT pour poursuivre l'exécution du programme pour un nouveau pivotage. Quand les différentes étapes ont été réalisées, le programme s'arrête.

Remarque:

Le programme 'PIVOT' teste constamment s'il a affaire, dans la matrice, à des coefficients entiers.

Le cas échéant, il procède à des simplifications au cours des opérations de pivotage (par appel du programme 'SIMP' dans le répertoire 'ARIT' (l'intérêt étant que les opérations réalisées sont les plus simples possibles).

Exemple:

Dans la situation	L1 ->	[[* * * * *]
ci-contre, l'opération	L2 ->	[0 8 * * *]
qui permet d'annuler le	L3 ->	[0 * * * *]
coefficient 6 de L4 est:	L4 ->	[0 6 * * *]
		 etc
'L4 = 4*L4 - 3*L2'		
(ce qui est plus simple que 'L4 = 8*L4 - 6*L2')		

Remarque:

Le programme 'PIVOT' appelle les programme 'SIMP' dans le répertoire 'ARIT'.

TEXTE DU PROGRAMME 'PIVOT'

=====

'PIVOT': (Checksum: # 13960d, Taille: 538.5 octets)

```

«   DUP   TYPE   NOT   DUP   DROPN   OVER   SIZE   EVAL
   «   RCLF   STD   "'L"   ROT   →STR   +   OBJ→   SWAP   STOF   »

→   flag   lig   col   L

«   1   lig   flag   NOT   -   col   MIN   FOR   j
   WHILE   DUP   j   j   2   →LIST   GET   NOT
   REPEAT   "Pivot nul"   HALT   END
   DUP   j   GETL   DUP   j   GET

→   Lj   piv

«   { }
   1   flag   NOT   j   *   +   lig   FOR   i
   IF   i   j   ≠   THEN
       OVER   i   GETL   DUP   j   GET
       IF   DUP   THEN
           piv
           IF   DUP   0   <   THEN
               R+C   NEG   C→R
           END
           IF   OVER   FP   NOT   OVER   FP   NOT   AND
               THEN   ARIT   SIMP   MATR
           END
           DUP2   SWAP   i   L   EVAL   ROT   OVER   *
           ROT   j   L   EVAL   *   -   =
           5   ROLL   SWAP   +   5   ROLLD
           ROT   *   Lj   ROT   *   -   i   PUTL   SWAP
       ELSE
           DROP2
       END
   END
NEXT
»

HALT

NEXT
»
»

```

PROGRAMME 'PIVOT': EXEMPLE D'UTILISATION.

=====

Exemple 1:

On veut calculer l'inverse de la matrice $A = \begin{bmatrix} 4 & 3 & 8 \\ 2 & 5 & 1 \\ 6 & 3 & 2 \end{bmatrix}$

On place la matrice A au niveau 1. On appelle le programme 'ADDID' ce qui a pour effet de border à droite la matrice A par la matrice identité d'ordre 3.

On place 1 au niveau 1, la matrice remontant au niveau 2, et on appelle le programme 'PIVOT'. On obtient les résultats suivants sur la pile (pour franchir une étape supplémentaire, on élimine la liste au niveau 1 avant de faire CONT)

2:	$\begin{bmatrix} 4 & 3 & 8 & 1 & 0 & 0 \\ 2 & 5 & 1 & 0 & 0 & 0 \\ 6 & 3 & 2 & 0 & 0 & 0 \end{bmatrix}$
1:	{ 'L2=2*L2-L1' 'L3=2*L3-3*L1' }

2:	$\begin{bmatrix} 28 & 0 & 74 & 10 & -6 & 0 \\ 0 & 7 & -6 & -1 & 2 & 0 \\ 0 & 0 & -158 & -24 & 6 & 14 \end{bmatrix}$
1:	{ 'L1=7*L1-3*L2' 'L3=7*L3+3*L2' }

2:	$\begin{bmatrix} 2212 & 0 & 0 & -98 & -252 & 518 \\ 0 & 553 & 0 & -7 & 140 & -42 \\ 0 & 0 & -158 & -24 & 6 & 14 \end{bmatrix}$
1:	{ 'L1=79*L1+37*L3' 'L2=79*L2-3*L3' }

Il suffit alors de diviser la première ligne par 2212, la deuxième par 553, la 3-ème par -158, pour obtenir dans la partie droite du tableau, l'expression de l'inverse de la matrice A initiale.

Faire par exemple, pour la ligne 1 (après avoir détruit la liste):

DUP 1 GETL 2212 / 1 PUTL
Plus précisément, en utilisant le
programme 'T->Q' du répertoire 'R.C'
sur chaque ligne, on trouve:

$$A^{-1} = \frac{1}{158} \begin{bmatrix} -7 & -18 & 37 \\ -2 & 40 & -12 \\ 24 & -6 & -14 \end{bmatrix}$$

PROGRAMME 'PIVOT': EXEMPLE D'UTILISATION.

=====

Exemple 2:

On veut calculer le rang de la famille de vecteurs:

U1 = [1 5 9 13 17],
 U2 = [3 7 11 15 19],
 U3 = [2 6 1 0 11],
 U4 = [1 3 14 21 16].

On place la matrice:

au niveau 1 et on

appelle le

programme 'PIVOT'

[[1 5 9 13 17]
 [3 7 11 15 19]
 [2 6 1 0 11]
 [1 3 14 21 16]]

On obtient les différentes étapes suivantes:

2:	[[1 5 9 13 17] [0 -8 -16 -24 -32] [0 -4 -17 -26 -23] [0 -2 5 8 -1]]
1:	{ 'L2=L2-3*L1' 'L3=L3-2*L1' 'L4=L4-L1' }

2:	[[1 5 9 13 17] [0 -8 -16 -24 -32] [0 0 18 28 14] [0 0 -36 -56 -28]]
1:	{ 'L3=2*L3-L2' 'L4=4*L4-L2' }

2:	[[1 5 9 13 17] [0 -8 -16 -24 -32] [0 0 -18 -28 -14] [0 0 0 0 0]]
1:	{ 'L4=L4+2*L3' }

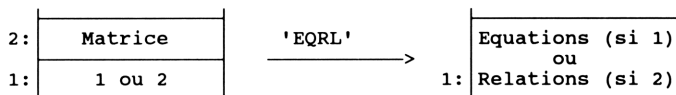
Cette dernière étape montre que la famille de vecteurs U1,U2,U3,U4, est de rang 3 (la triangulation s'achève avec trois pivots non nuls exactement).

RECHERCHE DE RELATIONS OU D'EQUATIONS LINEAIRES.

=====

'EQRL' est un programme dont le but est de rechercher les équations linéaires d'un espace vectoriel, ou les relations linéaires existant éventuellement entre des vecteurs donnés.

Il y a deux schémas fonctionnels possibles suivant l'usage qui veut être fait de ce programme:



'EQRL' appelle les programmes 'ADDID' et 'RANG'.

Pour plus de détails, voir les exemples d'utilisation dans les pages suivantes.

'EQRL' : (Checksum: # 14068d, Taille: 309 octets)

```

" { " TRN "'X" " "'U" } SWAP GET EVAL
OVER SIZE EVAL → ch n p
" DUP ADDID 1 RANG ROT RANG SWAP DROP -
  " RCLF STD ch ROT →STR + OBJ→ SWAP STOF "
  → m u
  " { } IF m THEN
    n m - 1 + n FOR i
      0 1 n FOR j
        3 PICK i p j + 2 →LIST
        GET j u EVAL * +
      NEXT 0 = +
    NEXT
  END SWAP DROP
"
"

```

Remarque:

Il est possible de forcer l'approximation en nombres rationnels des coefficients apparaissant dans les relations obtenues, en utilisant l'instruction →Q (après être passé dans un certain mode n FIX, de manière à contrer les erreurs d'arrondi).

PROGRAMME 'EQRL': RECHERCHE D'EQUATIONS LINEAIRES.

=====

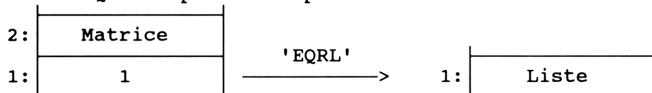
Le problème est le suivant: étant donnés n vecteurs $U_1, U_2, \dots, U_n$, ayant tous p coefficients (donc des vecteurs de $\mathbb{R}^p$), ces vecteurs forment-ils une famille génératrice de $\mathbb{R}^p$? Sinon, quelles sont les équations linéaires du sous-espace vectoriel de $\mathbb{R}^p$ qu'ils engendrent?

On appelle rang de cette famille de vecteurs la dimension r de ce sous-espace vectoriel:

Si $r=p$, les vecteurs U_k sont générateurs dans $\mathbb{R}^p$.

Si $r < p$, le sous-espace vectoriel qu'ils engendrent est défini par un système de $p-r$ équations linéaires indépendantes.

Le programme 'EQRL' répond à ce problème avec le schéma fonctionnel:



'Matrice' est une matrice dont les n différentes lignes sont les n vecteurs U_k (ayant chacun p coefficients).

'Liste' est une liste constituée des équations trouvées:

Cette liste est vide si les vecteurs U_k engendrent tout $\mathbb{R}^p$.

Si le système des n vecteurs U_k est de rang $r < p$, cette liste est constituée des $p-r$ équations linéaires indépendantes définissant le sous-espace engendré par les vecteurs U_k .

Chacune des équations est écrite sous forme d'une expression algébrique reliant les coordonnées notées X_1, X_2 , etc....

Exemple:

On veut déterminer le sous-espace de $\mathbb{R}^5$ engendré par les vecteurs:

$$U_1 = \begin{bmatrix} 1 & 5 & 9 & 13 & 17 \end{bmatrix}$$

$$U_2 = \begin{bmatrix} 3 & 7 & 11 & 15 & 19 \end{bmatrix}$$

$$U_3 = \begin{bmatrix} 2 & 3 & 4 & 5 & 6 \end{bmatrix}$$

$$U_4 = \begin{bmatrix} 3 & 9 & 15 & 21 & 27 \end{bmatrix}$$

On place donc au niveau 2 la matrice:

$$A = \begin{bmatrix} \begin{bmatrix} 1 & 5 & 9 & 13 & 17 \end{bmatrix} \\ \begin{bmatrix} 3 & 7 & 11 & 15 & 19 \end{bmatrix} \\ \begin{bmatrix} 2 & 3 & 4 & 5 & 6 \end{bmatrix} \\ \begin{bmatrix} 3 & 9 & 15 & 21 & 27 \end{bmatrix} \end{bmatrix}$$

On place au niveau 1 l'entier 1 et on appelle 'EQRL'.

Au bout de 32 secondes, on obtient, au niveau 1 de la pile:

```
{
  '-(.75*X1)+3*X4-2.25*X5=0'
  '-(.66666667*X2)+2*X4-1.33333333*X5=0'
  '-(.5*X3)+X4-.5*X5=0' }
```

Ces trois relations peuvent s'écrire plus simplement:

$$X_1 - 4X_4 + 3X_5 = 0, \quad X_2 - 3X_4 + 2X_5 = 0, \quad X_3 - 2X_4 + X_5 = 0.$$

Elles prouvent que le système des vecteurs U_k est de rang 2, puisqu'il faut 3 équations linéaires indépendantes pour caractériser le sous-espace vectoriel de $\mathbb{R}^5$ qu'ils engendrent.

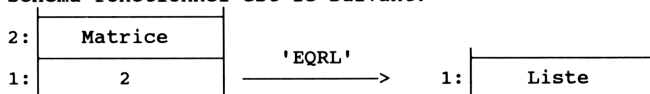
**PROGRAMME 'EQRL':
RECHERCHE DE
RELATIONS LINEAIRES.**

=====

Le problème est le suivant: étant donnés n vecteurs $U_1, U_2, \dots, U_n$, existe-t-il des relations linéaires non triviales entre ces vecteurs, c'est-à-dire des relations du type: $\alpha_1 U_1 + \alpha_2 U_2 + \dots + \alpha_n U_n = 0$, où les scalaires α_k ne sont pas tous égaux à zéro? (si tel est le cas, les vecteurs U_k sont dits linéairement dépendants, ou liés. Sinon ils sont dits linéairement indépendants, ou libres).

Le programme 'EQRL' répond à cette question, en donnant (si les vecteurs sont liés) le plus grand nombre possible de relations indépendantes entre ces vecteurs (Si ces n vecteurs satisfont à k relations linéaires indépendantes, ils forment un système de rang $n-k$).

Le schéma fonctionnel est le suivant:



'Matrice' est une matrice dont les lignes sont les vecteurs U_k .

'Liste' est la liste des différentes relations trouvées:

Cette liste est vide si les vecteurs U_k sont libres.

Si le système des n vecteurs U_k est de rang $n-k$, cette liste est constituée des k relations linéaires indépendantes obtenues entre les vecteurs U_k .

Chacune de ces relations est une expression algébrique où les vecteurs sont appelés $U_1, U_2, \dots$ selon la ligne où ils apparaissent dans la matrice initiale.

Exemple:

On veut trouver s'il existe des relations linéaires entre les vecteurs

$$\begin{array}{ll} U_1 = [1 & 5 \quad 9 \quad 13 \quad 17] & U_2 = [3 \quad 7 \quad 11 \quad 15 \quad 19] \\ U_3 = [2 & 3 \quad 4 \quad 5 \quad 6] & U_4 = [3 \quad 9 \quad 15 \quad 21 \quad 27] \end{array}$$

On place donc au niveau 2 la matrice:

$$A = \begin{bmatrix} [1 & 5 & 9 & 13 & 17] \\ [3 & 7 & 11 & 15 & 19] \\ [2 & 3 & 4 & 5 & 6] \\ [3 & 9 & 15 & 21 & 27] \end{bmatrix}$$

On place au niveau 1 l'entier 2 et on appelle 'EQRL'.

Au bout de 20 secondes on obtient, au niveau 1 de la pile, la liste:

```
{ '-(.75*U1)-.5*U3+.58333333*U4=0'
  '-(1.5*U2)+U3+.83333333*U4=0' }
```

Les deux relations obtenues s'écrivent plus simplement:

$$-9*U_1 - 6*U_3 + 7*U_4 = 0, \quad \text{et} \quad -9*U_2 + 6*U_3 + 5*U_4 = 0.$$

On en déduit que U_1, U_2, U_3, U_4 , sont liés. Plus précisément, ils forment un système de rang 2.

ANALYSE

Sous ce titre un peu vague, et dans le répertoire 'ANLY', on trouvera un certain nombre de programmes traitant de problèmes classiques comme:

- * équation de la tangente en un point donné d'une courbe $Y=F(X)$.
- * recherche de points à tangente horizontale d'une courbe $Y=F(X)$.
- * recherche de points d'inflexion d'une courbe $Y=F(X)$.
- * approximation d'une application au sens des moindres carrés.
- * interpolation par le polynôme de Lagrange.
- * résolution numérique approchée de systèmes non linéaires.
- * sommes partielles de séries de Fourier.

Dans d'autres répertoires, de nombreux programmes utilisent des techniques d'analyse. On ne trouvera ici que ceux qui ne trouvent pas leur place de façon évidente dans un répertoire plus spécialisé.

Plus précisément, le répertoire 'ANLY' contient:

- 'TGTE' : Equation de la tangente, en un point donné , à une courbe d'équation $Y=F(X)$.
- 'EXTRE': Recherche d'extremums pour la fonction $Y=F(X)$.
- 'INFL' : Recherche de points d'inflexion pour la courbe $Y=F(X)$.
- 'APMC' : Approximation, au sens des moindres carrés discrets, et à l'aide de combinaisons linéaires de fonctions libres, d'une fonction f connue par ses valeurs en un certain nombre de points.
- 'LAGR' : Calcul du polynôme interpolateur de Lagrange "passant" par une famille de points donnée.
- 'TRACE': En liaison avec les programmes 'APMC' et 'LAGR', tracé de la famille de points utilisée, puis de la courbe obtenue.
- 'SXY' : Résolution numérique approchée, par la méthode de Newton, d'un système de deux équations à deux inconnues
 $F(X,Y)=0$, $G(X,Y)=0$
- 'SXYZ' : Résolution numérique approchée, par la méthode de Newton, d'un système de 3 équations à 3 inconnues
 $F(X,Y,Z)=0$, $G(X,Y,Z)=0$, $H(X,Y,Z)=0$
- 'RK4' : Résolution approchée, par la méthode de Runge-Kutta d'ordre 4, de l'équation différentielle $Y'=F(X,Y)$.
- 'FOUR' : Calcul des sommes partielles de la série de Fourier d'une fonction périodique donnée.

TANGENTE EN UN POINT D'UNE COURBE D'EQUATION $Y=F(X)$.

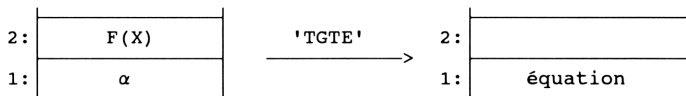
=====

'TGTE' donne l'équation de la tangente au point d'abscisse α de la courbe d'équation $Y=F(X)$, c'est-à-dire la droite d'équation:

$$Y = (X-\alpha)f'(\alpha) + f(\alpha).$$

Cette équation est obtenue sous la forme ' $Y=a*X+b$ '.

Le schéma fonctionnel est:



où F(X) est une expression donnant la fonction F (majuscule X obligatoire)
 et où α est un nombre réel (abscisse du point où on cherche la tangente).

'TGTE': (Checksum: # 8951d, Taille: 131 octets)

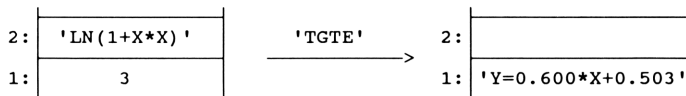
```

«   'X'   STO   DUP   'X'   @   SWAP   EVAL
  →   df   f
  «   'Y'   df   'X'   *   f   X   df   *   -   +   =
    'X'   PURGE
  »
»

```

Exemple:

Tangente à la courbe $y=\text{LN}(1+X^2)$ au point d'abscisse 3.
 On obtient, en mode 3 FIX:



POINTS A TANGENTE HORIZONTALE DE LA COURBE D'EQUATION $Y=F(X)$.

=====

'EXTRE' permet d'obtenir les caractéristiques d'un point à tangente horizontale d'une courbe d'équation $Y=F(X)$.

Le schéma fonctionnel est:



Au niveau 2, F(X) est l'expression de la fonction F (par rapport à la variable X), et a est une valeur numérique permettant de débiter la recherche. La valeur a ne doit pas être abusivement éloignée de la solution recherchée.

Le programme 'EXTRE' recherche (en utilisant l'instruction **ROOT**) une valeur X annulant la dérivée F'.

Si une telle valeur est trouvée, le programme donne:

- * La valeur de X (réel marqué "X" au niveau 3).
- * La valeur de F(X) (réel marqué "F" au niveau 2).
- * La valeur de la dérivée seconde F'' au point X (sous forme d'un réel marqué "F'")
 - Si F''(X) < 0 , on a trouvé un maximum relatif.
 - Si F''(X) > 0 , on a trouvé un minimum relatif.
 - Si F''(X) = 0 (aux erreurs d'arrondi près) , c'est en général un point d'inflexion à tangente horizontale.

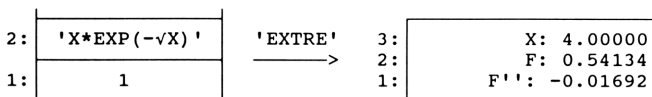
'EXTRE': (Checksum: # 51935d, Taille: 166.5 octets)

```

«  'X'  PURGE  OVER  'X'  Ø  DUP  'X'  Ø
→  f  a  df  ddf
«  df  'X'  a  ROOT  "X"  →TAG
   f  EVAL  "F"  →TAG
   ddf  EVAL  "F'"  →TAG  'X'  PURGE
»
»

```

Exemple: en 12 secondes, et en mode "5 FIX"



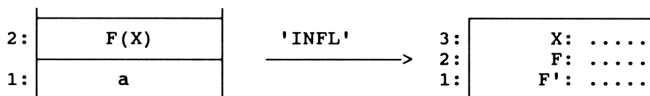
Le point (4,≈0.54134) est donc représentatif d'un maximum.

POINTS D'INFLEXION DE LA COURBE D'EQUATION $Y=F(X)$.

=====

'INFL' permet d'obtenir les caractéristiques d'un point à dérivée seconde nulle (en général un point d'inflexion) d'une courbe d'équation $Y=F(X)$.

Le schéma fonctionnel est:



Au niveau 2, F(X) est l'expression de la fonction F (par rapport à la variable X), et a est une valeur numérique permettant de débiter la recherche. La valeur a ne doit pas être abusivement éloignée de la solution recherchée.

Le programme 'INFL' recherche (en utilisant l'instruction **ROOT**) une valeur X annulant la dérivée seconde F''.

Si une telle valeur est trouvée, le programme donne:

* La valeur de X (réel marqué "X" au niveau 3).

* La valeur de F(X) (réel marqué "F" au niveau 2).

* La valeur de la dérivée F' au point X (sous forme d'un réel marqué "F'"):

Si $F'(X) = 0$, on a trouvé un point d'inflexion.

Si $F'(X) \neq 0$ (aux erreurs d'arrondi près), on n'est pas certain qu'il s'agisse d'un point d'inflexion.

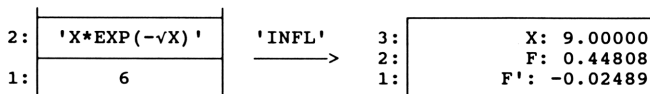
'INFL': (Checksum: # 16137d, Taille: 165.5 octets)

```

" 'X' PURGE OVER 'X' Ø DUP 'X' Ø
→ f a df ddf
" ddf 'X' a ROOT "X" →TAG
  f EVAL "F" →TAG
  df EVAL "F'" →TAG 'X' PURGE
"
"

```

Exemple: en 18 secondes, et en mode "5 FIX"



Le point (9,≈0.44808) est donc un point d'inflexion.

APPROXIMATION AU SENS DES MOINDRES CARRES DISCRETS.

=====

Soit $(X_1, Y_1), (X_2, Y_2), \dots, (X_i, Y_i), \dots, (X_n, Y_n)$ n points du plan (d'abscisses distinctes deux à deux).

Soit une famille de p fonctions $F_1, F_2, \dots, F_i, \dots, F_p$, linéairement indépendantes, avec $p < n$.

Alors il existe une unique application F , combinaison linéaire des F_k , qui réalise la meilleure approximation des points (X_i, Y_i) au sens des moindres carrés discrets, c'est-à-dire qui minimise la somme:

$$\sum_{i=1}^n (Y_i - F(X_i))^2.$$

Pour que le programme 'APMC' (qui détermine la solution F) fonctionne correctement, il faut au préalable:

- * placer dans une variable nommée 'DATA', un **vecteur** formé des points (X_i, Y_i) , écrits sous forme de nombres complexes.
exemple: 'DATA' <— [(-2,3) (-1,0) (0,1) (1,5) (2,3)]
- * placer dans une variable nommée 'FONC' la liste des paramètres permettant d'identifier les fonctions F_k .

Le format de 'FONC' doit être le suivant:

```
{ Fk debut pas nombre }. avec:
"Fk"= expression algébrique définissant la fonction Fk.
"début"= valeur entière minimum de l'entier k.
"pas"= incrément de l'entier k.
"nombre"= nombre de valeurs différentes de k.
```

Exemple:

```
{'X^K' 0 1 5} si on veut rechercher la solution F comme
combinaison linéaire des fonctions 1 ,X ,X^2, X^3, X^4.
```

Remarque: dans l'expression les majuscules 'K' et 'X' sont obligatoires.

La pile n'est pas affectée par le programme 'APMC', qui place la solution F , sous forme d'une expression algébrique, dans la variable 'EQ'. Le résultat est donc prêt à être visualisé au moyen de l'instruction DRAW ou à être évalué par SOLVR.

Répertoire 'ANLY'

programme 'APMC'
(suite)

TEXTE DU PROGRAMME 'APMC' ET EXEMPLE D'UTILISATION.

=====

```
'APMC': ( Checksum: # 15910d, Taille: 348.5 octets )
«   DATA C→R DUP SIZE 1 GET FONC EVAL
    → a b c f m p L
    « 1 L FOR i
        m i 1 - p * + 'K' STO
        1 c FOR j
            a j GET 'X' STO f EVAL
        NEXT
    NEXT
    L c 2 →LIST →ARRY DUP DUP TRN *
    MATR DIVPR ANLY b * 'X' PURGE
    0
    1 L FOR i
        OVER i GET m i 1 - p * +
        'K' STO f EVAL * +
    NEXT
    STEQ DROP 'K' PURGE
»
»
```

N.B: Le programme 'APMC' appelle le programme 'DIVPR' dans le répertoire 'MATR'.

Exemple:

On place le vecteur $[(-3,-1) (-2,1) (-1,2) (1,2) (2,1) (3,0)]$ dans la variable 'DATA'.
On cherche l'application réalisant la meilleure approximation (au sens des moindres carrés) du nuage des points $(-3,-1), \dots, (3,0)$, et qui s'écrit sous la forme:

$$F(X) = a + b \cos(X) + c \cos(2X) + d \cos(3X) + e \cos(4X).$$

On met donc la liste {'COS(K*X)' 0 1 5} dans la variable 'FONC'.

On appelle alors 'APMC'. L'exécution du programme dure 12s .

'APMC' met dans la variable 'EQ' l'expression algébrique:

(en mode 3 FIX):

$$'1.030+0.401 \cos(X)+0.349 \cos(2X)+0.140 \cos(3X)-1.588 \cos(4X)'$$

L'évaluation de 'EQ' donne les résultats suivants: (mode 3 FIX).

X	-3	-2	-1	1	2	3
F(X)	-0.500	1.000	2.000	2.000	1.000	-0.500

(en -2,-1,1,2 les valeurs obtenues sont exactes à 1E-11 près).

Remarque: les familles de fonctions les plus utilisées sont:

$$F_k(X) = \cos(kX), F_k(X) = \sin(kX), F_k(X) = X^K, F_k(X) = \exp(kX).$$

CALCUL DU POLYNOME INTERPOLATEUR DE LAGRANGE.

=====

Soit $(X_1, Y_1), (X_2, Y_2), \dots, (X_i, Y_i), \dots, (X_n, Y_n)$ n points du plan (d'abscisses distinctes deux à deux).

Il existe alors un unique polynôme P de degré $< n$ tel que pour tout indice i , $P(X_i) = Y_i$. On dit que P est le polynôme interpolateur de Lagrange correspondant au nuage des points (X_i, Y_i) .

'LAGR' calcule ce polynôme et le place dans la variable 'EQ', sous forme d'une expression algébrique. On peut alors directement tracer le polynôme par DRAW, ou l'évaluer par SOLVR.

Pour celà, il faut avoir préalablement placé dans la variable 'DATA' le vecteur $[(X_1, Y_1) (X_2, Y_2) \dots (X_n, Y_n)]$ dont les éléments sont les nombres complexes représentant les points (X_i, Y_i) .

'LAGR': (Checksum: # 16533d, Taille: 221.5 octets)

```
«   DATA  C→R  SWAP  DUP  SIZE  1  GET
   → a n
«   1 n FOR i
     a i GET
     0 n 1 - FOR j
       DUP j ^ SWAP
     NEXT
   DROP
   NEXT
   n n 2 →LIST →ARRY MATR DIVPR ANLY
   0
   1 n FOR i
     OVER i GET 'X' i 1 - ^ * +
   NEXT
   STEQ DROP
»
»
```

N.B: Le programme 'LAGR' appelle le programme 'DIVPR' dans le répertoire 'MATR'.

Exemple:

Après avoir mis $[(-3,3) (-2,1) (-1,2) (1,3) (2,-1) (3,2)]$ dans 'DATA' et avoir exécuté 'LAGR', on trouve dans 'EQ' (en 5 s) :

'4 + 1.033333333333*X - 1.666666666667*X^2 - .583333333333*X^3
+ .166666666667*X^4 + .05*X^5'.

Effectivement, ce polynôme vérifie:

$P(-3)=3, P(-2)=.9999999999, P(-1)=2, P(1)=3,$
 $P(2)=-1.00000000001, P(3)=2.$

TRACE DE POINTS ET D'UNE COURBE LES APPROCHANT.

=====

La variable '**DATA**' est supposée contenir un vecteur de nombres complexes (X_i, Y_i). Ces nombres complexes représentent un nuage de points du plan.

La variable '**EQ**' est supposée contenir une expression algébrique (ou un programme) traçable à l'écran par **DRAW**.

Le programme '**TRACE**' trace alors successivement:

- * les points de la variable '**DATA**'.
- * l'expression (ou le programme) '**EQ**'.

N.B.: Pour être facilement repérés, les points de la variable '**DATA**' sont en fait représentés par des petits carrés
Les dimensions de **PICT** et les intervalles de tracé en **X** et **Y** ne sont pas modifiés.

Le programme '**TRACE**' est particulièrement destiné à être appelé après les programmes '**LAGR**' (recherche du polynôme interpolateur de Lagrange) et '**APMC**' (approximation au sens des moindres carrés discrets).

On peut ainsi visualiser en même temps le nuage des points (X_i, Y_i) et la courbe qui en réalise une approximation.

A la fin du tracé, on passe dans l'environnement **GRAPH**.

On quitte le programme par un appui sur '**ON**'.

'**TRACE**': (Checksum: # 60176d, Taille: 203 octets)

```

« { # 1d # 1d }   PX->C   { # 0d # 0d }   PX->C   -
  → d
«   ERASE { # 0d # 0d }   PVIEW
    DATA OBJ-> 1 GET
    1 SWAP START
      DUP d - SWAP d + BOX
    NEXT
  FUNCTION 'X' INDEP DRAW GRAPH
»

```

RESOLUTION ITERATIVE D'UN SYSTEME A DEUX INCONNUES.

=====

'SXY' permet de résoudre, de manière approchée, par la méthode de Newton, un système de deux équations aux deux inconnues X,Y:

$$F(X,Y)=0, G(X,Y)=0$$

Le principe est le suivant:

On définit une suite (Xn,Yn) par la donnée d'un point initial (X0,Y0) et par la relation:

$$\begin{bmatrix} X(n+1) \\ Y(n+1) \end{bmatrix} = \begin{bmatrix} X(n) \\ Y(n) \end{bmatrix} - \left[J(X(n), Y(n)) \right]^{-1} \begin{bmatrix} F(X(n), Y(n)) \\ G(X(n), Y(n)) \end{bmatrix}$$

Où J est la matrice Jacobienne

$$J(X,Y) = \begin{bmatrix} F'_x(X,Y) & F'_y(X,Y) \\ G'_x(X,Y) & G'_y(X,Y) \end{bmatrix}$$

Sous certaines conditions, la suite (Xn,Yn) converge vers une solution du système.

Au début, la pile doit avoir l'aspect suivant:

F(X,Y) et G(X,Y) sont les expressions algébriques des applications F et G (dans ces expressions l'usage des majuscules X et Y est obligatoire).	2:	F(X,Y)
	1:	G(X,Y)

On appelle ensuite le programme 'SXY'. Au bout d'un certain temps (nécessaire au calcul des dérivées partielles), le programme 'SXY' est suspendu, avec affichage d'un menu présentant les deux entrées **NEW** (à gauche) et **EXIT** (à droite).

On place alors le point de départ [X0, Y0] des itérations, sous forme d'un vecteur, au niveau 1 de la pile.

On appuie sur la touche **NEW** pour voir apparaître [X1, Y1]. Un nouvel appui donne [X2, Y2], etc...

On sort du programme par un appui sur la touche **EXIT**.

A tout moment où le programme est suspendu, on peut avant de faire **NEW**, placer un nouveau point [X0, Y0] au niveau 1 de la pile (à la place du point [Xn,Yn] qu'on vient d'obtenir), afin de lancer une nouvelle recherche.

TEXTE DU PROGRAMME 'SXY' ET EXEMPLE D'UTILISATION.

=====

'SXY': (Checksum: # 56453d, Taille: 357.5 octets)

```

"  → f g
    " { X Y } PURGE
      f 'X'  ∂ f 'Y'  ∂ g 'X'  ∂ g 'Y'  ∂
      → dfx dfy dgx dgy
    " {
      { "NEW"
        " DUP DUP V→ 'Y' STO 'X' STO
          dfx EVAL dfy EVAL dgx EVAL dgy EVAL
          { 2 2 } →ARRY INV f EVAL g EVAL
            2 →ARRY * -
        "
      }
      { } { } { } { }
      { "EXIT" " CONT " }
    }
    TMENU HALT { X Y } PURGE 2 MENU
  "
"
```

Exemple:

On veut résoudre le système $X^2+Y^2=1$, $2X+Y=1$.

On crée donc la pile ci-dessous. On appelle ensuite 'SXY'.

Le menu contenant les entrées "NEW" et "EXIT" est affiché au bout de trois secondes. On place alors le point de départ, par ex:

2:	'X*X+Y*Y-1'	[1 0], au niveau 1, et on fait "NEW".
1:	'2*X+Y-1'	En une à deux secondes le point [1,-1] est placé sur la pile.

Chaque NEW amène un nouveau point (le calcul de chaque point durant 1 seconde). On trouve ainsi successivement en mode 6 FIX:

[0.833333 -0.666667], [0.801282 -0.602564],
et [0.800002 -0.600004], etc...

La suite (Xn,Yn) converge vers [0.8, -0.6], qui est bien une solution du problème.

On sort du programme par un appui sur "EXIT".

RESOLUTION ITERATIVE D'UN SYSTEME A TROIS INCONNUES.

=====

'SXYZ' permet de résoudre, de manière approchée, par la méthode de Newton, un système de 3 équations aux 3 inconnues X,Y,Z :

$$\begin{cases} F(X,Y,Z)=0 \\ G(X,Y,Z)=0 \\ H(X,Y,Z)=0 \end{cases}$$

Le principe est le suivant:

On définit une suite (Xn,Yn,Zn) par la donnée d'un point initial (X0,Y0,Z0) et par la relation:

$$\begin{bmatrix} X(n+1) \\ Y(n+1) \\ Z(n+1) \end{bmatrix} = \begin{bmatrix} X(n) \\ Y(n) \\ Z(n) \end{bmatrix} - \left[J(X(n), Y(n), Z(n)) \right]^{-1} \begin{bmatrix} F(X(n), Y(n), Z(n)) \\ G(X(n), Y(n), Z(n)) \\ H(X(n), Y(n), Z(n)) \end{bmatrix}$$

Où J est la matrice

$$J(X,Y,Z) = \begin{bmatrix} F'_x(X,Y,Z) & F'_y(X,Y,Z) & F'_z(X,Y,Z) \\ G'_x(X,Y,Z) & G'_y(X,Y,Z) & G'_z(X,Y,Z) \\ H'_x(X,Y,Z) & H'_y(X,Y,Z) & H'_z(X,Y,Z) \end{bmatrix}$$

Sous certaines conditions, la suite (Xn,Yn,Zn) converge vers une solution du système.

Au début, la pile doit avoir l'aspect:
"F(X,Y,Z)", "G(X,Y,Z)", "H(X,Y,Z)" sont
les expressions des applications F,G,H.
(Majuscules X ,Y, et Z obligatoires).

3:	F(X,Y,Z)
2:	G(X,Y,Z)
1:	H(X,Y,Z)

On appelle ensuite le programme 'SXYZ'. Au bout d'un certain temps (nécessaire au calcul des dérivées partielles), le programme 'SXYZ' est suspendu, avec affichage d'un menu présentant les deux entrées **NEW** (à gauche) et **EXIT** (à droite).

On place alors le point de départ [X0, Y0, Z0] des itérations, sous forme d'un vecteur, au niveau 1 de la pile.

On appuie sur la touche **NEW** pour voir apparaître [X1, Y1, Z1].
Un nouvel appui donne [X2, Y2, Z2], etc...

On sort du programme par un appui sur la touche **EXIT**.

A tout moment où le programme est suspendu, on peut avant de faire **NEW**, placer un nouveau point [X0, Y0, Z0] au niveau 1 de la pile (à la place du point [Xn, Yn, Zn] qu'on vient d'obtenir), afin de lancer une nouvelle recherche.

TEXTE DU PROGRAMME 'SXYZ' ET EXEMPLE D'UTILISATION.

=====

'SXYZ': (Checksum: #62647 d, Taille: 551 octets)

```

"  → f   g   h
   " { X Y Z } PURGE f 'X' ð f 'Y' ð f 'Z' ð
     g 'X' ð g 'Y' ð g 'Z' ð
     h 'X' ð h 'Y' ð h 'Z' ð
     → dfx dfy dfz dgx dgy dgz dhx dhy dhz
   " {
     { "NEW"
       " DUP      DUP      V→
         'Z'      STO      'Y'      STO      'X'      STO
         dfx      EVAL      dfy      EVAL      dfz      EVAL
         dgx      EVAL      dgy      EVAL      dgz      EVAL
         dhx      EVAL      dhy      EVAL      dhz      EVAL
         { 3 3 } →ARRY      INV      f      EVAL      g      EVAL
         h      EVAL      3      →ARRY      *      -
       "
     }
     { } { } { } { }
     { "EXIT" " CONT " }
   }
  TMENU  HALT   { X Y Z }   PURGE   2   MENU
"
"
"

```

Exemple: soit le système : $X^2+Y^2+Z^2=6$, $X+3X+2Z=5$, $X^3+YZ=-1$.

On crée la pile ci-dessous, et on appelle 'SXYZ': au bout de 7 s, <table border="1" style="border-collapse: collapse; width: 150px;"> <tr> <td style="padding: 2px;">3:</td> <td style="padding: 2px;">'X*X+Y*Y+Z*Z-6'</td> </tr> <tr> <td style="padding: 2px;">2:</td> <td style="padding: 2px;">'X+3*Y+2*Z-5'</td> </tr> <tr> <td style="padding: 2px;">1:</td> <td style="padding: 2px;">'X^3+Y*Z+1'</td> </tr> </table>	3:	'X*X+Y*Y+Z*Z-6'	2:	'X+3*Y+2*Z-5'	1:	'X^3+Y*Z+1'	Le menu avec les entrées "NEW" et "EXIT" est affiché. On place le point de départ, ex: [3 2 0], au niveau 1. Chaque appui sur NEW amène un nouveau point: en mode 3 FIX: [2.041 1.689 -1.053], et [1.427 1.846 -.0982], [1.111 1.967 -1.006] etc... On obtient bientôt: [1 2 -1].
3:	'X*X+Y*Y+Z*Z-6'						
2:	'X+3*Y+2*Z-5'						
1:	'X^3+Y*Z+1'						

La suite (X_n, Y_n, Z_n) converge vers le point $(1, 2, -1)$, qui est effectivement solution du problème.

Un appui sur "EXIT" permet de sortie du programme.

RESOLUTION D'UNE EQUATION DIFFERENTIELLE $Y' = F(X, Y)$.

=====

'RK4' permet de résoudre de manière approchée une équation différentielle $Y' = F(X, Y)$ (équation d'ordre 1, résolue en Y').

La méthode utilisée est "Runge-Kutta d'ordre 4".

Le principe en est le suivant:

On veut connaître les valeurs de la solution f du problème de Cauchy:

$f'(x) = F(x, f(x))$, $f(x_0) = y_0$, où (x_0, y_0) est un point donné.

(une telle solution existe de manière unique, sous certaines hypothèses de régularité de F).

On définit la suite $X_n = X_0 + n \cdot h$, où h est le "pas" de la méthode, et la suite Y_n , définie par la donnée du terme initial Y_0 , et par le système:

$a = h \cdot F(X(n-1), Y(n-1))$.

$b = h \cdot F(X(n-1) + h/2, Y(n-1) + a/2)$.

$c = h \cdot F(X(n-1) + h/2, Y(n-1) + b/2)$.

$d = h \cdot F(X(n-1) + h, Y(n-1) + c)$.

et $Y(n) = Y(n-1) + (a + 2 \cdot b + 2 \cdot c + d) / 6$.

Les $Y(n)$ sont alors des valeurs approchées de $F(X(n)) = F(X_0 + n \cdot h)$.

Avant d'appeler 'RK4', on doit placer au niveau 1 l'expression $F(X, Y)$ (majuscules X, Y obligatoires).

On lance 'RK4'. Au bout d'un certain temps (nécessaire au calcul des dérivées partielles), le programme 'RK4' est suspendu et le menu suivant est affiché:

NEW		→N	→PAS		EXIT
-----	--	----	------	--	------

Au même moment les réels marqués "PAS: 0.05" et "N: 1" sont placés aux niveaux 2 et 1, pour indiquer que par défaut, le pas de la méthode (le nombre h) est égal à 0.05, et que l'on va obtenir un point à chaque fois.

On peut modifier le pas en plaçant le nouveau pas au niveau 1 et en appuyant sur la touche "→PAS".

On peut demander d'obtenir n points à chaque fois en plaçant l'entier n au niveau 1 et en appuyant sur la touche "→N".

Chaque appui sur la touche "NEW" provoque le calcul et l'affichage de n points ($n=1$ par défaut).

Les points sont obtenus sous forme de vecteurs $[X, Y]$.

Un appui sur la touche "EXIT" permet de sortir du programme.

**TEXTE DU PROGRAMME 'RK4'
ET EXEMPLE D'UTILISATION.**

=====

```
'RK4': ( Checksum: # 19251d, Taille: 740 octets )
« [ 0 0 ] .05 1 0 0 0 0
  → f v h n k1 k2 k3 k4
  « .05 "PAS" →TAG 1 "N" →TAG
    {
      { "NEW"
        « DUP DUP 'v' STO V→ 'Y' STO 'X' STO
          1 n START
            f →NUM h * 'k1' STO 'X' h 2 / STO+
            'Y=v(2)+k1/2' DEFINE f →NUM h * 'k2' STO
            'Y=v(2)+k2/2' DEFINE f →NUM h * 'k3' STO
            'X' h 2 / STO+
            'Y=v(2)+k3' DEFINE f →NUM h * 'k4' STO
            X 'v(2)+(k1+2*k2+2*k3+k4)/6' →NUM
            2 →ARRY DUP 'v' STO
          NEXT
        »
      } { } { "→N" « 'n' STO » }
      { "→PAS" « 'h' STO » } { } { "EXIT" « CONT » }
    }
  TMENU HALT { X Y } PURGE 2 MENU
»
»
```

Exemple:

On cherche la solution f de l'équation $Y'=2*X*(1+Y^2)$ qui s'annule à l'origine. Cette solution est donnée par $f(X)=\text{TAN}(X^2)$.

On place ' $2*X*(1+Y^2)$ ' au niveau 1 et on appelle 'RK4'.

Le menu des entrées "NEW", "→N", "→PAS" et "EXIT" apparaît, de même que les réels marqués "PAS: 0.05" et "N: 1" aux niveaux 2 et 1.

On place le point de départ, [0 0], au niveau 1 de la pile, et on appuie sur "NEW".

On obtient donc le point suivant (en mode 3 FIX: [0.050 0.003]). Pour aller plus vite, on place 10 au niveau 1 et on appuie sur la touche "→N".

Un nouvel appui sur "NEW" donne alors les 10 points suivants:

Par exemple, en mode STD:

```
[.2 4.00235117567E-2], la valeur exacte étant
                                tan(.2^2) ≈ 4.00213469955E-2.
[.5 .255700574089], la valeur exacte étant:
                                tan(.5^2) ≈ .255341921221.
```

Un appui sur "EXIT" permet de quitter le programme.

SOMMES PARTIELLES D'UNE SERIE DE FOURIER.

=====

Soit f une fonction périodique de période $T > 0$. On suppose connue l'expression de f sur un intervalle $[a, b]$, avec $b = a + T$.

La série de Fourier de f s'écrit, sous réserve de convergence:

$$\sum_{k=0}^{+\infty} (a_k \cos(k \cdot w \cdot x) + b_k \sin(k \cdot w \cdot x)) \quad \text{où } w = \frac{2\pi}{T}.$$

avec, pour tout $k \geq 0$,

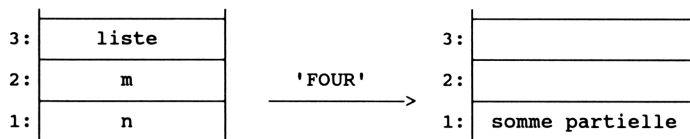
$$a_k = \frac{2}{T} \int_a^b f(x) \cos(k \cdot w \cdot x) dx \quad \text{et} \quad b_k = \frac{2}{T} \int_a^b f(x) \sin(k \cdot w \cdot x) dx$$

$$(\text{Exception: } a_0 = \frac{1}{T} \int_a^b f(x) dx)$$

'FOUR' permet de calculer toute somme partielle de cette série, c'est-à-dire toute expression:

$$\sum_{k=m}^n (a_k \cos(k \cdot w \cdot x) + b_k \sin(k \cdot w \cdot x))$$

Le schéma fonctionnel est le suivant:



où "liste" est une liste ayant le format suivant { F a b } avec:

F = expression de la fonction (majuscule X obligatoire)

a et b = extrémités de l'intervalle (la période est considérée égale à $T = b - a$).

m et n = respectivement indice minimum et indice maximum k pour lesquels on calcule a_k et b_k .

Les calculs d'intégrales sont faits avec la précision indiquée par le mode d'affichage (en mode **n FIX**, les intégrales sont calculées avec une précision de n décimales)

Le résultat ("somme partielle") est obtenu sous la forme d'une expression algébrique.

Dans cette expression, w et x sont désignées par les variables symboliques 'X' et 'W'. Un appui sur EVAL permet alors de remplacer 'W' par sa valeur, c'est-à-dire:

$$W = \frac{2\pi}{T} \quad (W = \text{"pulsation"}).$$

TEXTE DU PROGRAMME 'FOUR'.

=====

'FOUR': (Checksum: # 37251d, Taille: 380.5 octets)

```

"  → m n
"  " 'X' PURGE EVAL DUP2 - NEG
    → f a b t
"  RAD -2 SF 2 π * t / 'W' STO
    " a b ROT 'X' ∫ →NUM DUP
      ABS IERR > * t /
"
    → int
    " 0 m n FOR k
      IF k THEN
        k 'W' * 'X' *
        → h
        " h EVAL COS f * int EVAL 2 *
          h COS * + h EVAL SIN f *
            int EVAL 2 * h SIN * +
        "
      ELSE f int EVAL + END
    NEXT
  "
"
"
"

```

Remarque:

Au cours du calcul, si le programme 'FOUR' trouve une valeur d'intégrale inférieure en valeur absolue à l'erreur absolue donnée par la calculatrice pour cette intégrale, alors le programme 'FOUR' considère que cette intégrale est nulle, et la quantité correspondante n'apparaît pas dans l'expression de la somme partielle de la série de Fourier.

Cela se produit en particulier pour les fonctions paires (les "bk" sont nuls) ou impaires (les "ak" sont nuls).

Dans de tels cas, l'expression de la somme partielle de la série de Fourier est débarrassée de ses termes négligeables.

PROGRAMME 'FOUR': EXEMPLES D'UTILISATION.

=====

Exemple1:

Développer en série de Fourier la fonction f , périodique de période 1, égale à X^2 sur le segment $[0, 1]$. On veut par exemple calculer les coefficients a_k et b_k pour $0 \leq k \leq 3$. Les calculs d'intégrales seront faits en mode 3 FIX. On obtient en 33 secondes:

3:	{ 'X*X' 0 1 }		3:	
2:	0	'FOUR' →	2:	
1:	3		1:	expression

avec expression =

'0.333+0.101*COS(W*X)-0.318*SIN(W*X)+0.025*COS(2*W*X)
-0.159*SIN(2*W*X)+0.011*COS(3*W*X)-0.106*SIN(3*W*X)'

Un appui sur EVAL permet alors de remplacer 'W' par sa valeur, c'est-à-dire $2\pi/T = 2\pi \approx 6.283$.

Sur cet exemple, on peut s'intéresser uniquement aux coefficients a_5 et b_5 . il suffit de placer 5 et 5 aux niveaux 2 et 1.

On obtient alors, en 16 secondes, au niveau 1 de la pile, l'expression: '0.004*COS(5*W*X)-0.064*SIN(5*W*X)'.
'

Exemple2:

On s'intéresse à la même fonction $X \rightarrow X^2$, mais on veut la développer sur $[0,1]$ en série de cosinus. Il suffit de la considérer comme paire et définie sur $[-1,1]$ (donc périodique de période 2).

En mode 3 FIX, et en 26 secondes:

3:	{ 'X*X' -1 1 }		3:	
2:	0	'FOUR' →	2:	
1:	3		1:	expression

Avec expression =

'0.333-0.405*COS(W*X)+0.101*COS(2*W*X)-0.045**COS(3*W*X)'

EVAL permet alors de remplacer 'W' par sa valeur ($2\pi/T \approx 3.142$).

Exemple3:

On peut de même développer X^2 sur $[0,1]$ en série de sinus. Il suffit de prolonger X^2 sur $[-1,0]$ de façon impaire et de considérer que la période est 2. On place { 'X*X*SIGN(X)' -1 1 } au niveau 3 (les autres niveaux restant les mêmes que dans l'exemple 2). On obtient (en mode 3 FIX):

'0.379*SIN(W*X) - 0.318*SIN(2*W*X) + 0.203**SIN(3*W*X)'

EVAL permet alors de remplacer 'W' par sa valeur ($2\pi/T = \pi$).

DEVELOPPEMENTS LIMITES

La HP48 permet de calculer (sous réserve de dérivabilité suffisante) le développement limité à un ordre quelconque n, en 0, d'une fonction f, à l'aide de la formule de Taylor:

$$f(x) = f(0) + \frac{f'(0)}{1!} x + \frac{f''(0)}{2!} x^2 + \dots + \frac{f^{(n)}(0)}{n!} x^n + o(x^n).$$

Pour aboutir à un tel résultat, il faut utiliser l'instruction TAYLR du menu ALGBRA. L'inconvénient de cette méthode tient au fait que la HP48 calcule les dérivées successives de f, sous forme symbolique, avant de les évaluer en 0. Le calcul s'avère parfois très long, et nécessite une place en mémoire importante.

Pour prendre un exemple, le développement limité, à l'origine et à l'ordre 5, de la fonction f définie par $f(x) = \exp(\sin(x))$ (ce n'est pourtant pas un exemple bien compliqué!) est obtenu en 1 mn 30s et on trouve: '1+X+.5*X^2-.125*X^4-6.66666666667E-2*X^5'.

Avec les mêmes données, les programmes du répertoire 'DL' permettent d'obtenir ce résultat en 11 secondes.

On peut multiplier les exemples, de manière plus convaincante.

Le calcul du développement de ATAN(ASIN(X)) (arctangente de arcsinus de x), en 0 à l'ordre 5, et en utilisant l'instruction TAYLR du menu ALGBRA, est interrompu au bout de 2 minutes, sur une erreur "Insufficient Memory" (sur une machine possédant encore 4500 octets de mémoire libre). Sur la même machine, le même développement est obtenu en 11 secondes avec les programmes du répertoire 'DL'.... et on trouve: 'X-.166666666666*X^3+.108333333333*X^5'

Le gain de temps est très important, de même que l'espace mémoire indispensable aux calculs se trouve considérablement réduit.

L'idée qui préside aux programmes du répertoire 'DL' est qu'un développement limité comme $f(x) = a + bx + cx^2 + \dots + d x^m + o(x^m)$ peut être représenté par le vecteur [a b c d].

C'est sous cette forme (écriture suivant les puissances croissantes de x) que seront utilisés et obtenus les développements limités dans le répertoire 'DL'.

Pour la commodité de la lecture de ceux-ci, le programme 'VISU' transforme un tel vecteur en l'expression algébrique correspondante.

Il a fallu implanter les principales opérations sur les développements limités (somme, produit, puissance, quotient, composé) de même qu'il a fallu programmer le calcul du développement limité de chaque fonction usuelle. On trouvera donc un programme spécifique pour chacune de ces fonctions (il y en a 16 ici). Le résultat est un ensemble de 27 programmes.

Il était possible de réduire le nombre de ces programmes en stockant dans une matrice les développements usuels, jusqu'à l'ordre 7 par exemple (on aurait ainsi obtenu un tableau de $7*16=112$ réels, occupant en mémoire $112*6=672$ octets. l'inconvénient était double: l'ordre des développements limités devait être inférieur ou égal à 7 (pour reprendre cet exemple) et il devenait impossible de tirer partie des propriétés de telle ou telle fonction.

Attention: Un certain nombre de programmes du répertoire 'DL' font appel à des programmes du répertoire 'POLY'.
Le répertoire 'DL' contient donc les programmes suivants:

Opérations sur les développements limités:

'DIM' : Ordre et valuation d'un développement limité donné.
('DIM' est très souvent utilisé)
'EDL' : somme de deux développements limités.
'*DL' : produit de deux développements limités.
'CPDL' : composé de deux développements limités.
'INVDL' : inverse d'un développement limité.
'QDL' : quotient de deux développements limités.
'DLN' : puissance entière d'un développement limité.

Opérations usuelles sur les développements limités:

'VISU' : Pour passer de la forme "vecteur" d'un développement limité à sa forme algébrique traditionnelle.
'DERDL' : dérivation d'un développement limité.
'INTDL' : intégration d'un développement limité.
'X→XN' : remplacer X par Xⁿ dans un développement limité.
'X→aX' : remplacer X par a*X dans un développement limité.

Développements limités usuels et composition par les développements limités usuels:

'EX', 'AX', 'LG', 'XA';
'SN', 'CS', 'TG', 'SH', 'CH', 'TH';
'ASNS', 'ACS', 'ATG', 'ASH', 'ATH';

Ces programmes permettent de calculer le développement limité de chacune des fonction f suivantes, et de calculer le composé par une telle fonction f d'un développement limité donné. Les fonctions f implantées ici sont, en reprenant l'ordre ci-dessus:

X → EXP(X), X → AⁿX, X → LN(X), X → X^a;
X → SIN(X), X → COS(X), X → TAN(X);
X → SINH(X), X → COSH(X), X → TANH(X);
X → ASIN(X), X → ACOS(X), X → ATAN(X);
X → ASINH(X), X → ATANH(X);

N.B:

Les programmes du répertoire 'DL' admettent des développements limités comme arguments et ils en renvoient comme résultat.

En tant qu'argument, un développement limité devra être donné sous la forme d'un vecteur. Les programmes du répertoire 'DL' interpréteront cette longueur comme donnant l'ordre du développement limité. Par exemple, si vous mettez [1 3 -4] sur la pile, cela signifie que vous y mettez le développement limité 1 + 3*X -4*X² + o(X²). Si vous voulez calculer le développement limité de SIN(1 + 3*X -4*X²) à l'ordre 6, il vous faudra mettre [1 3 -4 0 0 0] sur la pile (qui représente le développement limité 1 + 3*X -4*X² + o(X⁶)).

Réciproquement, lorsque un programme du répertoire 'DL' donne un développement limité comme résultat, il le calcule avec l'ordre le plus grand possible (compte tenu des données, bien sûr), et tous les coefficients obtenus sont exacts (aux erreurs d'arrondi près, qui sont très souvent négligeables).

Tous les D.L. qui seront considérés ici sont des D.L. à l'origine (c'est en tout cas obligatoire pour le D.L. de g lorsqu'il s'agira de déterminer le D.L. de gof).

ORDRE ET VALUATION D'UN DEVELOPPEMENT LIMITE.

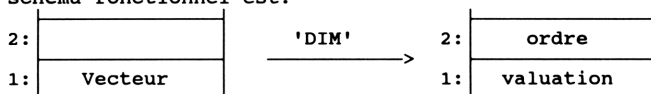
=====

$Sif(x) = A_0 + A_1X + A_2X^2 + \dots + A_nX^n + o(X^n)$ est un développement limité, son ordre est n et sa valuation est p (si p est le plus petit indice tel que $A_p \neq 0$). La valuation est nulle si le terme constant est non nul.

Ces deux indices sont d'une grande importance dans les opérations sur les développements limités (produit, quotient, composé). Ils permettent en effet de calculer l'ordre du développement résultant de telles opérations.

'DIM' permet de calculer ces deux indices, à partir de la forme "vecteur" d'un développement limité.

Le schéma fonctionnel est:



L'ordre n est ici égal à la dimension du vecteur (donnée par SIZE) diminuée de 1. En effet un développement limité d'ordre n est représenté par un vecteur de $n+1$ coefficients.

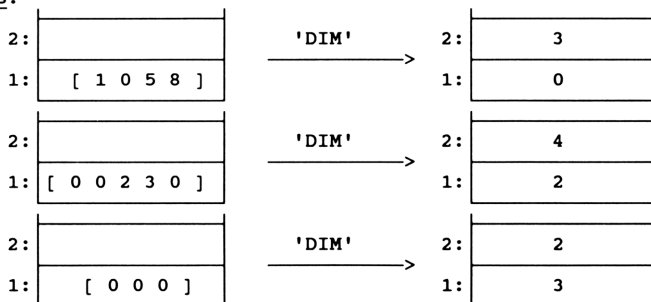
La valuation p est un entier tel que $0 \leq p \leq n$. Exception: dans le cas où le vecteur au niveau 1 ne contient que des coefficients nuls, on obtient $p=n+1$.

Remarque: en principe l'utilisateur n'a pas à appeler lui même le programme 'DIM'. Par contre, celui-ci est appelé par la plupart des programmes du répertoire 'DL'.

'DIM': (Checksum: # 20600d, Taille: 100 octets)

```
«   DUP   SIZE   1   GET   1   -   SWAP
    IF   DUP   RNRM
      THEN
        1   DO   GETI   UNTIL   ABS   END
        2   -   0   MAX   SWAP   DROP
      ELSE
        DROP   DUP   1   +
      END
    »
```

Exemples:



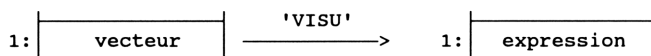
ECRITURE ALGEBRIQUE D'UN DEVELOPPEMENT LIMITE.

=====

'VISU' transforme un développement limité écrit sous forme de vecteur en l'expression algébrique qui lui correspond. la variable utilisée est X.

Cette expression contient, à la fin, un terme de la forme $o(X^n)$ qui indique l'ordre du développement limité. cela permet de distinguer les développements limités obtenus à partir, par exemple de [1 -2 5] et de [1 -2 5 0 0], qui sans cela donneraient la même expression.

Le schéma fonctionnel est le suivant:



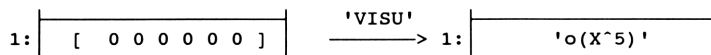
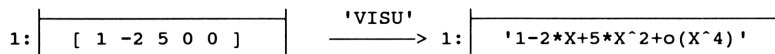
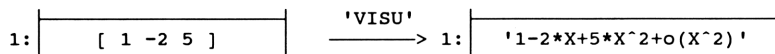
'VISU': (Checksum: # 31065d, Taille: 157.5 octets)

```

«  OBJ→  1  GET  1  -
   →  n
   «    0
      0  n  FOR  i
        n  i  -  2  +  ROLL  'X'  i  ^  *  +
      NEXT
      'o(X^Y)'  { Y }  n  +  ↓MATCH  DROP  +
   »
»

```

Exemples: (temps de calcul = 3 secondes)



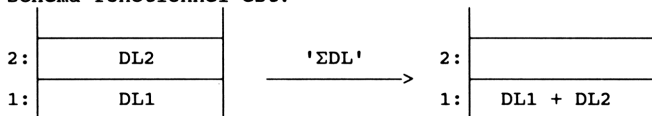
SOMME DE DEUX DEVELOPPEMENTS LIMITES.

=====

'ΣDL' effectue la somme de deux développements limités DL1 et DL2, tous deux exprimés sous la forme "vecteur". Le résultat est un vecteur.

Si les deux développements ont le même ordre, il ne s'agit ici que de la somme classique de deux vecteurs. Sinon, la somme s'effectue après avoir tronqué celui dont l'ordre est le plus élevé. Si DL1 et DL2 sont d'ordre n et p, le résultat obtenu représente donc un développement limité d'ordre $\min(n,p)$.

Le schéma fonctionnel est:



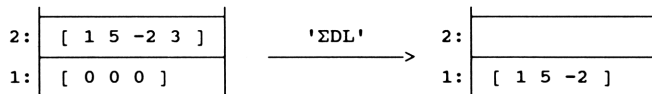
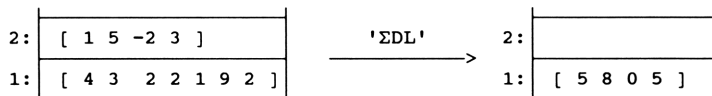
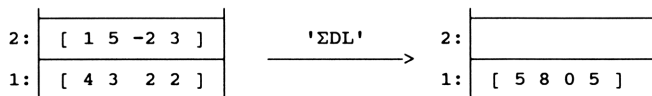
'ΣDL': (Checksum: # 26598d, Taille: 62.5 octets)

```

«   DUP2
    SIZE 1 GET SWAP SIZE 1 GET
    MIN 1 →LIST SWAP
    OVER RDM 3 ROLLD RDM +
»

```

Exemples: (en moins d'une seconde)



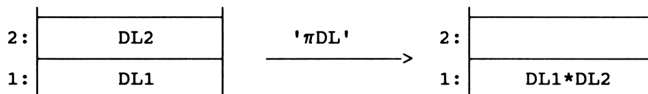
PRODUIT DE DEUX DEVELOPPEMENTS LIMITES.

=====

'πDL' effectue le produit de deux développements limités DL1 et DL2, tous deux exprimés sous la forme "vecteur". Le résultat est un vecteur.

L'ordre du développement limité obtenu est fonction de l'ordre et de la valuation de chacun des deux développements DL1 et DL2. En tout cas, 'πDL' donne le résultat à l'ordre le plus élevé possible (tous les coefficients obtenus sont exacts).

Le schéma fonctionnel est:

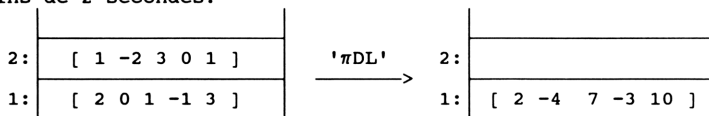


N.B: le programme 'πDL' appelle 'DIM' et le programme 'PRODP' du répertoire 'POLY'.

```
'πDL':( Checksum: # 24717d, Taille: 94.5 octets )
  «  DUP2  DIM  ROT  DIM  4  ROLL  +  3  ROLLD
    +  MIN  1  +  1  →LIST  3  ROLLD
      POLY  PRODP  DL  SWAP  RDM
  »
```

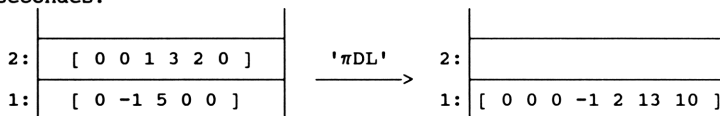
Exemples:

* en moins de 2 secondes:



Car $(1 - 2X + 3X^2 + X^4 + o(X^4)) * (2 + X^2 - X^3 + 3X^4 + o(X^4))$
 $= 2 - 4X + 7X^2 - 3X^3 + 10X^4 + o(X^4)$.

* en 2 secondes:



Car $(X^2 + 3X^3 + 2X^4 + o(X^5)) * (-X + 5X^2 + o(X^4))$
 $= -X^3 + 2X^4 + 13X^5 + 10X^6 + o(X^6)$

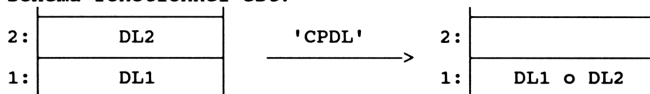
COMPOSE DE DEUX DEVELOPPEMENTS LIMITES.

=====

'CPDL' effectue la composition de deux développements limités DL1 et DL2, tous deux exprimés sous la forme "vecteur". Le résultat est un vecteur.

L'ordre du développement limité obtenu est fonction de l'ordre et de la valuation de chacun des deux développements DL1 et DL2 (plus précisément: la valuation de DL1 après en avoir annulé le terme constant). En tout cas, 'CPDL' donne le résultat à l'ordre le plus élevé possible (tous les coefficients obtenus sont exacts).

Le schéma fonctionnel est:



N.B: le programme 'CPDL' appelle 'DIM' et le programme 'PRODP' du répertoire POLY'.

Il est indispensable que le terme constant de DL2 soit nul (DL2 doit représenter un infiniment petit). Dans le cas contraire, Il y a un message d'erreur.

'CPDL':(Checksum: # 20203d, Taille: 326 octets)

```

" IF OVER 1 GET ABS NOT THEN
  DUP 1 0 PUT SWAP {1} RDM
  3 PICK DIM 4 PICK DIM
  → sf vf sg vg
  " IF sg THEN
    'vf*(vg-1)+sf+1' EVAL 'vf*(sg+1)' EVAL
    MIN 1 →LIST
    → d
    " d RDM
      POLY 1 1 →ARRY
      2 sg 1 + FOR i
      4 PICK PRODP d RDM
      DUP 4 PICK i GET *
      ROT + SWAP
    NEXT
    DROP DL
  "
END 3 ROLL2 DROP2
"
END

```

Exemple: en 4 secondes,



Car si $f(X) = -X^2 + 2X^3 + X^4 + o(X^4)$

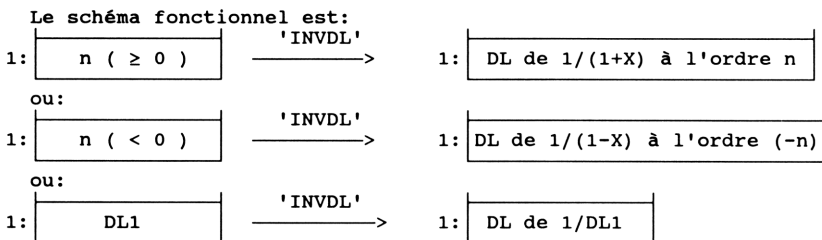
et si $g(X) = 1 - 2X^2 + X^3 + o(X^3)$, alors:

$gof(X) = 1 - 2X^4 + 8X^5 - 5X^6 + o(X^6)$.

INVERSE D'UN DEVELOPPEMENT LIMITE ET DEVELOPPEMENT DE $1/(1+X)$.

=====

'INVDL' permet tout à la fois d'obtenir le développement limité de $1/(1+X)$ ou de $1/(1-X)$ à un ordre quelconque n , et de calculer l'inverse d'un développement limité DL1 (le résultat est alors un développement limité. Dans ce cas il est indispensable que le terme constant de DL1 soit non nul, sans quoi une erreur "division par zéro" se produit).



'INVDL':(Checksum: # 18734d, Taille: 192.5 octets)

```

"  "  →  n  s
    "  0  n  FOR i  s  i  ^  NEXT n 1  +  →ARRY
    "
    SWAP  DUP
    IF  DUP  TYPE  THEN
      1  GET  DUP  INV  3  ROLL  /  1  0  PUT
      DUP  DIM  /  FLOOR  -1  5  ROLL  EVAL  CPDL  *
    ELSE
      ABS  SWAP  SIGN  NEG  ROT  EVAL
    END
  
```

Exemple:

* DL de $1/(1+X)$ à l'ordre 5: (en moins d'une seconde)

```

1: [ 5 ] --'INVDL'--> 1: [ 1 -1 1 -1 1 -1 ]
  
```

* DL de $1/(1-X)$ à l'ordre 7: (en moins d'une seconde)

```

1: [ -7 ] --> 1: [ 1 1 1 1 1 1 1 ]
  
```

* DL de $1/(1 - X^2 + X^3 - 2X^4 + o(X^5))$: en 3 secondes,

```

1: [ 1 0 -1 1 -2 0 ] --> 1: [ 1 0 1 -1 3 -2 ]
  
```

en effet ici:

$$1/(1 - X^2 + X^3 - 2X^4 + o(X^5)) = 1 + X^2 - X^3 + 3X^4 - 2X^5 + o(X^5)$$

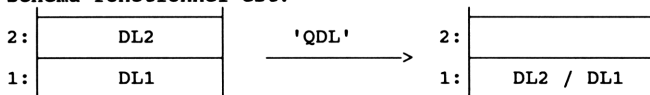
QUOTIENT DE DEUX DEVELOPPEMENTS LIMITES.

=====

'QDL' effectue le quotient de deux développements limités DL1 et DL2, tous deux exprimés sous la forme "vecteur". Le résultat est un vecteur.

L'ordre du développement limité obtenu est fonction de l'ordre et de la valuation de chacun des deux développements DL1 et DL2. En tout cas, 'QDL' donne le résultat à l'ordre le plus élevé possible (tous les coefficients obtenus sont exacts).

Le schéma fonctionnel est:



N.B: le programme 'QDL' appelle 'DIM' et le programme 'DIVPC' du répertoire 'POLY'.

Il se peut que les valuations de DL2 ou de DL1 soient nulles (DL1, DL2 représentant alors des infiniment petits). Ce qui est indispensable, c'est que la valuation de DL1 soit inférieure ou égale à celle de DL2. Dans le cas contraire, il y a un message d'erreur.

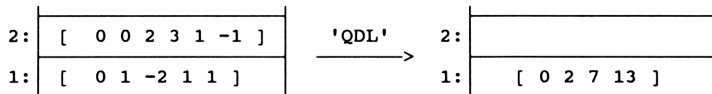
'QDL': (Checksum: # 49983d, Taille: 190.5 octets)

```

«   DUP2   DIM   ROT   DIM   ROT   MIN   3   ROLLD
   MIN   OVER   -   1   +   1   →LIST
→   q   r
«   «   OBJ→   1   GET   q   -   →ARRY
      q   1   +   ROLLD   q   DROPN   r   RDM
»
   ROT   OVER   EVAL   3   ROLLD   EVAL   r   1   GET
   POLY   DIVPC   DL
   DROP   r   RDM
»
»

```

Exemple: en moins de quatre secondes:



En effet,

si $f(X) = 2X^2 + 3X^3 + X^4 - X^5 + o(X^5)$
 et si $g(X) = X - 2X^2 + X^3 + X^4 + o(X^4)$, alors:

$$\frac{f(X)}{g(X)} = 2X + 7X^2 + 13X^3 + o(X^3).$$

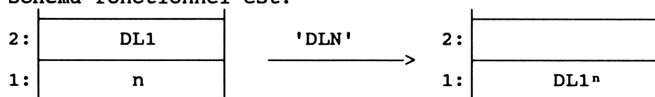
ELEVATION D'UN DEVELOPPEMENT LIMITE A UNE PUISSANCE ENTIERE.

=====

'DLN' permet d'élever un développement limité DL1 (représenté sous forme d'un vecteur) à une puissance entière n (n entier positif ou nul). Le résultat est un vecteur représentant DL1ⁿ.

L'ordre du développement obtenu dépend de l'ordre et de la valuation de DL1 (ainsi que de n évidemment). En tout cas, 'DLN' donne le résultat à l'ordre le plus élevé possible (tous les coefficients obtenus sont exacts).

Le schéma fonctionnel est:



N.B: 'DLN' appelle 'DIM' et le programme 'PRODP' du répertoire 'POLY'.
'DLN' s'appelle lui-même.

```
'DLN':( Checksum: # 48481d, Taille: 104.5 octets )
«   IF  DUP   1   ≤   THEN   DROP
    ELSE
      OVER  DUP   πDL  OVER   2   /   FLOOR   DLN
      IF  SWAP   2   MOD
        THEN   πDL
        ELSE   SWAP   DROP
      END
    END
  »
```

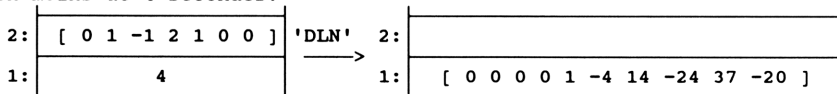
Exemples:

* en 4 secondes:



car (1 +X +2*X² -X³ +o(X³))⁵ = 1 +5*X +20*X² +45*X³ +o(X³).

* en moins de 4 secondes:

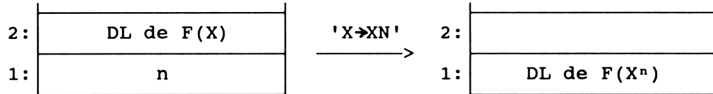


car (X -X²+2*X³+X⁴+o(X⁶))⁴
= X⁴ -4*X⁵ +14*X⁶ -24*X⁷ +37*X⁸ -20*X⁹ +o(X⁹).

**TRANSFORMATION $X \rightarrow X^N$
DANS UN DEVELOPPEMENT LIMITE.**
=====

'X→XN' permet de transformer le développement limité de F(X) en le développement limité de F(Xⁿ), où n est un entier naturel.

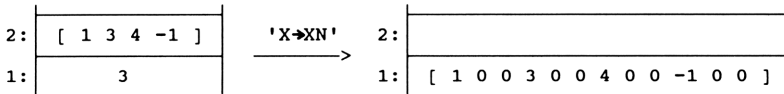
Le schéma fonctionnel est le suivant:



'X→XN': (Checksum: # 4914d, Taille: 48.5 octets)

```
«  OVER  SIZE  +  RDM
   TRN   DUP   SIZE  EVAL  *
   1     →LIST  RDM
»
```

Exemple: (en moins d'une seconde)

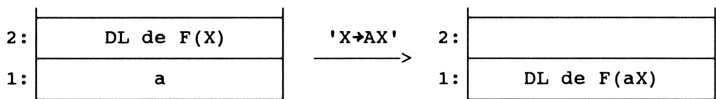


Car si $F(X) = 1 + 3X + 4X^2 - X^3 + o(X^4)$, alors
 $F(X^3) = 1 + 3X^3 + 4X^6 - X^9 + o(X^{11})$.

TRANSFORMATION $X \rightarrow A \cdot X$
DANS UN DEVELOPPEMENT LIMITE.
=====

'X→AX' permet de transformer le développement limité de $F(X)$ en le développement limité de $F(aX)$, où a est un scalaire.

Le schéma fonctionnel est le suivant:

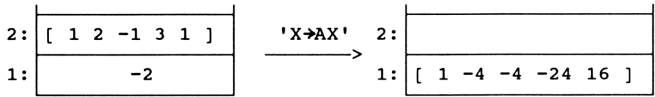


N.B.: le programme 'X→AX' appelle le programme 'DIM'.

'X→AX': (Checksum: # 55099d, Taille: 78.5 octets)

```
«  SWAP  DUP  DIM  FOR  i
    i  1  +  DUP2  GET
    4  PICK  i  ^  *  PUT
    -1  STEP
    SWAP  DROP
»
```

Exemple: (en moins d'une seconde)



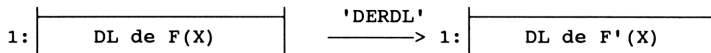
car si $F(X) = 1 + 2 \cdot X - X^2 + 3 \cdot X^3 + X^4 + o(X^4)$,
 $F(-2X) = 1 - 4 \cdot X - 4 \cdot X^2 - 24 \cdot X^3 + 16 \cdot X^4 + o(X^4)$.

DERIVATION D'UN DEVELOPPEMENT LIMITE.

=====

'DERDL' permet de passer du développement limité de $F(X)$ à celui-ci de $F'(X)$, par dérivation terme à terme de chaque coefficient (des hypothèses suffisantes de dérivabilité de F sont nécessaires).

Le schéma fonctionnel est:



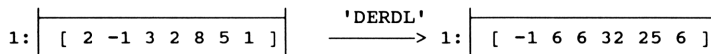
L'ordre du développement limité obtenu est évidemment inférieur d'une unité à celui du développement initial.

'DERDL':(Checksum: # 29761d, Taille: 121.5 octets)

```

«   OBJ→   1   GET   1   -
    IF   DUP   THEN
        →   t
        «   t   1   FOR   i   i   *   t   ROLL   -1   STEP
            t   →ARRY   SWAP   DROP
        »
    ELSE   DROP2   0   1   →ARRY   END
»
  
```

Exemple: (en moins d'une seconde)



Le développement limité :

$$2 -X +3*X^2 +2*X^3 +8*X^4 +5*X^5 +X^6 +o(X^6)$$

est donc transformé en:

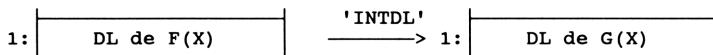
$$-1 +6*X +6*X^2 +32*X^3 +25*X^4 +6*X^5 +o(X^5).$$

INTEGRATION D'UN DEVELOPPEMENT LIMITE.

=====

'INTDL' permet de passer du développement limité de F(X) à celui-ci de la primitive G(X) de F(X) qui s'annule en 0, par intégration terme à terme de chaque coefficient.

Le schéma fonctionnel est:



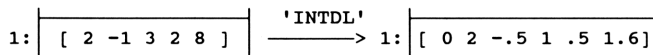
L'ordre du développement limité obtenu est évidemment supérieur d'une unité à celui du développement initial. Son terme constant (c'est à dire le premier coefficient du vecteur obtenu) est nul.

'INTDL': (Checksum: # 64552d, Taille: 89 octets)

```

"  0  SWAP  OBJ->  1  GET
  ->  t
  "  t  1  FOR  i
      i  /  t  ROLL
    -1  STEP
      t  1  +  ->ARRY
  "
  "
  
```

Exemple: (en moins d'une seconde)



Le développement limité :

$$2 -X +3*X^2 +2*X^3 +8*X^4 +o(X^4)$$

est donc transformé en:

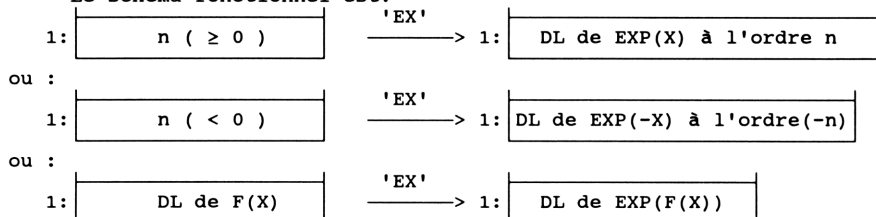
$$2*X -.5*X^2 +X^3 +.5*X^4 +1.6*X^5 +o(X^5).$$

DEVELOPPEMENT LIMITE DE EXP(X) ET EXPONENTIELLE D'UN D.L.

=====

'EX' permet tout à la fois de calculer le développement limité de $\text{EXP}(X)$ ou de $\text{EXP}(-X)$ à un ordre n quelconque, ou celui de $\text{EXP}(F(X))$ où F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



Dans ce dernier cas le D.L. de $\text{EXP}(F(X))$ est obtenu au même ordre que celui de $F(X)$.

N.B: 'EX' appelle les programmes 'DIM' et 'CPDL'

'EX': (Checksum: # 21723d, Taille: 189 octets)

```

" " → n s
" 0 n FOR i s i ^ i ! / NEXT
" n 1 + →ARRY
"
" SWAP DUP
IF DUP TYPE THEN
1 GET EXP SWAP 1 0 PUT DUP
DIM / FLOOR 1 5 ROLL EVAL CPDL *
ELSE
ABS SWAP SIGN ROT EVAL
END
"
  
```

Exemples:

* en moins d'une seconde:

```

1: [ 3 ] --'EX'--> 1: [ 1 1 .5 .166666666667 ]
  
```

* en moins d'une seconde:

```

1: [ -4 ] --'EX'--> 1: [ 1 -1 .5 -.166666666667 4.166666666667E-2 ]
  
```

* en 7 secondes:

```

[ 0 6 -1 2 1 5 ] --'EX'--> 1: [ 1 6 17 32.0000000001 49.5 76.8000000001 ]
  
```

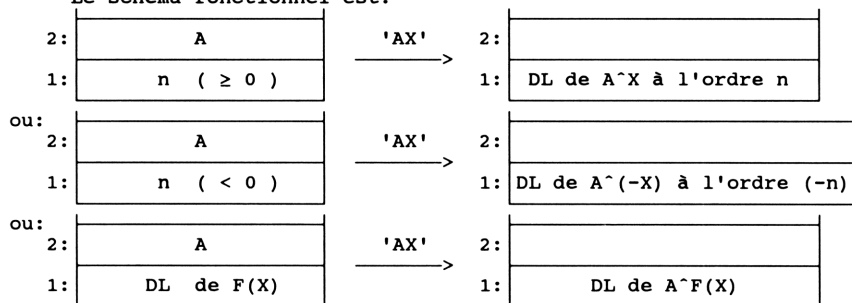
car: $\text{EXP}(6X - X^2 + 2X^3 + X^4 + 5X^5 + o(X^5))$
 $= 1 + 6X + 17X^2 + 32X^3 + 49.5X^4 + 76.8X^5 + o(X^5)$

DEVELOPPEMENT LIMITE DE A^X ET D.L. DE $A^F(X)$.

=====

'AX' permet tout à la fois de calculer le développement limité de A^X ou de $A^{(-X)}$ à un ordre n quelconque, ou celui de $A^{(F(X))}$ ou F est elle même donnée par son développement limité. A doit être > 0 .

Le schéma fonctionnel est:



Dans ce dernier cas le D.L. de $A^{(F(X))}$ est obtenu au même ordre que celui de $F(X)$.

N.B: 'AX' appelle les programmes 'EX' et 'X→AX'.

'AX':(Checksum: # 32770d, Taille: 72.5 octets)

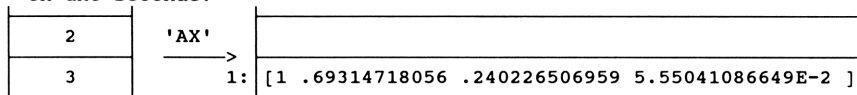
```

«   IF DUP TYPE THEN SWAP LN * EX
    ELSE EX SWAP LN X→AX END
»

```

Exemples:

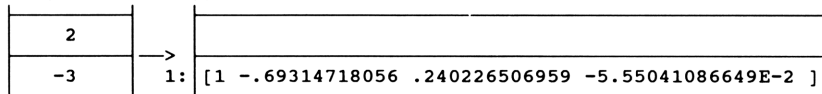
* en une seconde:



Car:

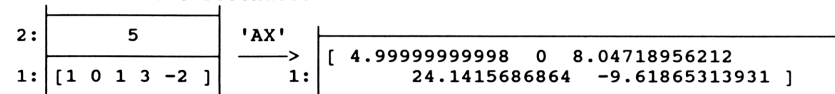
$2^X = 1 + .69314718056 * X + .240226506959 * X^2 + 5.55041086649E-2 * X^3 + o(X^3)$.

* en une seconde:



(on obtient ainsi le D.L. de $2^{(-X)}$ à l'ordre 3).

* en moins de 4 secondes:



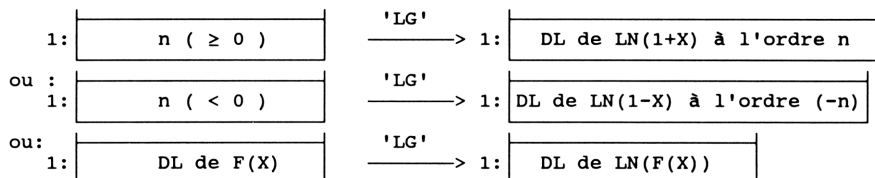
On obtient ainsi le D.L. de $5^{(1 + X^2 + 3 * X^3 - 2 * X^4 + o(X^4))}$ à l'ordre 4.

DEVELOPPEMENT LIMITE DE $\text{LN}(1+X)$ ET LOG. NEPERIEN D'UN D.L.

=====

'LG' permet tout à la fois de calculer le développement limité de $\text{LN}(1+X)$ ou de $\text{LN}(1-X)$ à un ordre n quelconque, ou celui de $\text{LN}(F(X))$ ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



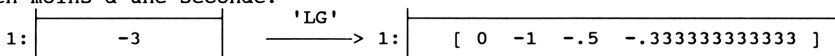
Dans ce dernier cas le D.L. de $\text{LN}(F(X))$ est obtenu au même ordre que celui de $F(X)$. De plus on doit avoir $F(0) > 0$.

N.B: 'LG' appelle les programmes 'DIM' et 'CPDL'

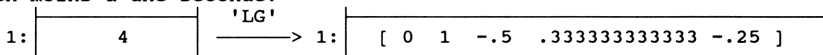
```
'LG':( Checksum: # 11671d, Taille: 223.5 octets )
"  " → n s
    " 0 IF n THEN 1 n FOR i s i ^ i / NEXT END
      n 1 + →ARRY NEG
    "
  "
  SWAP DUP
  IF DUP TYPE THEN
    1 GET DUP LN 3 ROLLD / 1 0 PUT DUP
    DIM / FLOOR -1 5 ROLL EVAL CPDL 1 ROT PUT
  ELSE
    ABS SWAP SIGN NEG ROT EVAL
  END
"
```

Exemples:

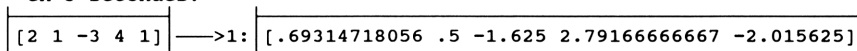
* en moins d'une seconde:



* en moins d'une seconde:



* en 5 secondes:



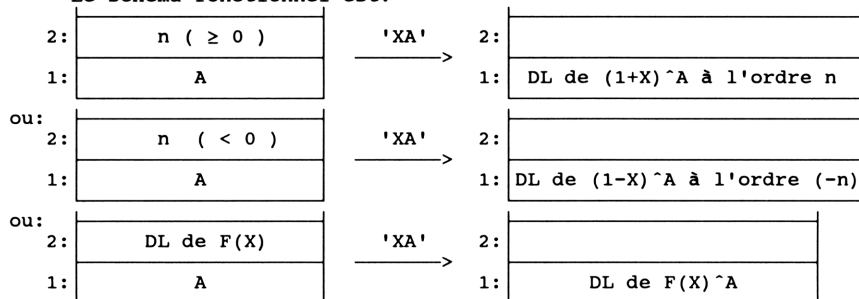
Car $\text{LN}(2 + X - 3X^2 + 4X^3 + X^4 + o(X^4)) =$
 $.69314718056 + .5X - 1.625X^2 + 2.79166666667X^3 - 2.015625X^4 + o(X^4)$.

DEVELOPPEMENT LIMITE DE $(1+X)^A$ ET D.L. DE $F(X)^A$.

=====

'XA' permet tout à la fois de calculer le développement limité de $(1+X)^A$ ou de $(1-X)^A$ à un ordre n quelconque, ou celui de $(F(X))^A$ ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



Dans ce dernier cas le D.L. de $F(X)^A$ est obtenu au même ordre que celui de $F(X)$. $F(0)$ doit être positif.

N.B: 'XA' appelle les programmes 'DIM' et 'CPDL'.

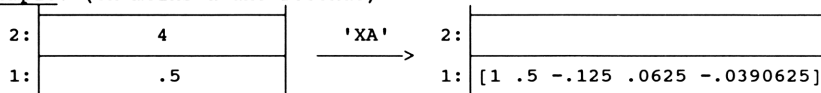
'XA': (Checksum: # 54046d, Taille: 254.5 octets)

```

" → a
  " " → n s
    " 1 IF n THEN 1 n FOR i DUP 's*(a-i+1)/i'
      EVAL * NEXT END n 1 + →ARRY
    "
  "
  SWAP DUP
  IF DUP TYPE THEN
    1 GET DUP a ^ 3 ROLLD / 1 0 PUT
    DUP DIM / FLOOR 1 5 ROLL EVAL CPDL *
  ELSE ABS SWAP SIGN ROT EVAL END
"
"

```

Exemple: (en moins d'une seconde)



car $(1+X)^{(1/2)} = 1 + (1/2)*X - (1/8)*X^2 + (1/16)*X^3 - (5/128)*X^4 + o(X^4)$

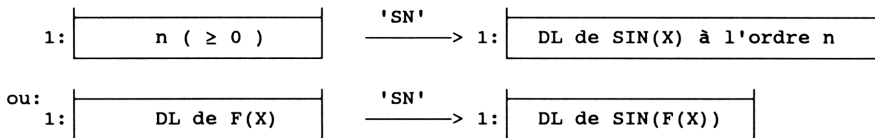
Remarque: Si vous manquez de mémoire sur votre HP48, 'XA' peut s'écrire " **SWAP LG * EX** " C'est évidemment beaucoup plus court, mais parfois un peu moins précis (prendre l'exemple ci-dessus), et en tout cas plus lent.

DEVELOPPEMENT LIMITE DE SIN(X) ET SINUS D'UN D.L.

=====

'SN' permet tout à la fois de calculer le développement limité de $\sin(X)$ à un ordre n quelconque ($n \geq 0$), ou celui de $\sin(F(X))$ ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



Dans ce dernier cas le D.L. de $\sin(F(X))$ est obtenu au même ordre que celui de $F(X)$.

N.B.: 'SN' appelle les programmes 'EX' et 'X→AX'.

'SN': (Checksum: # 2996d, Taille: 123 octets)

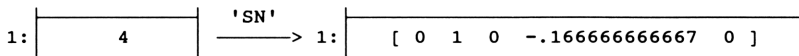
```

«   IF   DUP   TYPE   THEN
      (0,1) *   EX
    ELSE
      EX   (1,0) *   (0,1)   X→AX
    END
    IM
»

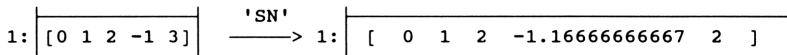
```

Exemples:

* en deux secondes:



* en 7 secondes:



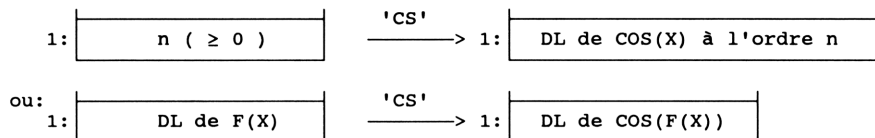
Car $\sin(X + 2X^2 - X^3 + 3X^4 + o(X^4))$
 $= X + 2X^2 - 1.166666666667X^3 + 2X^4 + o(X^4)$.

DEVELOPPEMENT LIMITE DE COS(X) ET COSINUS D'UN D.L.

=====

'CS' permet tout à la fois de calculer le développement limité de $\cos(X)$ à un ordre n quelconque ($n \geq 0$), ou celui de $\cos(F(X))$ ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



Dans ce dernier cas le D.L. de $\cos(F(X))$ est obtenu à l'ordre maximum compte tenu de la valuation du D.L. de $F(X)$.

N.B: 'CS' appelle les programmes 'DIM', 'CPDL', 'EX'et 'X→AX'.

'CS':(Checksum: # 32093d, Taille: 138 octets)

```

"  "  EX  (1,0)  *  (0,1)  X→AX  RE
"
IF  OVER  TYPE  THEN
  OVER  DIM  /  FLOOR  1  +  SWAP  EVAL  CPDL
ELSE  EVAL
END
"

```

Exemples:

* en deux secondes:

```

1: [ 5 ] --'CS'--> 1: [ 1 0 -.5 0 4.166666666667E-2 0 ]

```

* en 6 secondes:

```

1: [ 0 0 1 -3 1 5 ] --'CS'--> 1: [ 1 0 0 0 -.5 3 -5.5 -2 ]

```

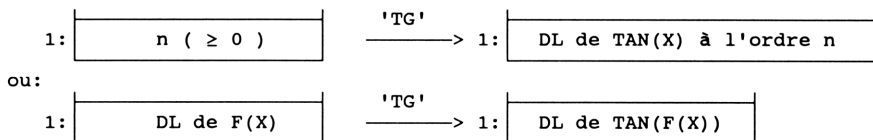
Car $\cos(X^2 - 3X^3 + X^4 + 5X^5 + o(X^5))$
 $= 1 - .5X^4 + 3X^5 - 5.5X^6 - 2X^7 + o(X^7).$

DEVELOPPEMENT LIMITE DE TAN(X) ET TANGENTE D'UN D.L.

=====

'TG' permet tout à la fois de calculer le développement limité de TAN(X) à un ordre n quelconque ($n \geq 0$), ou celui de TAN(F(X)) ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



Dans ce dernier cas le D.L. de TAN(F(X)) est obtenu au même ordre que celui de F(X).

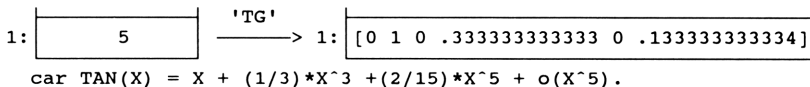
N.B.: 'TG' appelle les programmes 'CS', 'SN', 'QDL'.

'TG':(Checksum: # 40691d, Taille: 39 octets)

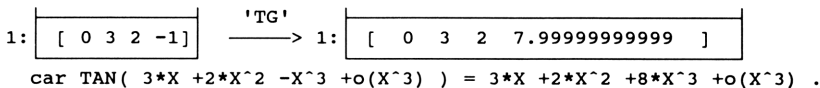
« DUP SN SWAP CS QDL »

Exemples:

* en 8 à 9 secondes:



* en 14 secondes:

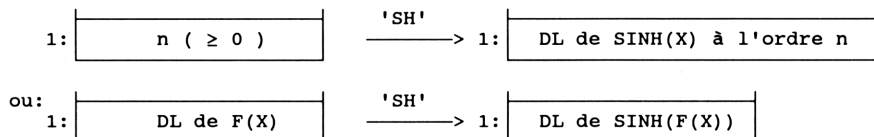


DEVELOPPEMENT LIMITE DE SINH(X) ET SINUS HYPERBOLIQUE D'UN D.L.

=====

'SH' permet tout à la fois de calculer le développement limité de $\sinh(X)$ à un ordre n quelconque ($n \geq 0$), ou celui de $\sinh(F(X))$ ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



Dans ce dernier cas le D.L. de $\sinh(F(X))$ est obtenu au même ordre que celui de $F(X)$.

N.B.: 'SN' appelle les programmes 'DIM', 'CPDL', et 'EX'.

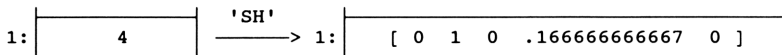
```

'SH':( Checksum: # 49613d, Taille: 104 octets )
  "  " DUP  EX  SWAP  NEG  EX  -  2  /
    "
    IF  OVER  TYPE  THEN
      OVER  DIM  /  FLOOR  SWAP  EVAL  CPDL
    ELSE  EVAL
    END
  "

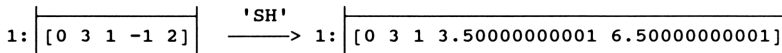
```

Exemples:

* en une seconde:



* en 5 à 6 secondes:



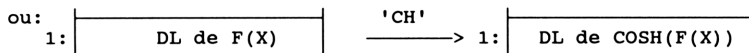
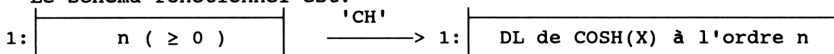
Car $\sinh(3X + X^2 - X^3 + 2X^4 + o(X^4))$
 $= 3X + X^2 + 3.5X^3 + 6.5X^4 + o(X^4)$.

DEVELOPPEMENT LIMITE DE COSH(X) ET COSINUS HYPERBOLIQUE D'UN D.L.

=====

'CH' permet tout à la fois de calculer le développement limité de $\cosh(X)$ à un ordre n quelconque ($n \geq 0$), ou celui de $\cosh(F(X))$ ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



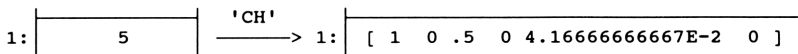
Dans ce dernier cas le D.L. de $\cosh(F(X))$ est obtenu à l'ordre maximum compte tenu de la valuation du D.L. de $F(X)$.

N.B: 'CH' appelle les programmes 'DIM', 'CPDL', et 'EX'.

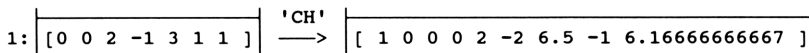
```
'CH':( Checksum: # 303d, Taille: 109 octets )
"  "  DUP  EX  SWAP  NEG  EX  +  2  /
"  "
IF  OVER  TYPE  THEN
OVER  DIM  /  FLOOR  1  +  SWAP  EVAL  CPDL
ELSE  EVAL
END
"
```

Exemples:

* en une seconde:



* en 7 à 8 secondes:



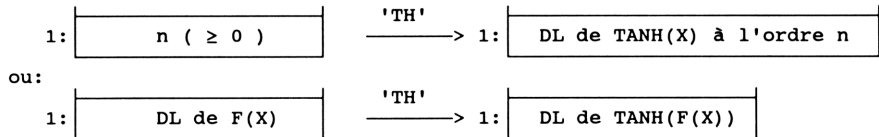
Car $\cosh(2X^2 - X^3 + 3X^4 + X^5 + X^6 + o(X^6))$
 $= 1 + 2X^4 - 2X^5 + 6.5X^6 - X^7 + 6.16666666667X^8 + o(X^8)$.

DEVELOPPEMENT LIMITE DE TANH(X) ET TANGENTE HYPERBOLIQUE D'UN D.L.

=====

'TG' permet tout à la fois de calculer le développement limité de $\text{TANH}(X)$ à un ordre n quelconque ($n \geq 0$), ou celui de $\text{TANH}(F(X))$ ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



Dans ce dernier cas le D.L. de $\text{TANH}(F(X))$ est obtenu au même ordre que celui de $F(X)$.

N.B: 'TH' appelle les programmes 'CH', 'SH', 'QDL'.

'TH': (Checksum: # 23631d, Taille: 39 octets)
 « DUP SH SWAP CH QDL »

Exemples:

* en 6 secondes:

1:

5

'TH' → 1:

[0 1 0 -.333333333333 0 .133333333334]
--

car $\text{TAN}(X) = X - (1/3)*X^3 + (2/15)*X^5 + o(X^5)$.

* en 16 secondes:

1:

[0 2 -1 3 1]

'TH' → 1:

[0 2 -1 .333333333334 5]

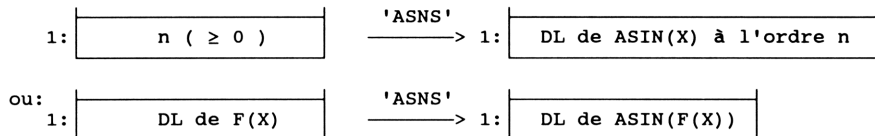
car $\text{TANH}(2*X - X^2 + 3*X^3 + X^4 + o(X^3))$
 $= 2*X - X^2 + (1/3)*X^3 + 5*X^4 + o(X^4)$.

DEVELOPPEMENT LIMITE DE ASIN(X) ET ARC-SINUS D'UN D.L.

=====

'ASNS' permet tout à la fois de calculer le développement limité de $\text{ASIN}(X)$ à un ordre n quelconque ($n \geq 0$), ou celui de $\text{ASIN}(F(X))$ ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



Dans ce dernier cas le D.L. de $\text{ASIN}(F(X))$ est obtenu au même ordre que celui de $F(X)$. Le coefficient $F(0)$ doit être nul.

N.B: 'ASNS' appelle les programmes 'DIM', 'CPDL'.

'ASNS': (Checksum: # 42167d, Taille: 222 octets)

```

" " → n
    " 0 n FOR i
        0 i DUP 2 / COMB 2 i ^ / i 1 + /
        2 STEP
        n 2 MOD NOT DROPN
        n 1 + →ARRY
    "
    IF OVER TYPE THEN
        OVER 1 GET
        IF ABS THEN DROP "Erreur"
        ELSE OVER DIM / FLOOR SWAP EVAL CPDL END
        ELSE EVAL
        END
    "
  
```

Exemples:

* en une seconde:

```

1: [ 6 ] --'ASNS'--> 1: [ 0 1 0 .166666666667 0 .075 0 ]
  
```

* en 5 secondes:

```

1: [ 0 1 -3 2 5 ] --'ASNS'--> 1: [ 0 1 -3 2.166666666667 3.5 ]
  
```

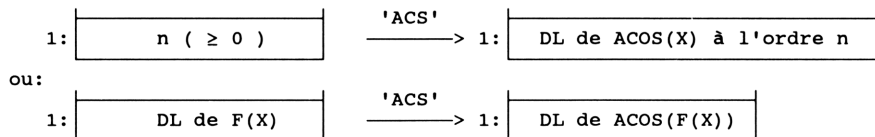
Car $\text{ASIN}(X - 3X^2 + 2X^3 + 5X^4 + o(X^4))$
 $= X - 3X^2 + 2.166666666667X^3 + 3.5X^4 + o(X^4)$.

DEVELOPPEMENT LIMITE DE ACOS(X) ET ARC-COSINUS D'UN D.L.

=====

'ACS' permet tout à la fois de calculer le développement limité de $\text{ACOS}(X)$ à un ordre n quelconque ($n \geq 0$), ou celui de $\text{ACOS}(F(X))$ ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



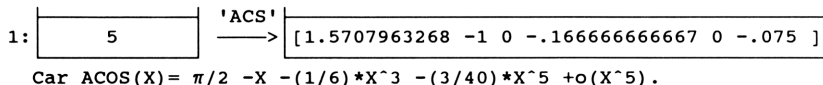
Dans ce dernier cas le D.L. de $\text{ACOS}(F(X))$ est obtenu au même ordre que celui de $F(X)$. Le coefficient $F(0)$ doit être nul.

N.B: 'ACS' appelle le programme 'ASNS'.

'ACS':(Checksum: # 63964d, Taille: 42.5 octets)

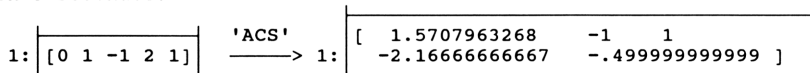
« ASNS NEG 1 π →NUM 2 / PUT »

* en une seconde:



Car $\text{ACOS}(X) = \pi/2 - X - (1/6)*X^3 - (3/40)*X^5 + o(X^5)$.

* en 5 secondes:



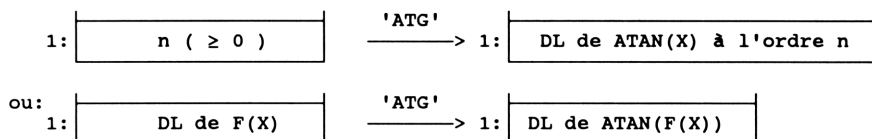
Car $\text{ACOS}(X - X^2 + 2*X^3 + X^4 + o(X^4))$
 $= \pi/2 - X + X^2 - (13/6)*X^3 - (1/2)*X^4 + o(X^4)$.

DEVELOPPEMENT LIMITE DE ATAN(X) ET ARC-TANGENTE D'UN D.L.

=====

'ATG' permet tout à la fois de calculer le développement limité de ATAN(X) à un ordre n quelconque ($n \geq 0$), ou celui de ATAN(F(X)) ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



Dans ce dernier cas le D.L. de ATAN(F(X)) est obtenu au même ordre que celui de F(X). Le coefficient F(0) doit être nul.

N.B: 'ATG' appelle les programmes 'DIM', 'CPDL'.

'ATG': (Checksum: # 23911d, Taille: 214 octets)

```

" " → n
" 0 n FOR i
    0 i 1 + INV
    0 i 3 + INV NEG
    4 STEP
    3 n 4 MOD - DROPN
    n 1 + →ARRY
"
"
IF OVER TYPE THEN
OVER 1 GET
IF ABS THEN DROP "Erreur"
ELSE OVER DIM / FLOOR SWAP EVAL CPDL END
ELSE EVAL
END
"

```

Exemples:

* en une seconde:

```

1: [ 6 ] --'ATG'--> 1: [ 0 1 0 -.333333333333 0 .2 0 ]

```

* en 5 secondes:

```

1: [ 0 3 1 -1 2 ] --'ATG'--> 1: [ 0 3 1 -9.999999999999 -6.999999999999 ]

```

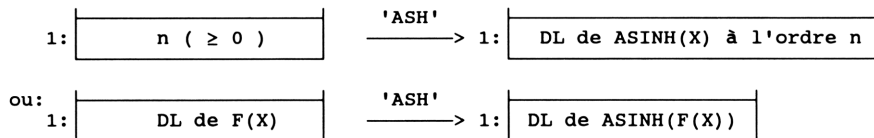
Car $ATAN(3X + X^2 - X^3 + 2X^4 + o(X^4))$
 $= 3X + X^2 - 10X^3 - 7X^4 + o(X^4)$.

DEVELOPPEMENT LIMITE DE ASINH(X) ET ARG-SINUS HYPERBOLIQUE D'UN D.L.

=====

'ASH' permet tout à la fois de calculer le développement limité de $\text{ASINH}(X)$ à un ordre n quelconque ($n \geq 0$), ou celui de $\text{ASINH}(F(X))$ ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



Dans ce dernier cas le D.L. de $\text{ASINH}(F(X))$ est obtenu au même ordre que celui de $F(X)$. Le coefficient $F(0)$ doit être nul.

N.B: 'ASH' appelle les programmes 'ASNS', 'X→AX'.

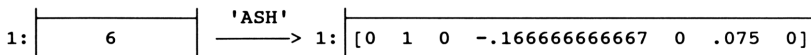
'ASH': (Checksum: # 25447d, Taille: 128 octets)

```

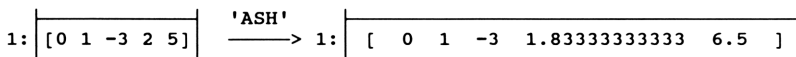
«   IF   DUP   TYPE   THEN
      (0,1) *   ASNS
    ELSE
      ASNS   (1,0) *   (0,1)   X→AX
    END
    IM
»
  
```

Exemples:

* en 2 secondes:



* en 6 à 7 secondes:



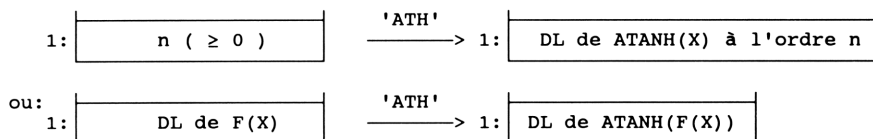
Car $\text{ASINH}(X - 3X^2 + 2X^3 + 5X^4 + o(X^4))$
 $= X - 3X^2 + (11/6)X^3 + (13/2)X^4 + o(X^4)$.

DEVELOPPEMENT LIMITE DE ATANH(X) ET ARG-TANGENTE HYPERBOLIQUE D'UN D.L.

=====

'ATH' permet tout à la fois de calculer le développement limité de ATANH(X) à un ordre n quelconque ($n \geq 0$), ou celui de ATANH(F(X)) ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



Dans ce dernier cas le D.L. de ATANH(F(X)) est obtenu au même ordre que celui de F(X). Le coefficient F(0) doit être nul.

N.B: 'ATH' appelle les programmes 'ATG', '~~X~~→AX'.

'ATH': (Checksum: # 59212d, Taille: 126 octets)

```

«   IF   DUP   TYPE   THEN
      (0,1) *   ATG
    ELSE
      ATG   (1,0) *   (0,1)   X→AX
    END
  IM
»
  
```

Exemples:

* en 2 secondes:

```

1: [ 6 ] --'ATH'--> 1: [ 0 1 0 .333333333333 0 .2 0 ]
  
```

* en 6 secondes:

```

1: [ 0 3 1 -1 2 ] --'ATH'--> 1: [ 0 3 1 7.9999999999 11 ]
  
```

Car $\text{ATANH}(3X + X^2 - X^3 + 2X^4 + o(X^4))$
 $= 3X + X^2 + 8X^3 + 11X^4 + o(X^4).$

GEOMETRIE

Le répertoire 'GEOM' contient quelques programmes de géométrie affine ou euclidienne.

Il s'agit ici essentiellement de trouver l'équation de tel ou tel ensemble de points (droite, plan, cercle, sphère) quand on en connaît les éléments essentiels, ou de calculer des distances ou des écarts angulaires.

Le répertoire 'GEOM' contient les programmes suivants:

'DRTE': équation d'une droite dont on connaît:

- * deux points
- ou * un point et un vecteur directeur.

'PLAN': équation d'un plan dont on connaît:

- * trois points
- ou * deux points et un vecteur directeur
- ou * un point et deux vecteurs directeurs.

'CERC': équation d'un cercle ou d'une sphère dont on connaît:

- * le centre et le rayon
- ou * deux points diamétralement opposés.

'DIST': calcul de la distance entre:

- * un point et une droite (en dimension 2 ou 3)
- ou * un point et un plan
- ou * deux droites (dans l'espace).

'ANGL': calcul de l'écart angulaire entre:

- * deux vecteurs
- ou * deux droites
- ou * deux plans

EQUATION D'UNE DROITE DU PLAN.

=====

'DRTE' permet d'obtenir, sous forme symbolique, l'équation d'une droite D du plan dont on connaît:

* deux points distincts P et Q.

ou * un point P et un vecteur directeur u.

P et Q (ou P et u) doivent être placés aux niveaux 1 et 2.

Un point de coordonnées x,y est représenté par le vecteur [x,y,1]
 Un vecteur de composantes x,y est représenté par [x, y, 0].
 C'est donc la troisième composante qui détermine s'il s'agit d'un point ou d'un vecteur.

L'équation est obtenue sous la forme symbolique: 'A*X+B*Y+C=0'.

'DTRE': (Checksum: # 63312d, Taille: 65 octets)

```
«   CROSS   V→   ROT   'X'   *   ROT
   'Y'   *   +   SWAP   +   0   =
»
```

Exemple1:

(en une seconde)

Equation de la droite passant par les points P(-1,3) et Q(2,5).

2:	[	-1	3	1	]		2:
1:	[	2	5	1	]		1:

—————> 'DRTE'

2:	[						2:
1:	[	- (2*X)	+3*Y	-11=0	]		1:

Exemple2:

(en une seconde)

Equation de la droite passant par le point P(3,2) et de vecteur directeur u(4,1).

2:	[	3	2	1	]		2:
1:	[	4	1	0	]		1:

—————> 'DRTE'

2:	[						2:
1:	[	-X+4*Y	-5=0	]			1:

Dans l'exemple ci-dessus, on peut inverser l'ordre dans lequel sont donnés P et u.

EQUATION D'UN PLAN.

=====

'PLAN' permet d'obtenir, sous forme symbolique, l'équation d'un plan (π) dont on connaît:

- * trois points P, Q, et R.
- ou * deux points P et Q et un vecteur directeur u.
- ou * un point P et deux vecteurs directeurs u et v.

Un point de coordonnées x,y,z est représenté par le vecteur:

[x y z 1].

Un vecteur de composantes x,y,z est représenté par le vecteur:

[x y z 0].

C'est donc la quatrième composante qui détermine s'il s'agit d'un vecteur ou d'un point.

Les trois éléments P,Q,R (ou P,Q,u) (ou P,u,v) doivent être placés aux niveaux 1,2 et 3 de la pile (dans un ordre quelconque).

L'équation est alors obtenue au niveau 1, sous une forme symbolique du type: 'A*X+B*Y+C*Z+D=0'.

'PLAN': (Checksum: # 63868d, Taille: 166.5 octets)

```

" { X Y Z 1 } → a b c L
  " 0 0 0 0 0 a V→ b V→ c V→
    { 4 4 } →ARRY 0
    1 4 FOR k
      OVER k 1 PUT DET L k GET * +
    NEXT
    0 = SWAP DROP
  "
"

```

Exemple1:

Equation du plan passant par les points:

P(-1,2,3) Q(4,1,5) et R(2,-3,0):

(en 3 secondes)

3:	[-1 2 3 1]	→ 'PLAN' →	3:	
2:	[4 1 5 1]		2:	
1:	[2 -3 0 1]		1:	'13*X+21*Y-22*Z+37=0'

Exemple2:

Equation du plan passant par le point P(-1,1,-3) et dirigé par les vecteurs u(-2,1,3) et v(-1,4,-2).

(en 3 secondes)

3:	[-2 1 3 0]	→ 'PLAN' →	3:	
2:	[-1 1 -3 1]		2:	
1:	[-1 4 -2 0]		1:	'14*X+7*Y+7*Z+28=0'

EQUATION D'UN CERCLE OU D'UNE SPHERE.

=====

'CERC' permet d'obtenir, sous forme symbolique, l'équation d'un cercle (C) du plan, ou d'une sphère (S) de l'espace, dont on connaît:

- * le centre Ω et le rayon R.
- ou * deux points diamétralement opposés.

Un point est ici représenté par le vecteur de ses coordonnées [x,y] (si on est dans le plan) ou [x y z] (dans l'espace).

Le rayon R est un nombre réel positif.

Les deux éléments Ω et R (ou A et B) doivent être placés aux niveaux 1 et 2, dans un ordre quelconque. L'équation est alors obtenue au niveau 1, sous une forme symbolique du type:

'X^2+Y^2+A*X+B*Y+C=0' dans le cas d'un cercle.

'X^2+Y^2+Z^2+A*X+B*Y+C*Z+D=0' dans le cas d'une sphère.

'CERC': (Checksum: # 31394d, Taille: 311.5 octets)

```

"  DUP2 TYPE SWAP TYPE IF < THEN SWAP END
  DUP SIZE 1 GET -> b a d
  " IF b TYPE 3 == THEN
    b a + 2 / b a - 2 /
    ABS 'b' STO 'a' STO
  END
  0 1 d FOR k
    "" 87 k + CHR + OBJ-> 2 ^ +
  NEXT
  1 d FOR k
    a k GET 2 * NEG ""
    87 k + CHR + OBJ-> * +
  NEXT
  a DUP DOT b SQ - + 0 =
"
»

```

Exemple1: (en deux secondes)

Equation du cercle de centre $\Omega(2,3)$ et de rayon 5.

2:	[2 3]	→ 'CERC' →	2:	
1:	5		1:	'X^2+Y^2-4*X-6*Y-12=0'

Exemple2: (en 3 à 4 secondes)

Equation de la sphère de diamètre AB avec A(1,1,1) et B(3,5,5).

2:	[1 1 1]	→ 'CERC' →	2:	
1:	[3 5 5]		1:	'X^2+Y^2+Z^2-4*X-6*Y-6*Z+13=0'

CALCUL DE DISTANCES.

=====

'DIST' permet de calculer la distance entre:

- * un point A(x,y) et une droite définie par son équation (dans le plan)
- ou * un point A(x,y,z) et un plan défini par son équation.
- ou * un point et une droite définie par un point et un vecteur directeur (dans le plan ou dans l'espace).
- ou * deux droites définies chacune par un point et un vecteur directeur (dans l'espace).

Un point A(x,y) est représenté par le vecteur [x y].

Un point A(x,y,z) est représenté par le vecteur [x y z].

La droite d'équation 'ax+by+c=0' est représentée par [a b c].

Le plan 'ax+by+cz+d=0' est représenté par [a b c d].

Quand une droite est définie par un point A(x,y,z) (respectivement A(x,y)) et un vecteur directeur u(a,b,c) (respectivement u(a,b)), elle est représentée sur la pile par une liste: (dans cet ordre)

{ [x y z] [a b c] }
(respectivement { [x y] [a b] }).

Les deux objets dont on veut calculer la distance sont placés aux niveaux 1 et 2 de la pile. Si l'un d'eux est un point, il doit obligatoirement être placé au niveau 2.

La distance cherchée est obtenue au niveau 1.

'DIST': (Checksum: # 49003d, Taille: 344.5 octets)

```

«   DUP2  TYPE  SWAP  TYPE
    →  t1  t2
    «   CASE
        t1  3  ==  t2  3  ==  AND  THEN
            →  p  e
            «   e  p  SIZE  RDM  DUP  p  DOT  e
                DUP  SIZE  GET  +  ABS  SWAP  ABS  /
            »
        END
        t2  3  ==  t1  5  ==  AND  THEN
            →  p  d
            «   d  EVAL  DUP  ROT  p  -
                CROSS  ABS  SWAP  ABS  /
            »
        END
        →  d1  d2
        «   d1  2  GET  d2  2  GET  DUP2  V→
            4  ROLL  V→  d1  1  GET
            d2  1  GET  -  V→  {3 3}  →ARRY
            DET  ABS  3  ROLLD  CROSS  ABS  /
        »
    »
END
»
»

```

PROGRAMME 'DIST' EXEMPLES D'UTILISATION.

=====

Exemple1: (en moins d'une seconde)

Distance du point $M(1,-2)$ à la droite d'équation $3x+4y+1=0$.

2:	[1 -2]	'DIST'	2:	
1:	[3 4 1]	→	1:	.8

Exemple2: (en une seconde)

Distance du point $M(2,-1,3)$ au plan d'équation $x+2y+2z-5=0$.

2:	[2 -1 3]	'DIST'	2:	
1:	[1 2 2 -5]	→	1:	.333333333333

Exemple3: (en moins d'une seconde)

Distance du point $M(1,-2)$ à la droite définie par le point $A(5,-4)$ et le vecteur directeur $u=(-4,3)$. Ce sont les mêmes données que dans l'exemple1.

2:	[1 -2]	'DIST'	2:	
1:	{ [5 -4] [-4 3] }	→	1:	0.8

Exemple4: (en moins d'une seconde)

Distance du point $M(3,-2,1)$ à la droite D définie par le point $A(-1,1,4)$ et le vecteur directeur $u(2,-1,5)$.

2:	[3 -2 1]	'DIST'	2:	
1:	{ [-1 1 4] [2 -1 5] }	→	1:	5.78503817331

Exemple5: (en une seconde)

Distance de la droite D définie par le point $A(1,0,4)$ et le vecteur $u(2,3,5)$ à la droite D' définie par le point $B(-1,-5,2)$ et le vecteur $v(4,3,0)$.

2:	{ [1 0 4] [2 3 5] }	'DIST'	2:	
1:	{ [-1 -5 2] [4 3 0] }	→	1:	2.25593854059

CALCUL D'ECART ANGULAIRE:

=====

'ANGL' calcule l'écart angulaire θ ($0 \leq \theta \leq \pi/2$) compris entre:

- * deux vecteurs u et v (du plan ou de l'espace)
- ou * deux droites de vecteurs directeurs connus u et v (dans le plan ou dans l'espace)
- ou * deux plans connus par leurs équations.

Un vecteur u(a,b) (resp. u(a,b,c)) est représenté par [a b],
(resp [a b c]).
[a b] (resp. [a b c]) permet également de désigner toute droite d'équation $ax+by+c=0$ (resp. tout plan d'équation $ax+by+cz+d=0$).

Les deux objets dont on veut calculer l'écart angulaire doivent être placés au niveau 1 et 2.

L'écart angulaire θ est alors obtenu à la fois en radians, en degrés (avec le format HMS), et en grades. Les 3 résultats sont donnés avec quatre décimales sous forme de réels marqués (voir l'exemple ci-dessous).

'ANGL': (Checksum: # 11528d, Taille: 120 octets)

```

«   RAD   4   FIX   DUP2   DOT   ABS   SWAP
   ABS   ROT   ABS   *   /   ACOS   "radians"   →TAG
   DUP   R→D   →HMS   "deg(HMS)"   →TAG
   DUP   HMS→   .9   /   "grades"   →TAG
»

```

Exemple:

Ecart angulaire entre les vecteurs u(1,-4,2) et v(3,1,5).
(en une seconde)

2: [1 -4 2] 1: [3 1 5]	→ → →	4: radians: 1.2324 3: deg(HMS): 70.3642 2: grades: 78.4573 1:
--	--	---

GEOMETRIE DIFFERENTIELLE

Le répertoire 'GDIF' contient un certain nombre de programmes de géométrie différentielle. Il s'agit de la résolution de problèmes géométriques où interviennent des calculs de dérivées et de primitives.

Dans ce domaine, la HP48SX permet de programmer simplement des problèmes relativement complexes (et dont la résolution dans un langage de programmation courant comme "Turbo-Pascal" poserait des problèmes délicats).

On utilise en effet dans ces programmes la possibilité qu'a le HP48 d'intégrer et surtout de dériver une fonction écrite sous forme symbolique. Il est en particulier très intéressant de pouvoir disposer des expressions des dérivées partielles d'une fonction (par rapport à telle ou telle de ses variables).

Le répertoire 'GDIF' contient les programmes suivants:

- 'RECTP' : Rectification (calcul de la longueur) d'une courbe plane (donnée par une équation $Y=F(X)$, ou en polaires, ou par un paramétrage $X=X(T)$, $Y=Y(T)$).
- 'RECTG' : Rectification (calcul de la longueur) d'une courbe gauche (en dimension 3), donnée sous forme paramétrée.
- 'CURV' : Calcul d'une intégrale curviligne, dans le plan ou dans l'espace, le long d'un arc paramétré.
- 'AIRE' : Aire d'un domaine plan limité par une courbe fermée, cette dernière étant connue par une représentation sous forme paramétrique (polaires ou cartésiennes).
- 'CRBRE' : Calcul du rayon de courbure et du centre de courbure en un point quelconque d'une courbe plane.
- 'DIVRG' : Calcul de la divergence d'un champ de vecteurs.
- 'ROTA' : Calcul du rotationnel d'un champ de vecteurs.
- 'GRADI' : Gradient d'une fonction de plusieurs variables.
- 'DELTA' : Laplacien d'une fonction de plusieurs variables.
- 'DIFF' : Calcul de la différentielle d'une fonction de plusieurs variables.

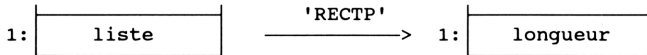
RECTIFICATION D'UNE COURBE PLANE.

=====

'RECTP' calcule, de façon approchée, la longueur d'une courbe plane définie d'une des trois manières suivantes:

- * par une équation cartésienne $Y=F(X)$, où $A \leq X \leq B$.
- * par une représentation paramétrique $X=X(T)$, $Y=Y(T)$, où $A \leq T \leq B$.
- * par une équation en polaires $RO=RO(T)$, où $A \leq T \leq B$.

Le schéma fonctionnel est:



où "liste" est une liste contenant les éléments du calcul,
où "longueur" est une valeur approchée du résultat,

L'organisation de "liste" est la suivante:

{ F(X) A B } dans le premier cas,
{ X(T) Y(T) A B } dans le deuxième cas,
{ RO(T) A B } dans le dernier cas,
Avec:
A: valeur inférieure du paramètre (X ou T).
B: valeur supérieure du paramètre (X ou T).
F(X): expression symbolique de la variable 'X'.
X(T),Y(T),RO(T): expressions symboliques de la variable 'T'.

La précision du calcul intégral est fonction du mode d'affichage:

- * En mode **n FIX**, les intégrales sont calculées avec une précision de n décimales.
- * En mode **STD** les intégrales sont calculées avec la précision maximum (ça peut s'avérer long...)

A la sortie, la variable **IERR** contient un majorant de l'erreur absolue commise.

```

'RECTP': ( Checksum: # 62297d, Taille: 268.5 octets )
«  OBJ→ { X T }  PURGE
   →  a  b  n
   «  IF  n  4  ==  THEN
      'T'  ð  2  ^  SWAP  'T'  ð  2  ^  +  √  'T'
    ELSE
      DUP  'T'  ð
      IF  DUP  0  SAME  THEN
        DROP  'X'  ð  2  ^  1  +  √  'X'
      ELSE
        2  ^  SWAP  2  ^  +  √  'T'
      END
    END
  END
  a  b  4  ROLL  4  ROLL  ∫  →NUM
»
»

```

PROGRAMME 'RECTP': EXEMPLES D'UTILISATION.

=====

Exemple 1:

Longueur de l'arc de chaînette $Y=CH(X)$, et $0 \leq X \leq 1$.
(On se place préalablement en mode 5 FIX)

2:		'RECTP'	2:	
		→		
1:	{ 'COSH(X)' 0 1 }		1:	1.17520

Le résultat est obtenu en environ 6 secondes. On montre que le
résultat exact est: $SH(1) = (e - 1/e) / 2 \approx 1.17520119364$.

Exemple 2:

Longueur de l'arc de cycloïde $X=\cos(T)$, $Y=T-\sin(T)$, $0 \leq T \leq 2\pi$.
(On se place préalablement en mode 3 FIX)

2:		'RECTP'	
		→	
1:	{ 'COS(T)' 'T-SIN(T)' 0 '2*π' }	1:	8.000

Le résultat est obtenu en environ 9 secondes. On montre que le
résultat exact est 8.

Exemple 3:

Longueur de la cardioïde $R_0=1+\cos(T)$.
(On se place préalablement en mode 4 FIX)
Pour obtenir la longueur totale, T doit varier de 0 à 2π .

2:		'RECTP'	
		→	
1:	{ '1+cos(T)' 0 '2*π' }	1:	8.0000

Le résultat est obtenu en environ 9 secondes. On montre que le
résultat exact est 8.

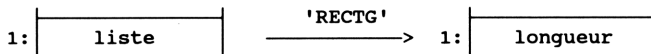
RECTIFICATION D'UNE COURBE GAUCHE.

=====

'RECTG' calcule, de façon approchée, la longueur d'une courbe gauche définie d'une des deux manières suivantes:

- * par une représentation paramétrique $X=X(T), Y=Y(T), Z=Z(T)$, où T parcourt le segment [A, B].
- * par une représentation paramétrique en coordonnées cylindriques $RO=RO(T), Z=Z(T)$, avec $A \leq T \leq B$.

Le schéma fonctionnel est:



où "liste" est une liste contenant les éléments nécessaires au calcul,
où "longueur" est une valeur approchée du résultat,

L'organisation de "liste" est la suivante:

- { X(T) Y(T) Z(T) A B } dans le premier cas,
- { RO(T) Z(T) A B } dans le deuxième cas.

Avec:

A: valeur inférieure du paramètre T.

B: valeur supérieure du paramètre T.

X(T),Y(T),Z(T),RO(T): expressions symboliques de la variable 'T'.

La précision du calcul intégral est fonction du mode d'affichage:

- * En mode **n FIX**, les intégrales sont calculées avec une précision de n décimales.
- * En mode **STD** les intégrales sont calculées avec la précision maximum (ça peut s'avérer long...)

A la sortie, la variable **IERR** contient un majorant de l'erreur absolue commise.

'RECTG': (Checksum: # 41324d, Taille: 202.5 octets)

```

"  OBJ-> 'T'  PURGE
    ->  a  b  n
    "  'T'  ð  2  ^  SWAP
      IF  n  5  ==  THEN
        'T'  ð  2  ^  +  SWAP  'T'  ð  2  ^
      ELSE
        DUP  'T'  ð  2  ^  SWAP  2  ^  +
      END
    +  √  a  b  ROT  'T'  ∫  ->NUM
  "
  "
  
```


**PROGRAMME 'RECTG':
EXEMPLES D'UTILISATION.**

=====

Exemple 1:

Longueur de l'arc défini par la représentation paramétrique:

$$Y=X^2, \quad Z=(2/3)X^3, \quad \text{où } 0 \leq X \leq 1.$$

Ici il faut prendre X pour paramètre T.

On se place préalablement en mode 4 FIX.

2:			
1:	{ 'T' 'T^2' '2*T^3/3' 0 1 }	'RECTG'>	1: 1.6667

Le résultat est obtenu en 10 secondes.

On montre que le résultat exact est $5/3 \approx 1.66666666667$.

Exemple 2:

Longueur de l'arc d'hélice circulaire défini en coordonnées cylindriques par: $R0=\cos(T)$, $Z=T$, où $0 \leq T \leq 2\pi$.

On se place préalablement en mode 4 FIX.

2:			
1:	{ 'COS(T)' 'T' 0 '2*π' }	'RECTG'>	1: 8.8858

Le résultat est obtenu en 4 secondes.

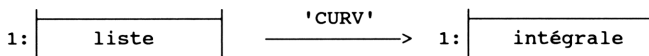
On montre que le résultat exact est $2\sqrt{2}\pi \approx 8.8857658763$.

CALCUL D'INTEGRALES CURVILIGNES.

=====

'CURV' calcule l'intégrale curviligne:

- 1) de la forme différentielle
 $w = P(X,Y)dX + Q(X,Y)dY$
 le long de l'arc $X=X(T), Y=Y(T)$, avec $A \leq T \leq B$.
 - 2) de la forme différentielle
 $w = P(X,Y,Z)dX + Q(X,Y,Z)dY + R(X,Y,Z)dZ$
 le long de l'arc $X=X(T), Y=Y(T), Z=Z(T)$, avec $A \leq T \leq B$.
- Le schéma fonctionnel est:



où "liste" est une liste contenant les éléments nécessaires au calcul,
 où "intégrale" est une valeur approchée du résultat,

L'organisation de "liste" est la suivante:

{P(X,Y) Q(X,Y) X(T) Y(T) Z(T) A B} dans le premier cas,
 {P(X,Y,Z) Q(X,Y,Z) R(X,Y,Z) X(T) Y(T) Z(T) A B} dans le 2me cas.
 Avec:
 A: valeur inférieure du paramètre T.
 B: valeur supérieure du paramètre T.
 P(X,Y), Q(X,Y): expressions symboliques en X,Y.
 P(X,Y,Z) Q(X,Y,Z) R(X,Y,Z): expressions symboliques en X,Y,Z.
 X(T),Y(T),Z(T): expressions symboliques de la variable 'T'.

La précision du calcul intégral est fonction du mode d'affichage:

- * En mode **n FIX**, les intégrales sont calculées avec une précision de n décimales.
- * En mode **STD** les intégrales sont calculées avec la précision maximum (ça peut s'avérer long...)

A la sortie, la variable **IERR** contient un majorant de l'erreur absolue commise.

'CURV': (Checksum: # 58416d, Taille: 282.5 octets)

```

"  OBJ-> 'T'  PURGE
    ->  a  b  n
    "  IF  n  8  ==  THEN  'Z'  STO  END
        'Y'  STO  'X'  STO
        IF  n  8  ==  THEN
            'Z'  'T'  ð  *
        ELSE  0
    END
    3  ROLL  'Y'  'T'  ð  *  SWAP
    'X'  'T'  ð  *  +  +
    a  b  ROT  'T'  SHOW  'T'  ∫  →NUM
    { X Y Z }  PURGE
"  "
  
```

PROGRAMME 'CURV': EXEMPLES D'UTILISATION.

=====

Exemple 1:

Soit à calculer l'intégrale curviligne $\int_{\Gamma} (2-y)dx + xdy$,
où Γ est l'arche de cycloïde:
 $X=T-\text{SIN}(T)$, $Y=1-\text{COS}(T)$, $0 \leq T \leq 2*\pi$.

On se place préalablement en mode 4 FIX.

On place la liste

{ '2-Y' 'X' 'T-SIN(T)' '1-COS(T)' 0 '2*\pi' }
au niveau 1 de la pile et on appelle 'CURV'.

Au bout de 17 secondes, on obtient:

1:	-6.2832
----	---------

On montre que le résultat exact est:

$-2\pi \approx -6.28318530718$.

Exemple 2:

Soit à calculer l'intégrale curviligne

$$\int_{\Gamma} (y-z)dx + (z-x)dy + (x-y)dz,$$

où Γ est la spire d'hélice $X=\text{COS}(T)$, $Y=\text{SIN}(T)$, $Z=T$, $0 \leq T \leq 2\pi$.

On se place préalablement en mode 4 FIX.

On place la liste :

{ 'Y-Z' 'Z-X' 'X-Y' 'COS(T)' 'SIN(T)' 'T' 0 '2*\pi' }
au niveau 1 de la pile et on appelle 'CURV'.

Au bout de 19 secondes, on obtient:

1:	-12.5664
----	----------

On montre que le résultat exact est:

$-4\pi \approx -12.5663706144$.

CALCUL D'AIRES PLANES.

=====

'AIRE' calcule l'aire du domaine plan limité par une courbe fermée Γ définie:

- 1) par une représentation paramétrique $X=X(T)$, $Y=Y(T)$, $A \leq T \leq B$.
- 2) en coordonnées polaires $RO=RO(T)$ avec $A \leq T \leq B$.

Le schéma fonctionnel est:



où "liste" est une liste contenant les éléments nécessaires au calcul,
où "aire" est une valeur approchée du résultat,

L'organisation de "liste" est la suivante:

- { $X(T)$ $Y(T)$ A B } dans le premier cas,
- { $RO(T)$ A B } dans le deuxième cas.

Avec:

A: valeur inférieure du paramètre T.

B: valeur supérieure du paramètre T.

$X(T), Y(T), RO(T)$: expressions symboliques de la variable 'T'

La précision du calcul intégral est fonction du mode d'affichage:

- * En mode **n FIX**, les intégrales sont calculées avec une précision de n décimales.
- * En mode **STD** les intégrales sont calculées avec la précision maximum (ça peut s'avérer long...)

A la sortie, la variable **IERR** contient un majorant de l'erreur absolue commise.

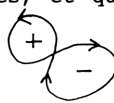
Remarques:

Dans le premier cas, l'intégrale calculée est $\int_{\Gamma} XdY$, où Γ est la courbe frontière parcourue dans le sens $T=A \rightarrow T=B$.

Dans le second cas, l'intégrale calculée est $(1/2) \int_A^B RO^2(T) dT$.

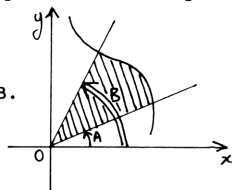
Le sens $A \rightarrow B$ doit correspondre à une orientation de Γ dans le sens trigonométrique (lors du parcours de la courbe frontière, les points du domaine intérieur doivent être sur la gauche). dans le cas contraire, on obtient une aire comptée négativement.

Dans le cas où la courbe admet des points doubles, et que le domaine intérieur est formé de plusieurs composantes, celles de ces composantes qui sont parcourues dans le sens trigonométrique sont comptées positivement, les autres négativement.



Aucun contrôle n'est effectué par le programme pour vérifier que la courbe Γ est effectivement fermée.

Dans le cas d'une courbe définie en coordonnées polaires, et si la courbe n'est pas fermée, on obtient l'aire du domaine limité par la courbe Γ et les demi-droites d'angles polaires A et B.



TEXTE DU PROGRAMME 'AIRE'. EXEMPLES D'UTILISATION.

=====

```
'AIRE': ( Checksum: # 57317d, Taille: 134 octets )
"
  OBJ→ 'T'  PURGE
  → a b n
  " IF n 3 == THEN
    2 ^ 2 /
  ELSE
    'T' ∂ *
  END
  a b ROT 'T' ∫ →NUM
"
"
```

Exemple 1:

On veut déterminer l'aire intérieure à l'ellipse d'équation:
 $X^2/4 + Y^2/9 = 1$
 Une représentation paramétrique de cette ellipse est donnée par:
 $X=2\cos(T)$, $Y=3\sin(T)$, avec $0 \leq T \leq 2\pi$.
 On se place préalablement en mode 5 FIX.

	'AIRE'	
{ '2*COS(T)' '3*SIN(T)' 0 '2*π' }	→	1: 18.84956

Le résultat est obtenu en 8 secondes.
 On montre que le résultat exact est $6\pi \approx 18.8495559215$.

Exemple 2:

On veut déterminer l'aire intérieure à la cardioïde $RO=1+\cos(T)$,
 entièrement parcourue quand T parcourt le segment $[0, 2\pi]$.
 On se place préalablement en mode 6 FIX.

L'utilisation du programme 'AIRE' donne ici:

2:			
1:	{ '1+COS(T)' 0 '2*π' }	→	1: 4.712389

Le résultat est obtenu en 15 secondes.
 On montre que le résultat exact est $3\pi/2 \approx 4.71238898038$.

CENTRE ET RAYON DE COURBURE D'UNE COURBE PLANE.

=====

'CRBRE' calcule le centre de courbure, et le rayon de courbure, en un point quelconque d'une courbe plane définie:

- 1) par une équation $Y=F(X)$.
- 2) par une représentation paramétrique $X=X(T)$, $Y=Y(T)$.
- 3) par une représentation en polaires $RO=RO(T)$.

La marche à suivre est la suivante:

Placer au niveau 1 de la pile,

* l'expression algébrique donnant $F(X)$ (premier cas).

* la liste { $X(T)$ $Y(T)$ } donnant les expressions algébriques $X(T)$ et $Y(T)$ (deuxième cas).

* l'expression algébrique donnant $RO(T)$ (troisième cas).

Appeler le programme 'CRBRE'.

Au bout d'un certain laps de temps (nécessaire aux calculs de dérivées partielles), le programme 'CRBRE' est suspendu.

Un menu personnalisé apparaît, l'écran devenant:

4: 3: 2: 1:					
→PAR		CENT	RAYO		EXIT

Pour se placer en un point déterminé de la courbe, on met la valeur du paramètre au niveau 1 et on appuie sur la touche "→PAR".

Une pression sur "CENT" donne les coordonnées (XR,YR) du centre de courbure (sous forme d'un nombre complexe).

Une pression sur "RAYO" donne le rayon de courbure R.

On peut répéter ces opérations en un nombre de points quelconque. Pendant ce temps le programme 'CRBRE' reste suspendu.

Pour sortir du programme, il suffit d'appuyer sur la touche "EXIT".

Les calculs de R et (XR,YR) sont très rapides (le programme 'CRBRE' détermine en effet l'expression symbolique de R, XR, YR. Un appui sur les touches correspondantes ne faisant alors qu'évaluer ces expressions au point choisi précédemment par l'utilisateur).

N.B: le calcul du rayon de courbure est effectué en considérant que la courbe est orienté dans le sens des paramètres croissants.

PROGRAMME 'CRBRE': EXEMPLES D'UTILISATION.

=====

Exemple 1:

Rayon de courbure et courbure de la chaînette $Y = CH(X)$.
 On place 'COSH(X)' au niveau 1 et on appelle 'CRBRE'.
 Au bout de deux secondes, le menu personnalisé apparaît.
 Si on donne au paramètre (ici X) la valeur 0, on trouve:
 Centre de courbure: (0, 2) , Rayon de courbure: 1.
 Si on donne au paramètre la valeur 1, on trouve (en mode 5 FIX):
 Centre de courbure: (-0.81343, 3.08616) ,
 Rayon de courbure: 2.38110.
 On sort du programme par un appui sur "EXIT".

Remarque: ici les valeurs théoriques sont au point d'abscisse X:
 $XR = X - \cosh(X) * \sinh(X)$, $YR = 2 * \cosh(X)$, $R = \cosh(X)^2$.

Exemple 2:

Rayon de courbure et courbure de la cycloïde
 $X(T) = T - \sin(T)$, $Y(T) = 1 - \cos(T)$.
 On {'T-SIN(T)' '1-COS(T)'} au niveau 1 et on appelle 'CRBRE'.
 Au bout de deux secondes, le menu personnalisé apparaît.
 Si on donne au paramètre (ici T) la valeur π , on trouve:
 (En mode 5 FIX)
 Centre de courbure: (3.14159, -2.00000) ,
 Rayon de courbure: -4.00000.

Remarque: ici les valeurs théoriques sont au point de paramètre T:
 $XR = T + \sin(T)$, $YR = -1 + \cos(T)$, $R = -4 * \sin(T/2)$.

Exemple 3:

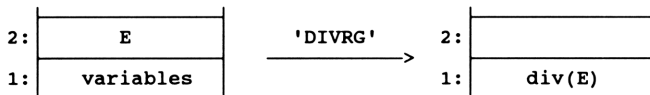
Rayon de courbure et courbure de la cardioïde définie par son
 équation en coordonnées polaires $RO = 1 + \cos(T)$.
 On place '1+cos(T)' au niveau 1 et on appelle 'CRBRE'.
 Au bout de 9 secondes, le menu personnalisé apparaît.
 Si on donne au paramètre la valeur 0, on trouve (mode 5 FIX):
 Centre de courbure: (0.66667, 0.00000) ,
 Rayon de courbure: 1.33333.
 Si on donne au paramètre la valeur $\pi/2$, on trouve (mode 5 FIX):
 Centre de courbure: (0.66667, 0.33333) ,
 Rayon de courbure: 0.94281.

Remarque: La valeur théorique de R est, au point de paramètre T:
 $R = 4/3 * \cos(T/2)$.

DIVERGENCE D'UN CHAMP DE VECTEURS.

=====

'DIVRG' calcule la divergence $\text{div}(E)$ d'un champ de vecteurs E . Le schéma fonctionnel est:



Le champ E doit ici être représenté par la liste de ses composantes, chacune d'elles étant une expression algébrique.

"variables" désigne la liste des variables par rapport auxquelles doivent être calculées les dérivées partielles.

Dans le cas par exemple d'un champ de l'espace à trois dimensions, les coordonnées étant X, Y et Z , le champ E est représenté par la liste de ses trois composantes (P, Q, R) , où $P=P(X, Y, Z)$, $Q=Q(X, Y, Z)$, $R=R(X, Y, Z)$ sont des expressions par rapport aux variables X, Y, Z , et "variables" est la liste $\{ X \ Y \ Z \}$. $\text{Div}(E)$ est alors égal à:

$$\frac{\partial P}{\partial X} + \frac{\partial Q}{\partial Y} + \frac{\partial R}{\partial Z}.$$

le programme 'DIVRG' est suspendu (après calcul des dérivées partielles) sur un menu contenant les entrées "DIVG", "EXIT" et une entrée par variable (ce qui permet de leur donner des valeurs, de les purger, etc...)

Ceci permet, par un appui sur "DIVG", de calculer $\text{div}(E)$ en des point précis, ou bien d'obtenir son expression symbolique.

Un appui sur "EXIT" permet de sortir du programme.

'DIVRG': (Checksum: # 8422d, Taille: 196.5 octets)

```

«   DUP   PURGE   →   f   v
   «   0   1   f   SIZE   FOR   i
       f   i   GET   v   i   GET   ð   +
   NEXT   'f'   STO
       { { "DIVG"   « f EVAL » } }   v   +
       { { "EXIT"   « CONT » } }   +
       TMENU   HALT   v   PURGE   2   MENU
   »
»

```

PROGRAMME 'DIVRG': EXEMPLES D'UTILISATION.

=====

Exemple 1:

Calculer la divergence du champ $E(X,Y) = (\cos(Y/X), \exp(XY))$, en un point quelconque du plan (ici $P(X,Y) = \cos(Y/X)$, $Q(X,Y) = \exp(XY)$).

2:	{ 'COS(Y/X)' 'EXP(X*Y)' }
1:	{ X Y }

On crée la pile ci-dessus et on appelle 'DIVRG', qui est suspendu, au bout de deux secondes, avec affichage d'un menu comportant les entrées "DIVG", "X", "Y", "EXIT".

Un appui sur "DIVG", sans donner de valeur à X et à Y, donne, au niveau 1 de la pile:

1:	'-(SIN(Y/X)*-(Y/X^2))+X*EXP(X*Y)'
----	-----------------------------------

Ce qui signifie que:

$$\text{div}(E) = \frac{\partial P}{\partial X} + \frac{\partial Q}{\partial Y} = \frac{Y}{X^2} \sin\left(\frac{Y}{X}\right) + X \exp(XY).$$

Un appui sur "EXIT" permet de sortir du programme.

Exemple 2:

On veut calculer la divergence du champ

$E(X,Y,Z) = (\ln(X+Y), Y*Z^2, X/(YZ))$, au point (1,2,3).

(Ici $P = \ln(X+Y)$, $Q = Y*Z^2$ et $R = X/(YZ)$).

On crée la pile ci-dessous, et on appelle 'DIVRG'.

2:	{ 'LN(X+Y)' 'Y*Z^2' 'X/Y/Z' }
1:	{ X Y Z }

'DIVRG' est alors suspendu avec présentation d'un menu contenant les entrées "DIVG", "X", "Y", "Z", et "EXIT".

On donne à X, Y, Z, les valeurs respectives 1, 2, 3 (par exemple, placer 3 au niveau 1, appuyer sur la touche "Z", puis STO).

On appuie sur "DIVG" : le résultat est immédiat.

On trouve

1:	9.2777777777
----	--------------

ce qui signifie qu'au point (1,2,3):

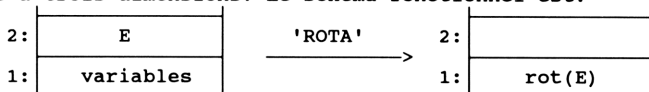
$$\text{Div}(E) = \frac{\partial P}{\partial X}(1,2,3) + \frac{\partial Q}{\partial Y}(1,2,3) + \frac{\partial R}{\partial Z}(1,2,3) = 167 / 18.$$

Un appui sur "EXIT" permet de sortir du programme.

ROTATIONNEL D'UN CHAMP DE VECTEURS.

=====

'ROTA' calcule le rotationnel $\text{rot}(E)$ d'un champ de vecteurs E de l'espace à trois dimensions. Le schéma fonctionnel est:



Le champ E doit ici être représenté par la liste $\{ P, Q, R \}$ de ses composantes, chacune d'elles étant une expression algébrique.

"variables" désigne la liste des trois variables par rapport auxquelles doivent être calculées les dérivées partielles de P, Q, R .

$\text{rot}(E)$ est une liste de 3 expressions algébriques représentant les composantes du rotationnel de E , c'est à dire du champ de vecteurs (Si P, Q, R sont les composantes de E , et si X, Y, Z sont les trois variables):

$$\left(\frac{\partial R}{\partial Y} - \frac{\partial Q}{\partial Z}, \frac{\partial P}{\partial Z} - \frac{\partial R}{\partial X}, \frac{\partial Q}{\partial X} - \frac{\partial P}{\partial Y} \right)$$

N.B: Le programme 'ROTA' est suspendu avec présentation d'un menu contenant les entrées "ROTA", "EXIT" (pour sortir) et une entrée par variable (ce qui permet éventuellement de leur donner des valeurs).

On peut ainsi obtenir (par un appui sur "ROTA"), l'expression symbolique du rotationnel, ou sa valeur en un point.

```
'ROTA': ( Checksum: # 10807d, Taille: 337 octets )
  " IF DUP SIZE 3 == THEN DUP PURGE 0 → f v j
    " 1 3 FOR i
      IF i 3 == THEN 1 ELSE i 1 + END
      'j' STO f j GET v i GET 0
      f i GET v j GET 0 -
    NEXT
    3 ROLL 3 →LIST 'f' STO
    { { "ROTA"
      " f EVAL 1 3 START ROT EVAL NEXT
        IFERR 3 →ARRY THEN →LIST END
      } }
    v + { { "EXIT" " CONT " } } +
    TMENU HALT v PURGE 2 MENU
  " END
"
```

Remarque:

Si le calcul du rotationnel a été demandé en un point précis, le résultat est obtenu sous forme d'un vecteur à trois composantes.

Sinon, et donc si les composantes de $\text{rot}(E)$ sont obtenues sous forme symbolique, le résultat est obtenu sous forme d'une liste à trois éléments.

PROGRAMME 'ROTA': EXEMPLE D'UTILISATION.

=====

Exemple :

On veut calculer le rotationnel du champ de vecteurs:

$$E(X,Y,Z) = (XY, YZ, ZX).$$

On crée la pile ci-dessous:

2:	{ 'X*Y' 'Y*Z' 'Z*X' }
1:	{ X Y Z }

On appelle ensuite le programme 'ROTA'.

Celui-ci est alors suspendu sur un menu contenant les entrées
"ROTA", "X", "Y", "Z".

- 1) si on appuie sur "ROTA" sans donner de valeur à X,Y,Z, le résultat est:

1:	{ '-Z' '-X' '-Y' }
----	--------------------

En effet l'expression du rotationnel de E en un point quelconque
(X,Y,Z) est: $\text{rot}(E) = (-Z, -X, -Y)$.

- 2) si on appuie sur "ROTA" après avoir uniquement donné à X la valeur 1, le résultat obtenu est:

1:	{ '-Z' -1 '-Y' }
----	------------------

On obtient ainsi l'expression du rotationnel de E en tout point
du type (1,Y,Z).

- 3) si on appuie sur "ROTA" après avoir donné à X,Y,Z les valeurs respectives 1,2 et 3, le résultat obtenu est:

1:	[-3 -1 -2]
----	--------------

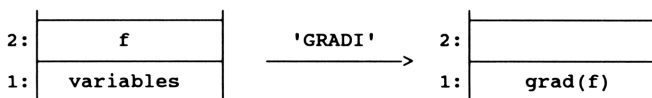
On obtient ainsi la valeur du rotationnel de E au point (1,2,3).

Un appui sur la touche "EXIT" permet de quitter le programme.

GRADIENT D'UNE FONCTION DE PLUSIEURS VARIABLES.

=====

'GRADI' calcule le gradient d'une fonction f de plusieurs variables. Le schéma fonctionnel est:



Ici "f" est une expression algébrique représentant la fonction f.

"variables" est la liste des variables par rapport auxquelles il faut calculer les dérivées partielles.

"grad(f)" est la liste des composantes du vecteur gradient de f (les composantes sont des expressions symboliques). Dans le cas où l'utilisateur a demandé le calcul en un point particulier, le résultat est un vecteur.

Rappelons que si par exemple f est une fonction des trois variables X,Y,Z, alors grad(f) est le vecteur:

$$\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right).$$

N.B: Le programme 'GRADI' est suspendu avec présentation d'un menu contenant les entrées "GRADI", "EXIT" (pour sortir) et une entrée par variable (ce qui permet éventuellement de leur donner des valeurs).

On peut ainsi obtenir (par un appui sur "GRADI"), l'expression symbolique du gradient, ou sa valeur en un point.

```
'GRADI': ( Checksum: # 44516d, Taille: 245 octets )
  « DUP PURGE DUP SIZE → f v n
    « 1 n FOR i f v i GET ð NEXT
      n →LIST 'f' STO
      { { "GRADI"
        « 1 n FOR i f i GET EVAL NEXT
          IFERR n →ARRY THEN →LIST END
        » } }
      v +
      { { "EXIT" « CONT » } } +
      TMENU HALT v PURGE 2 MENU
    »
  »
```

PROGRAMME 'GRAD': EXEMPLES D'UTILISATION.

=====

Exemple 1 :On veut calculer le gradient de $f(X,Y) = \exp(XY) \sin(Y)$.

On crée la pile ci-dessous:

2:	'EXP(X*Y)*SIN(Y)'
1:	{ X Y }

On appelle ensuite le programme 'GRADI'. Celui-ci est alors suspendu, avec présentation d'un menu personnalisé contenant les entrées "GRADI", "X", "Y", "EXIT".

1) si on appuie sur "GRADI" sans donner de valeur à X,Y,Z, le résultat est (en 3 secondes):

1: { 'Y*EXP(X*Y)*SIN(Y)' 'X*EXP(X*Y)*SIN(Y)+EXP(X*Y)*COS(Y)' }

c'est l'expression du gradient de f en un point quelconque (X,Y).

2) si on appuie sur "GRADI" après avoir donné à X,Y les valeurs respectives 1,2 le résultat obtenu est (en MODE 5 FIX) :

1: [13.43770 3.64392]

On obtient ainsi la valeur du gradient de f au point (1,2).
Un appui sur "EXIT" permet de sortir du programme.

Exemple 2 :On veut calculer le gradient de $f(X,Y,Z) = 'XY + Z^2 + Y/Z'$.

On crée la pile ci-dessous:

2:	'X*Y+Z^2+Y/Z'
1:	{ X Y Z }

On appelle ensuite le programme 'GRADI'. Celui-ci est alors suspendu, avec présentation d'un menu contenant les entrées "GRADI", "X", "Y" et "Z".

1) si on appuie sur "GRADI" sans donner de valeur à X,Y,Z, le résultat est (en 2 secondes):

1: { Y 'X+1/Z' '2*Z-Y/Z^2' }

c'est l'expression de grad(f) en un point quelconque (X,Y,Z).

2) si on appuie sur "GRADI" après avoir donné à X,Y les valeurs respectives 1,2 le résultat obtenu est :

1: { 2 '1+1/Z' '2*Z-2/Z^2' }

On obtient ainsi l'expression du gradient de f en tout point de coordonnées (1,2,Z).

LAPLACIEN D'UNE FONCTION DE PLUSIEURS VARIABLES.

=====

'DELTA' calcule le laplacien d'une fonction f de plusieurs variables. Le schéma fonctionnel est:



Ici "f" est une expression algébrique représentant la fonction f.

"variables" est la liste des variables par rapport auxquelles il faut calculer les dérivées partielles.

"Δ(f)" est l'expression du laplacien de f. Dans le cas où l'utilisateur a demandé le calcul en un point particulier, le résultat est un réel.

Rappelons que si par exemple f est une fonction des trois variables x,y,z, alors Δ(f) est le scalaire:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} .$$

N.B: Le programme 'DELTA' est suspendu avec présentation d'un menu contenant les entrées "LAPL", "EXIT" (pour sortir) et une entrée par variable (ce qui permet éventuellement de leur donner des valeurs).

On peut ainsi obtenir (par un appui sur "LAPL"), l'expression symbolique du laplacien, ou sa valeur en un point.

'DELTA': (Checksum: # 35128d, Taille: 199.5 octets)

```

«  DUP  PURGE  →  f  v
  «  0  1  v  SIZE  FOR  i
    f  1  2  START  v  i  GET  ð  NEXT  +
  NEXT  'f'  STO
    {  {  "LAPL"  «  f  EVAL  »  }  }
    v  +
    {  {  "EXIT"  «  CONT  »  }  }  +
    TMENU  HALT  v  PURGE  2  MENU
  »
»

```

PROGRAMME 'DELTA': EXEMPLES D'UTILISATION.

=====

Exemple 1_:On veut calculer le laplacien de $f(X,Y) = X^2 \cdot \sin(Y)$.

On crée la pile ci-dessous:

2:	$X^2 \cdot \sin(Y)$	On appelle ensuite le programme 'DELTA'. Celui-ci est suspendu avec présentation d'un menu personnalisé contenant les entrées "LAPL", "X", "Y", et "EXIT".
1:	{ X Y }	

- 1) si on appuie sur "LAPL" sans donner de valeur à X,Y, le résultat est :

1: $2 \cdot \sin(Y) + X^2 \cdot -\sin(Y)$

(expression du laplacien de f en un point quelconque (X,Y)).

- 2) si on appuie sur "LAPL" après avoir donné à X,Y les valeurs respectives 1,2 le résultat est :

1: .909297426824

On obtient ainsi la valeur du laplacien de f au point (1,2).

Un appui sur "EXIT" permet alors de sortir du programme.

Exemple 2 :On veut calculer le laplacien de $f(X,Y,Z) = XY + Z^2 + Y/Z$.

On crée la pile ci-dessous:

2:	$X \cdot Y + Z^2 + Y/Z$	On appelle ensuite le programme 'DELTA' qui est suspendu, avec présentation d'un menu personnalisé contenant les entrées: "LAPL", "X", "Y", "Z" et "EXIT".
1:	{ X Y Z }	

- 1) si on appuie sur "LAPL" sans donner de valeur à X,Y,Z, le résultat est :

1: $2 + Y \cdot (2 \cdot Z) / Z^2 \cdot 2$

car l'expression de $\Delta(f)$ en un point quelconque (X,Y,Z) est:
 $2 + 2 \cdot Y / Z^3$. (pour le voir utiliser COLCT et EXPAN).

- 2) si on appuie sur "LAPL" après avoir donné à Z la valeur 1, on trouve l'expression du laplacien de f en tout point de coordonnées (X, Y, 1), c'est-à-dire: $2 + Y \cdot 2$.

Un appui sur "EXIT" permet de sortir du programme.

DIFFERENTIELLE D'UNE FONCTION DE PLUSIEURS VARIABLES.

=====

'DIFF' calcule la différentielle d'une fonction f de plusieurs variables. Le schéma fonctionnel est:



Ici " f " est une expression algébrique représentant la fonction f .

"variables" est la liste des variables par rapport auxquelles il faut calculer les dérivées partielles.

"df" est l'expression de la différentielle de f .

Rappelons que si par exemple f est une fonction des trois variables X, Y, Z , alors df s'écrit:

$$df = \frac{\partial f}{\partial X} dX + \frac{\partial f}{\partial Y} dY + \frac{\partial f}{\partial Z} dZ.$$

N.B: Le programme 'DIFF' est suspendu avec présentation d'un menu contenant les entrées "DIFF", "EXIT" (pour sortir) et une entrée par variable (ce qui permet éventuellement de leur donner des valeurs).

On peut ainsi obtenir (par un appui sur "DIFF"), l'expression symbolique de la différentielle, ou sa valeur en un point.

```
'DIFF': ( Checksum: # 37567d, Taille: 231 octets )
"  DUP    PURGE    →    f    v
"  " 0 1 v SIZE FOR i
    f v i GET 0 "d" v i GET →STR
    2 OVER SIZE 1 - SUB + OBJ→ * +
NEXT
'f' STO
{ { "DIFF" " f EVAL " } }
v +
{ { "EXIT" " CONT " } } +
TMENU HALT v PURGE 2 MENU
"
"
```

N.B: Si les variables s'appellent X, Y, Z (par exemple), le programme 'DIFF' utilise les noms 'dx', 'dy', 'dz'. Pour que 'DIFF' fonctionne correctement, il faut qu'aucun d'eux ne soit le nom d'une variable du répertoire.

PROGRAMME 'DIFF': EXEMPLES D'UTILISATION.

=====

Exemple 1 :

On veut calculer la différentielle de $f(X,Y) = X^2 \cdot \sin(Y)$.
On crée la pile ci-dessous:

2:	'X^2*SIN(Y)'	On appelle ensuite le programme 'DIFF'. Celui-ci est suspendu, avec présentation d'un menu personnalisé contenant les entrées "DIFF", "X", "Y" et "EXIT".
1:	{ X Y }	

- 1) si on appuie sur "DIFF" sans donner de valeur à X,Y, le
résultat est :

1: '2*X*SIN(Y)*dX+X^2*COS(Y)*dY'

(expression de df en un point quelconque (X,Y)).

- 2) si on appuie sur "DIFF" après avoir donné à X,Y les valeurs
respectives 1,2 le résultat obtenu est (en mode 5 FIX):

1: '1.81859*dX-0.41615*dY'

On obtient ainsi la valeur de df au point (1,2).

Un appui sur "EXIT" permet de sortir du programme.

Exemple 2 :

On veut calculer la différentielle de
 $f(RO,TETA,FI) = RO^2 \cdot \sin(TETA) \cdot \exp(FI \cdot TETA)$.
On crée la pile ci-dessous:

2:	'RO^2*SIN(TETA)*EXP(FI*TETA)'
1:	{ RO TETA FI }

On appelle ensuite 'DIFF'. Celui-ci est suspendu sur un menu
contenant les entrées "DIFF", "RO", "TETA", "FI" et "EXIT".

Si on donne à RO, TETA, et FI les valeurs respectives 1, 2 et 3,
et si on appuie ensuite sur "DIFF" on trouve (en mode 2 FIX):

1: '733.67*dRO+932.62*dTETA+733.67*dFI'

ce qui est l'expression de df au point (1,2,3).

Un appui sur "EXIT" permet de sortir du programme.

GRAPHISME

La HP48SX possède un écran graphique de 131*64 points. Un grand nombre d'instructions se rapportent à la gestion de cet écran; on les trouve regroupées dans les menus **PLOT**, ou **PGR DSPL**, ou dans l'environnement **GRAPH**.

Dans le domaine du graphisme, et par rapport à la HP28S, la HP48SX marque un progrès considérable. A tel point d'ailleurs que les programmes de ce chapitre, qui à l'origine s'appliquaient à la HP28S, sont aujourd'hui moins indispensables.

Néanmoins j'ai conservé, en les modifiant au besoin, les programmes suivants:

- 'PAR' : tracé d'une courbe paramétrée ($X=X(T)$, $Y=Y(T)$).
- 'POL' : tracé d'une courbe en polaires ($R_o = R_o(\theta)$).
- 'POLP' : tracé d'une courbe en polaires paramétrées ($R_o = R_o(T)$, $\theta = \theta(T)$).
- 'TRACE' : module de tracé utilisé par 'PAR', 'POL' et 'POLP'.
- 'ENV' : tracé de l'enveloppe d'une famille de droites.
- 'MTCL' : utilisation de la méthode de Monte-Carlo pour visualiser des courbes implicites $F(X,Y)=0$.
- 'FANT' : tracé d'une famille de courbes dépendant de T.
- 'ANIM' : production d'images-écran successives.

Les programmes 'PAR', 'POL', 'POLP', 'ENV' ont pour intérêt de tracer des courbes tout en mémorisant les points utilisés dans la matrice EDAT, ce que ne permet pas l'instruction DRAW et ce qui est utile si vous devez tracer ces courbes sur feuille.

J'ai hésité à conserver le programme 'MTCL'. Il est en effet possible de visualiser des courbes $F(X,Y)=0$ en traçant l'expression ' $F(X,Y)>0$ ' (ou ' $F(X,Y)<0$ ') avec l'instruction DRAW (et après avoir défini "TRUTH" comme type de tracé).

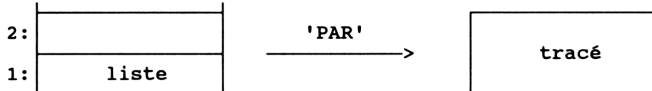
Néanmoins 'MTCL' peut s'avérer plus rapide, car il permet de tester une partie seulement des points de l'écran (et non pas la totalité comme avec le type de tracé "TRUTH").

TRACE D'UNE COURBE PARAMETREE.

=====

'PAR' permet de tracer une courbe définie par une représentation paramétrique : $X=X(T)$, $Y=Y(T)$.

Le schéma fonctionnel est le suivant:



La liste du niveau 1 a le format { M(T) debut fin pas } où :

- * M(T) est une expression algébrique, dont la valeur est le nombre complexe indiquant les coordonnées (X(t),Y(t)) du point courant (la majuscule T est obligatoire).
- * "debut", "fin" représentent respectivement la valeur initiale et la valeur finale prises par le paramètre T.
- * "pas" est l'incrément du paramètre T. Le temps de tracé est évidemment inversement proportionnel à T.

Le programme 'PAR' appelle le programme 'TRACE'.

En particulier:

Le tracé n'est pas interrompu si l'évaluation d'un point est impossible: ce point est simplement ignoré.

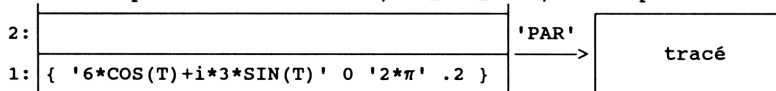
Les points trouvés sont placés dans la matrice EDAT (d'où une possibilité de les relire, ou de les utiliser pour cadrer l'écran avec l'instruction SCATRPLOT).

A la fin du tracé, on est placé dans l'environnement 'GRAPH'. On sort du programme par un appui sur 'ON'.

'PAR': (Checksum: # 3906d, Taille: 38.5 octets)

“ “ ” TRACE ”

Exemple: tracé de la courbe $X(T)=6*\cos(T)$, $Y(T)=3*\sin(T)$, avec les valeurs par défaut dans PPAR, $0 \leq T \leq 2*\pi$, et un pas de 0.2



C'est une ellipse. Elle est obtenue en 25 secondes.

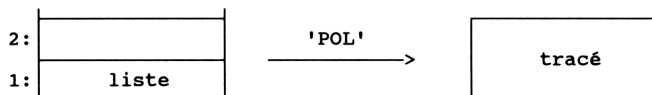
A la sortie la matrice EDAT contient les 32 points utilisés.

TRACE D'UNE COURBE EN POLAIRES.

=====

'POL' permet de tracer une courbe définie par une représentation en coordonnées polaires: $Ro = Ro(T)$.

Le schéma fonctionnel est le suivant:



La liste du niveau 1 a le format { $Ro(T)$ debut fin pas } où

- * $Ro(T)$ est l'expression algébrique représentant le rayon-vecteur du point courant, d'angle polaire T (la majuscule T est obligatoire).
- * "debut", "fin" représentent respectivement la valeur initiale et la valeur finale prises par l'angle polaire T .
- * "pas" est l'incrément de l'angle polaire T . Le temps de tracé est évidemment inversement proportionnel à T .

Le programme 'POL' appelle le programme 'TRACE'.

En particulier:

Le tracé n'est pas interrompu si l'évaluation d'un point est impossible: ce point est simplement ignoré.

Les points trouvés sont placés dans la matrice EDAT (d'où une possibilité de les relire, ou de les utiliser pour cadrer l'écran avec l'instruction SCATRPLOT).

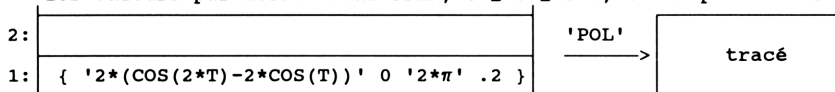
A la fin du tracé, on est placé dans l'environnement 'GRAPH'.

On sort du programme par un appui sur 'ON'.

'POL': (Checksum: # 6067d, Taille: 53 octets)

“ “ T i * EXP * ” TRACE ”

Exemple: tracé de la courbe $Ro(\theta) = 2 * (\cos(2*\theta) - 2*\cos(\theta))$, avec les valeurs par défaut dans PPAR, $0 \leq \theta \leq 2*\pi$, et un pas de 0.2



La courbe est obtenue en 25 secondes.

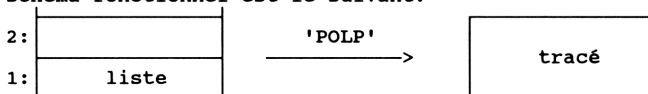
A la sortie la matrice EDAT contient les 32 points utilisés.

TRACE D'UNE COURBE EN POLAIRES PARAMETREES.

=====

'POLP' permet de tracer une courbe définie par une représentation en polaires paramétrées : $R_o = R_o(T)$, $\theta = \theta(T)$.

Le schéma fonctionnel est le suivant:



La liste du niveau 1 a le format { M(T) debut fin pas } où

- * M(T) est une expression algébrique dont la valeur est le nombre complexe représentant un couple de coordonnées polaires ($R_o(T), \theta(T)$) du point courant (la majuscule T est obligatoire).
- * "debut", "fin" représentent respectivement la valeur initiale et la valeur finale prises par le paramètre T.
- * "pas" est l'incrément du paramètre T. Le temps de tracé est évidemment inversement proportionnel à T.

Le programme 'POLP' appelle 'TRACE'.

En particulier:

Le tracé n'est pas interrompu si l'évaluation d'un point est impossible: ce point est simplement ignoré.

Les points trouvés sont placés dans la matrice Σ DAT (d'où une possibilité de les relire, ou de les utiliser pour cadrer l'écran avec l'instruction SCATRPLOT).

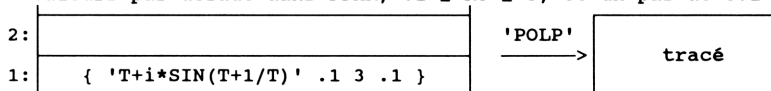
A la fin du tracé, on est placé dans l'environnement 'GRAPH'.

On sort du programme par un appui sur 'ON'.

'POLP': (Checksum: # 17671d, Taille: 52 octets)

« « C+R i * EXP * » TRACE »

Exemple: tracé de la courbe définie par $\theta = \sin(R_o + 1/R_o)$, avec les valeurs par défaut dans PPAR, $.1 \leq R_o \leq 3$, et un pas de 0.1



Le tracé de la courbe est obtenu en 30 secondes.

A la sortie la matrice Σ DAT contient les 30 points utilisés.

MODULE DE TRACE UTILISE PAR 'PAR', 'POL', et 'POLP'.

=====

Les programmes 'PAR', 'POL', 'POLP' utilisent le même programme 'TRACE' que voici:

'TRACE': (Checksum: # 13063d, Taille: 258.5 octets)

```

«   SWAP  OBJ→  DROP  CLE
    ERASE  { #0 #0 }  PVIEW  DRAX  0
    → sp m d f p v
    «   d →NUM  f →NUM  FOR i
        i 'T'  STO
        IFERR
            m  EVAL  sp  EVAL
            DUP  C→R  2  →ARRY  E+
            DUP
            IF  v  0  ≠
                THEN  v  SWAP  LINE
                ELSE  PIXOM
            END
            'v'  STO
        THEN
            CLEAR
        END
    p  STEP
    'T'  PURGE  GRAPH
    »
»

```

Indications:

La variable 'v' (v pour "vieux") contient le point précédent (utile pour tracer les segments). Au départ, on met 0 dans 'v', et le test "IF v 0 ≠" teste si on n'est pas au premier point. (Ce n'est qu'à partir du 2ième point qu'on trace des segments).

Les différents points (les extrémités des segments) sont placés dans la matrice ΣDAT, qui au départ est purgée.

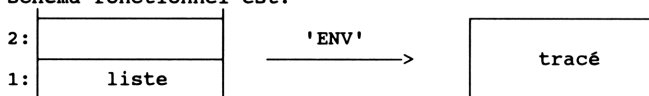
TRACE DE L'ENVELOPPE D'UNE FAMILLE DE DROITES.

=====

'ENV' permet de tracer l'enveloppe d'une famille de droites D(T),
l'équation de D(T) s'écrivant:

$$A(T)*X + B(T)*Y + C(T) = 0.$$

Le schéma fonctionnel est:



La liste du niveau 1 a le format suivant:

- { A(T) B(T) C(T) debut fin pas } avec:
- * A(T), B(T), C(T): expressions algébriques de la variable T, intervenant dans l'équation de la droite D(T). La majuscule T est obligatoire.
- * "debut", "fin" représentent respectivement la valeur initiale et la valeur finale prises par le paramètre T.
- * "pas" est l'incrément du paramètre T. Le temps de tracé est évidemment inversement proportionnel à T.

Le tracé n'est pas interrompu par une erreur éventuelle sur le calcul d'un point de la courbe. Ce point est simplement ignoré.

Le programme 'ENV' appelle le programme 'PAR', qui lui même appelle le programme 'TRACE'.

En particulier les différents points utilisés dans le tracé sont conservés dans la matrice ΣDAT.

'ENV': (Checksum: # 3853d, Taille: 272.5 octets)

```

"   'T'   PURGE   EVAL
→   a   b   c   d   f   p
"   a   'T'   d   b   'T'   d   c   'T'   d
→   a1  b1  c1
"   c1  b   *   c   b1   *   -   a   b1   *   a1  b   *
    -   DUP   ROT   ROT   /   SWAP   a1  c   *   a   c1
    *   -   SWAP   /   i   *   +   d   f   p   4   →LIST   PAR
"
"
"

```

Exemple: Pour tracer l'enveloppe de la famille de droites d'équations:
 $SIN(T)*X + COS(T)*Y - 3*SIN(T)*COS(T) = 0$, on place au niveau 1 la
 liste: { 'SIN(T)' 'COS(T)' '-3*SIN(T)*COS(T)' 0 '2*π' .1 }
 (On obtient une astéroïde, en 1 mn 40 s).

METHODE DE MONTE-CARLO (VISUALISER LES COURBES $F(X,Y)=0$)

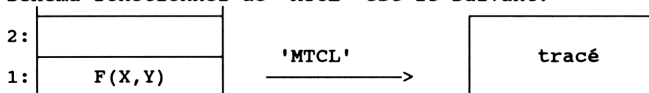
=====

On veut visualiser une courbe définie par une équation $F(X,Y)=0$.

Le principe de la méthode de Monte-Carlo, mis en oeuvre ici par le programme 'MTCL', est de tester le signe de F en un certain nombre de points (en principe uniformément répartis), d'"allumer" les points où F est positive, et de laisser les points où F est négative.

On obtient ainsi une visualisation des régions du plan délimitées par la courbe $F(X,Y)=0$, et par là même on a une idée de cette courbe.

Le schéma fonctionnel de 'MTCL' est le suivant:



où $F(X,Y)$ est l'expression algébrique de l'application F .
(les majuscules X et Y sont obligatoires).

Le programme 'MTCL' utilise un "pas" h. Par défaut, h=1.

h=1 signifie que tous les pixels sont passés en revue.

h=2 signifie qu'une ligne et une colonne sur 2 sont concernées.
etc...

En cours de tracé, l'appui sur la touche + augmente h de 1, et l'appui sur la touche - diminue h de 1 (h restant cependant toujours compris entre 1 et 8).

'MTCL': (Checksum: # 64594d, Taille: 449.5 octets)

```

" 1 → f h
  " ERASE { #0 #0 } PVIEW DRAX PICT SIZE
    PPAR 2 GET PPAR 1 GET DUP2
    - C→R 5 ROLL B→R / SWAP 5 ROLL B→R /
    → pmax pmin pasy pasx
    " pmin RE pmax RE FOR i i 'X' STO
      pmin IM pmax IM FOR j j 'Y' STO
      f EVAL
      IF 0 > THEN i j R→C PIXON END
      IF KEY THEN
        { 85 95 } SWAP POS 1 + { 0 -1 1 }
        SWAP GET h + 1 MAX 8 MIN 'h' STO
      END
      pasy h * STEP
      pasx h * STEP
      { X Y } PURGE GRAPH
    "
  "
"

```

Remarque:

A la fin du tracé, on passe dans l'environnement GRAPH.

On en sort par un appui sur 'ON'.

PRODUCTION D'IMAGES-ECRAN SUCCESSIVES.

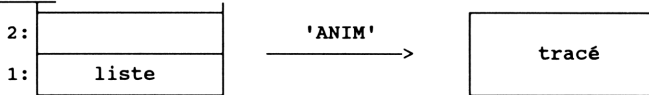
=====

'ANIM' permet de visualiser successivement différentes images-écran .

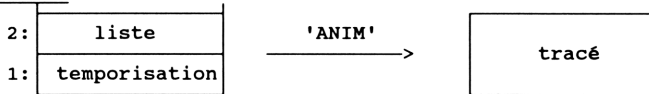
Ces différentes images doivent être représentées par des objets graphiques, regroupés dans une liste.

Le schéma fonctionnel est le suivant:

Premier cas:



Deuxième cas:



Si la liste a été stockée dans une variable, il est possible d'utiliser ce nom de variable. Les différentes images sont envoyées successivement à l'écran, dans l'ordre de leur apparition dans la liste.

Quand la dernière image a été affichée, le processus recommence au départ.

On sort du programme 'ANIM' par un appui sur la touche 'ON'.

Dans le premier cas:

Le programme est suspendu quand une image est affichée. 'ANIM' attend alors que l'utilisateur appuie sur une touche quelconque (sauf 'ON' qui interromprait le programme) pour passer à l'affichage suivant.

Dans le deuxième cas:

"temporisation" est un réel positif, représentant la durée en secondes, entre deux affichages successifs. Le défilement des images à l'écran est alors automatique (Le réel 0 ramène au premier cas).

```
'ANIM': ( Checksum: # 57620d, Taille: 99.5 octets )
«   IF  DUP   TYPE  THEN  0  ELSE  ABS  END
    →  t
    «   { 1 }  CLLCD
        DO
            GETI   →LCD  t   WAIT
            IF  t   NOT   THEN  DROP  END
            UNTIL  0   END
    »
»
```


ENTIERS LONGS

Le répertoire 'LONG' contient un ensemble de programmes permettant de manipuler des 'entiers longs'. Sous ce terme, on entend ici des entiers positifs ou nuls dont le nombre de chiffres peut être supérieur aux douze chiffres significatifs que le calculateur peut garder en mémoire et afficher.

Ici, un entier long n est représenté par un vecteur dont les composantes forment la décomposition de n en base 10^5 . Ces composantes doivent donc être des entiers compris entre 0 et 99999.

Par exemple: $N = 165088696783290882115695$ s'écrit
[1650 88696 78329 8821 15695].

Inversement:
[87 132 6 78999 1] représente $N = 8700132000067899900001$.
Plus simplement 1 s'écrit [1], et [1 0] est égal à $100000 = 10^5$.

Le répertoire 'LONG' contient les programmes permettant d'effectuer les opérations courantes sur les entiers longs:

- 'ADDL' : addition de deux entiers longs.
- 'PRODL' : produit de deux entiers longs.
- 'EXPOL' : puissances entières d'un entier long.
- 'DIVL' : division de 2 entiers longs (calcul du quotient et du reste entiers).
- 'PGDL' : pgcd de deux entiers longs.
- 'PPML' : ppcm de deux entiers longs.
- 'FACTL' : obtention, sous forme d'un entier long, de la factorielle d'un entier naturel.

On trouve également dans le répertoire 'LONG', les routines qui permettent de passer de telle à telle forme de l'écriture d'un entier long:

- 'L→CH' : écriture d'un entier long sous la forme d'une chaîne de caractères.
- 'CH→L' : traduction d'une chaîne de caractères sous forme d'un entier long.
- 'R→L' : passage de la forme "réel" à la forme entier long.
- 'L→R' : passage de la forme entier long à la forme "réel".

On trouve enfin deux routines dont le rôle est de permettre un bon fonctionnement des programmes cités plus hauts, mais que l'utilisateur n'a en principe pas à utiliser directement:

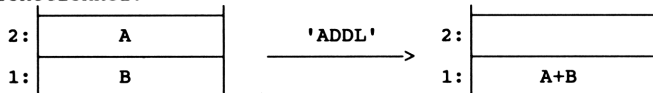
- 'FRMT' : gestion des débordements dans les calculs sur les entiers longs.
- 'ELMG' : dans un entier long, élimination des éventuels coefficients nuls à gauche.

Remarque: grâce à la façon dont sont représentés les entiers longs et les polynômes, il se trouve qu'un certain nombre de programmes sur les entiers longs sont en fait des appels à des programmes analogues sur des polynômes (avec donc passage dans le répertoire 'POLY', puis retour dans le répertoire 'LONG'). C'est le cas des programmes 'ADDL', 'PRODL' et 'ELMG'.

ADDITION DE DEUX ENTIERS LONGS.

=====

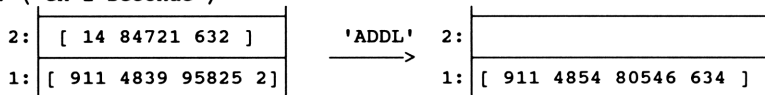
'ADDL' permet d'additionner deux entiers longs A et B, suivant le schéma fonctionnel:



Le résultat est obtenu dans le format 'entier long'. Le programme 'ADDL' passe dans le répertoire 'POLY' où il appelle le programme 'ADDP'. Il revient dans le répertoire 'LONG', où il appelle le programme 'FRMT' (gestion des débordements).

'ADDL': (Checksum: # 63368d, Taille: 48.5 octets)
" POLY ADDP LONG FRMT "

Exemple: (en 1 seconde)

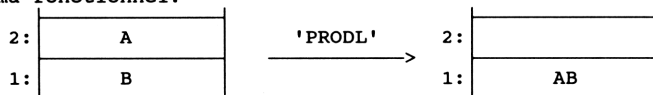


car 14 84721 00632 + 911 04839 95825 00002 = 911 04854 80546 00634 .

PRODUIT DE DEUX ENTIERS LONGS.

=====

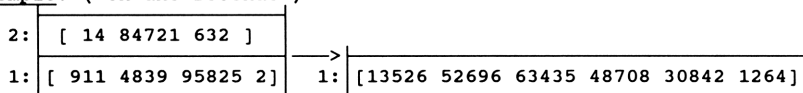
'PRODL' permet de multiplier deux entiers longs A et B, suivant le schéma fonctionnel:



Le résultat est obtenu dans le format 'entier long'. Le programme 'PRODL' passe dans le répertoire 'POLY' où il appelle le programme 'PRODP'. Il revient dans le répertoire 'LONG', où il appelle le programme 'FRMT' (gestion des débordements).

'PRODL': (Checksum: # 40829d, Taille: 50.5 octets)
" POLY PRODP LONG FRMT "

Exemple: (en une seconde)



car 14 84721 00632 * 911 04839 95825 00002 est égal à:
13526 52696 63435 48708 30842 01264.

ELEVATION D'UN ENTIER LONG A UNE PUISSANCE ENTIERE.

=====

'EXPOL' permet de calculer A^n , où A est un entier long, et où n est un entier naturel.

Le schéma fonctionnel est:



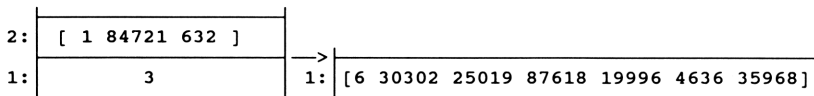
N.B: 'EXPOL' appelle le programme 'PRODL'.

'EXPOL': (Checksum: # 51318d, Taille: 169 octets)

```

«   →   a   n
    «   1 DUP →ARRY
        WHILE n 0 >
        REPEAT
            IF n 2 MOD THEN
                a PRODL
            END
            n 2 / FLOOR 'n' STO
            IF n THEN
                a DUP PRODL 'a' STO
            END
        END
    »
  
```

Exemple: (en 3 secondes)

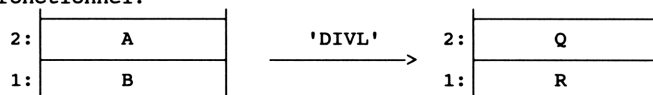


car (1 84721 00632)³ = 6 30302 25019 87618 19996 04636 35968.

DIVISION DE DEUX ENTIERS LONGS.

=====

'DIVL' permet de diviser deux entiers longs A et B, suivant le schéma fonctionnel:



où Q et R sont respectivement le quotient et le reste entiers dans cette division ($A = BQ + R$ avec $R < B$), tous les deux obtenus dans le format entier long.

N.B: 'DIVL' appelle les programmes 'ELMG' et 'ADDL'.

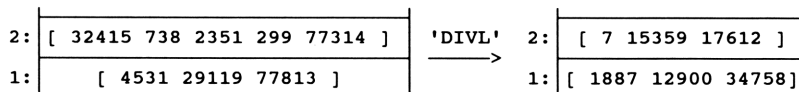
'DIVL': (Checksum: # 46375d, Taille: 411 octets)

```

«  ELMG  DUP 1 GET 1 + OVER SIZE 1 GET 0
   → b d tb q
   « 0 1 →ARRY SWAP
      WHILE
        DUP 1 GET d / FLOOR 'q' STO
        DUP SIZE 1 GET tb DUP2 > 3 ROLLD
        ==
        IF DUP q NOT AND THEN
          3 PICK b - ELMG
          1 GET 0 ≥ 'q' STO
        END
        q AND
        OR
      REPEAT
        IF q NOT THEN
          OBJ→ 1 GET 1 - →ARRY 1 OVER
          1 GET 4 ROLL 100000 * +
          DUP d / FLOOR 'q' STO PUT
        END
        q DUP 1 →ARRY 3 PICK SIZE 1 GET
        tb - 1 + 1 →LIST RDM
        4 ROLL ADDL 3 ROLLD
        b * NEG OVER SIZE RDM ADDL
      END
   »
»

```

Exemple: (en 9 secondes)



car 32415 00738 02351 00299 77314 est égal à
 (4531 29119 77813) * (7 15359 17612) + 1887 12900 34758.

PGCD DE DEUX ENTIERS LONGS.

=====

'PGDL' calcule le pgcd (plus grand commun diviseur) de deux entiers longs A et B. Le schéma fonctionnel est:



Le Pgcd est obtenu dans le format 'entier long'.

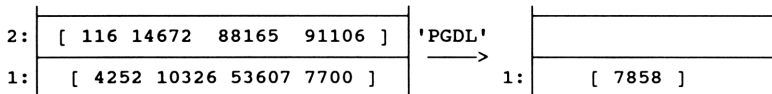
N.B: 'PGDL' utilise les programmes 'L→R', 'R→L', 'FRMT' et 'DIVL'.
 Il passe également dans le répertoire 'ARIT' où il appelle le programme 'PGCD' (pgcd de deux entiers).
 'PGDL' s'appelle lui-même.

'PGDL': (Checksum: # 25464d, Taille: 166.5 octets)

```

«   IF   DUP   ABS   THEN
      IF DUP2 SIZE 1 GET SWAP SIZE 1 GET MAX 3 <
      THEN
        L→R SWAP L→R ARIT PGCD LONG 1 →ARRY FRMT
      ELSE
        DUP 3 ROLL DIVL SWAP DROP PGDL
      END
    ELSE
      DROP
    END
»
  
```

Exemple: (en 31 secondes)

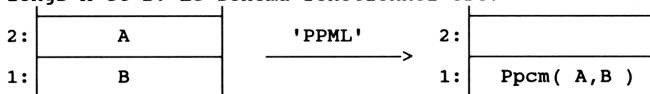


car le Pgcd de 116 14672 88165 91106 et de 4252 10326 53607 07700 est égal à 7858.

PPCM DE DEUX ENTIERS LONGS.

=====

'PPML' calcule le ppcm (plus petit commun multiple) de deux entiers longs A et B. Le schéma fonctionnel est:

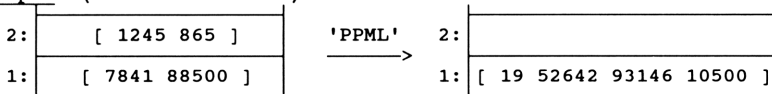


Le Ppcm est obtenu dans le format 'entier long'.

N.B: 'PPMP' appelle les programmes 'PGDL', 'DIVL' et 'PRODL'.

'PPML': (Checksum: # 12104d, Taille: 47 octets)
 « **DUP2** **PGDL** **DIVL** **DROP** **PRODL**
 »

Exemple: (en 20 secondes)

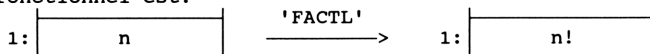


car ppcm (1245 00865, 7841 88500) = 19 52642 93146 10500.

FUNCTION FACTORIELLE.

=====

'FACTL' calcule la factorielle n! d'un entier naturel n. Le schéma fonctionnel est:



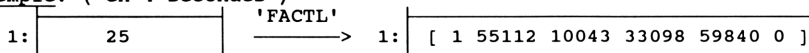
n doit être un entier naturel. Le résultat est obtenu dans le format 'entier long'.

On peut vérifier que la fonction "!" de la HP48 ne fournit tous les chiffres de n! que jusqu'à n=14 (et alors n! = 87178291200). Le rôle de 'FACTL' est d'obtenir tous les chiffres significatifs de n! quand n>14.

Le programme 'FACTL' s'appelle lui-même, et appelle 'FRMT'.

'FACTL': (Checksum: # 56148d, Taille: 121 octets)
 « → **n**
 « **IF** **n** **14** ≤ **THEN** **n** **!** **1** →ARRY **FRMT**
 ELSE **n** **n** **1** - **FACTL** * **FRMT** **END**
 »
 »

Exemple: (en 4 secondes)



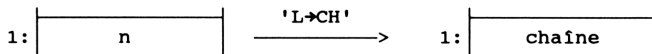
car 25! = 1 55112 10043 33098 59840 00000.

TRANSFORMATION D'UN ENTIER LONG EN CHAÎNE DE CARACTÈRES.

=====

'L→CH' transforme un entier long n en une chaîne de caractères représentant n, avec séparation par un espace tous les trois chiffres.

Le schéma fonctionnel est:

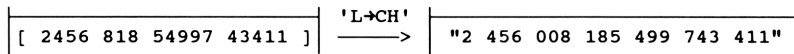


'L→CH': (Checksum: # 21764d, Taille: 233 octets)

```

«   ELMG   DUP    100000   CON   +   OBJ→   1   GET
    →      n
    «      ""      1      n      START
          SWAP    →STR    2      OVER    SIZE    SUB    SWAP    +
    NEXT
    WHILE  DUP    NUM    48    ==
          REPEAT  2      OVER    SIZE    SUB    END
    DUP    SIZE    3      -
    IF     DUP    0      >      THEN
      1     FOR    i
        DUP  1     i     SUB    " "    +    SWAP
        i    1     +     OVER    SIZE    SUB    +
      -3    STEP
    ELSE   DROP    END
  »
»
  
```

Exemple: (une seconde)

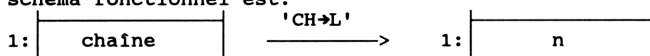


TRANSFORMATION D'UNE CHAÎNE DE CARACTÈRES EN ENTIER LONG.

=====

'CH→L' transforme une chaîne de caractères, exclusivement composée de chiffres ou d'espaces, en l'entier long n correspondant.

Le schéma fonctionnel est:



Le programme s'arrête sur un message d'erreur si aucun chiffre n'est trouvé, ou si un caractère non autorisé est rencontré.

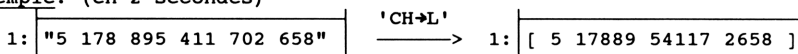
NB: dans le texte ci-dessous, la chaîne "0123456789 " se termine par un espace.

'CH→L': (Checksum: # 63609d, Taille: 365 octets)

```

«    DUP    SIZE    DEPTH
→   ch    n    p
«    1    n    FOR    i
      "0123456789 "    ch    i    i    SUB    POS
      IF    DUP    THEN
        IF    DUP    11    ==    THEN    DROP
        ELSE    1    -    END
      ELSE    "Erreur"    DOERR    END
NEXT
DEPTH    p    -    2    +    DUP    5    MOD    DUP2    -
→   n    r    m
«    IF    r    THEN
      r    4    START    0    n    1    +    ROLL    NEXT
    END
    m    1    +    m    5    /    1    +    FOR    k
    1    4    FOR    i
      SWAP    10    i    ^    *    +
    NEXT
    k    ROLL
  -4    STEP
  m    5    /    1    +    →ARRY
»
»
»
  
```

Exemple: (en 2 secondes)

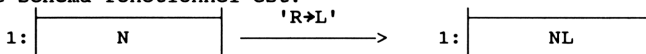


TRANSFORMATION D'UN REEL EN ENTIER LONG.

=====

'R→L' transforme un nombre N au format réel (en principe un entier) en son équivalent NL dans le format 'entier long'.

Le schéma fonctionnel est:



N.B: 'R→L' appelle le programme 'FRMT'.

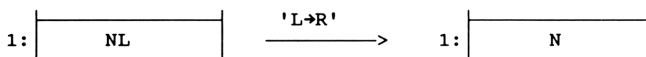
'R→L': (Checksum: # 7262d, Taille: 45.5 octets)
 « .5 + FLOOR 1 →ARRY FRMT »

Exemples: 1 est transformé en [1].
 71875488654 est transformé en [7 18754 88654].
 125487.7 est transformé en [1 25488].
 1.49865E30 est transformé en [1 49865 0 0 0 0 0].

TRANSFORMATION D'UN ENTIER LONG EN REEL.

=====

'L→R' transforme un entier long NL en son équivalent réel N. Le schéma fonctionnel est:



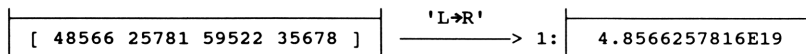
Evidemment, si NL représente un entier supérieur à 1E12, le passage au format réel s'accompagne d'une perte de précision. Néanmoins le programme 'L→R' est utile pour connaître rapidement l'ordre de grandeur d'un entier représenté sous le format 'entier long'.

N.B: le programme 'L→R' passe dans le répertoire 'POLY' où il appelle le programme 'VALP'.

'L→R': (Checksum: # 34456d, Taille: 50.5 octets)

« POLY 100000 VALP LONG »

Exemple: (en moins d'une seconde)

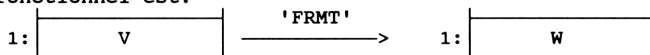


GESTION DES DEBORDEMENTS
SUR LES ENTIERS LONGS.

=====

Lorsqu'on effectue des opérations sur les vecteurs représentant des entiers longs (exemple une soustraction), il arrive que le résultat ne soit pas exactement au format entier long (composantes toutes positives et inférieures à 100000). Le rôle de 'FRMT' est de gérer ces 'débordements' et de mettre le vecteur au bon format.

'FRMT' est appelé par de nombreux programmes du répertoire 'LONG'. En principe l'utilisateur n'a pas à l'appeler directement. Le schéma fonctionnel est:



où W est le vecteur résultant éventuellement de la modification de V.

```

'FRMT': ( Checksum: # 42172d, Taille: 192.5 octets )
" 0 SWAP OBJ→ 1 GET
  → n
  " 1 n START
    DUP 100000 MOD DUP n 3 + ROLL - 100000 / +
  NEXT
  WHILE DUP 0 >
    REPEAT
      DUP 100000 MOD DUP 'n' 1 STO+
      n 2 + ROLL - 100000 /
    END
  DROP n →ARRY
"

```

Exemples:

[106622 110050 129366] est transformé en [1 6623 10051 29366].
[13902 -35682 -21383] est transformé en [13901 64317 78617].

ELIMINATION DES ZEROS A GAUCHE.

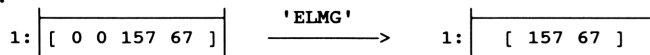
=====

'ELMG' élimine tous les coefficients nuls qui pourraient débiter un vecteur. Cette routine est utilisée par certains programmes du répertoire 'LONG'. Elle n'a en principe pas à être appelée directement par l'utilisateur. 'ELMG' se contente de passer dans le répertoire 'POLY' pour y appeler le programme du même nom.

```

'ELMG': ( Checksum: # 56099d, Taille: 41 octets )
" POLY ELMG LONG "

```

Exemple:

PROBABILITES

Le répertoire 'PROBA' contient un certain nombre de programmes permettant de connaître, sans avoir recours à des tables numériques (et donc sans avoir à y faire d'interpolations) les lois de probabilités et les fonctions de répartition des lois usuelles (discrètes ou continues).

Puisque les dénombrements interviennent fréquemment dans les calculs de probabilités, le répertoire 'PROBA' contient quelques petits programmes s'appliquant aux dénombrements.

Signalons également que certains programmes d'autres répertoires peuvent être utiles ici: en particulier 'SIMP' et 'CALC' du répertoire 'ARIT', et l'instruction $\rightarrow Q$ (il est souvent nécessaire de pouvoir donner la valeur d'une probabilité sous forme de fraction simplifiée, plutôt que sous forme réelle).

Si on veut par exemple pouvoir utiliser 'CALC' dans le répertoire 'PROBA', tout en le laissant dans le répertoire 'ARIT', il suffit d'inclure le programme:

```
«  ARIT  CALC  PROBA  »  
dans le répertoire 'PROBA' (en l'appelant 'CALC' à nouveau, mais  
ce n'est pas obligatoire).
```

On trouvera donc dans le répertoire 'PROBA':

'CNP'	: nombre de combinaisons sans répétitions, de p éléments pris parmi n .
'ANP'	: nombre d'arrangements de p éléments parmi n .
'GANP'	: nombre de combinaisons avec répétitions, de p éléments pris parmi n .
'BINO'	: liste des coefficients du binôme.
'BNP'	: loi binômiale.
'FBNP'	: fonction de répartition de la loi binômiale.
'HNP'	: loi hypergéométrique.
'FHNAP'	: fonction de répartition de la loi hypergéométrique.
'POIS'	: loi de Poisson.
'FPOIS'	: fonction de répartition de la loi de Poisson.
'GEO'	: loi géométrique.
'FGEO'	: fonction de répartition de la loi géométrique.
'PRP'	: loi de Pascal.
'FPRP'	: fonction de répartition de la loi de Pascal.
'JRP'	: loi binômiale négative.
'FJRP'	: fonction de répartition de la loi binômiale négative.
'FEXP'	: fonction de répartition de la loi exponentielle.
' $\rightarrow$ NRM'	: fonction de répartition de la loi normale.
' $\rightarrow$ NCR'	: fonction de répartition de la loi normale centrée réduite.
'NCR $\rightarrow$ '	: Inverse de la fonction de répartition de la loi normale centrée réduite.
'NRM $\rightarrow$ '	: Inverse de la fonction de répartition de la loi normale.
'AJUST'	: Ajustement par une loi normale $N(m, \sigma)$.

COMBINAISONS SANS REPETITIONS.

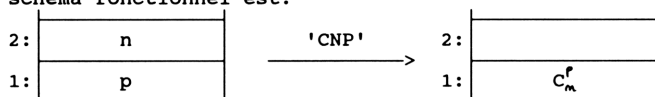
=====

'CNP' calcule le nombre de parties de p éléments extraites d'un ensemble à n éléments (combinaisons sans répétitions). Ce nombre est noté C_m^p et vaut

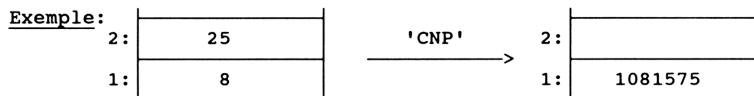
$$\frac{n!}{p!(n-p)!}.$$

'CNP' ne fait que reprendre COMB, mais est à la fois plus accessible et plus explicite. D'autre part le résultat de 'CNP' est 0 dans le cas où p n'est pas compris entre 0 et n.

Le schéma fonctionnel est:



'CNP': (Checksum: # 3917d, Taille: 37.5 octets)
 « IFERR COMB THEN DROP2 0 END »

**ARRANGEMENTS.**

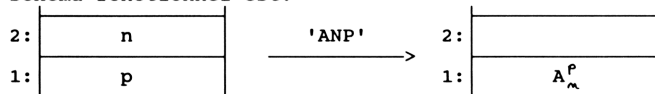
=====

'ANP' calcule le nombre d'arrangements de p éléments extraits d'un ensemble à n éléments (combinaisons ordonnées sans répétitions). Ce nombre est noté A_m^p et vaut

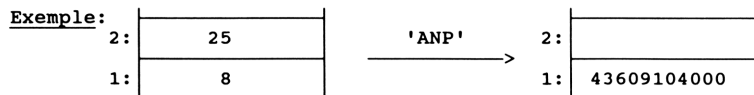
$$\frac{n!}{(n-p)!}.$$

'ANP' ne fait que reprendre PERM, mais est à la fois plus accessible et plus explicite. D'autre part le résultat de 'ANP' est 0 dans le cas où p n'est pas compris entre 0 et n.

Le schéma fonctionnel est:



'ANP': (Checksum: # 40761d, Taille: 37.5 octets)
 « IFERR PERM THEN DROP2 0 END »



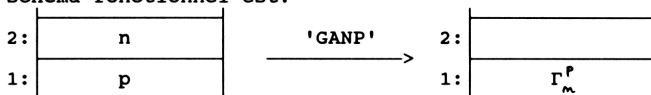
COMBINAISONS AVEC REPETITIONS.

=====

'GANP' calcule le nombre de combinaisons, avec répétitions possibles, de p éléments extraits d'un ensemble à n éléments. Ce nombre est noté Γ_m^p et vaut

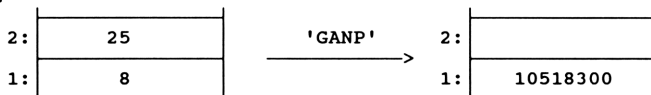
$$\Gamma_m^p = C_{m+p-1}^p$$

Le schéma fonctionnel est:



'GANP': (Checksum: # 7101d, Taille: 40 octets)
 « DUP ROT + 1 - SWAP CNP »

Exemple:

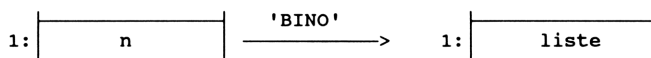
LISTE DES COEFFICIENTS
DU BINOME.

=====

'BINO' donne la liste des coefficients du binôme apparaissant dans le développement de $(x+y)^n$, c'est à dire les coefficients:

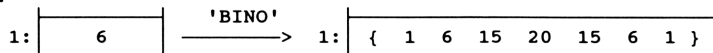
C_m^k avec $0 \leq k \leq n$. (liste ordonnée suivant k).

Le schéma fonctionnel est:



'BINO': (Checksum: # 23431d, Taille: 70.5 octets)
 «
 → n
 « 0 n FOR i n i COMB NEXT
 n 1 + →LIST
 »
 »

Exemple:



LOI BINOMIALE B(n,p).

=====

'BNP' calcule la probabilité $p(X=k)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi binomiale de paramètres n, p .

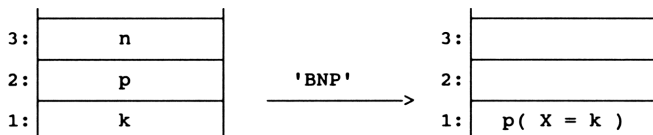
Autrement dit, $p(X=k)$ est la probabilité de voir se réaliser k succès dans une suite de n épreuves indépendantes, la probabilité d'un succès à l'une quelconque de ces épreuves étant égale à p .

Nécessairement k, n sont entiers avec $0 \leq k \leq n$, et $0 \leq p \leq 1$.

La formule utilisée est:

$$p(X=k) = C_n^k p^k (1-p)^{n-k}.$$

Le schéma fonctionnel est:



N.B. 'BNP' appelle le programme 'CNP'.

'BNP': (Checksum: # 2409d, Taille: 109 octets)

```

"  →  n  p  k
  "    n  k  CNP
    IF DUP THEN
      p k ^ 1 p - n k - ^ * *
    END
  "

```

Exemple:



(on obtient ainsi la probabilité d'obtenir 10 fois pile si on effectue 25 lancers successifs d'une pièce honnête).

FONCTION DE REPARTITION DE LA LOI BINOMIALE B(n,p).

=====

'FBNP' calcule la probabilité $p(X \leq k)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi binomiale de paramètres n, p . On obtient ainsi la fonction de répartition de X .

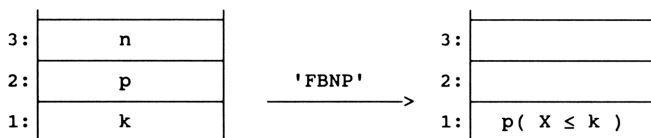
Autrement dit, $p(X \leq k)$ est la probabilité de voir se réaliser au plus k succès dans une suite de n épreuves indépendantes, la probabilité d'un succès à l'une de ces épreuves étant égale à p .

Nécessairement k, n sont entiers avec $0 \leq k \leq n$, et $0 \leq p \leq 1$.

La formule utilisée est:

$$p(X \leq k) = \sum_{i=0}^k C_n^i p^i (1-p)^{(n-i)}.$$

Le schéma fonctionnel est:



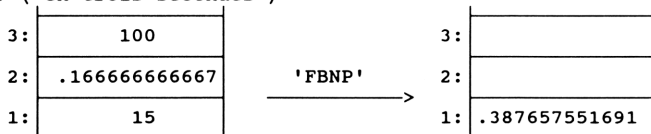
'FBNP': (Checksum: # 13835d, Taille: 172.5 octets)

```

"  →  n  p  k
    "  IF  n  2  /  k  <  THEN
        1  n  1  p  -  n  k  -  1  -  FBNP  -
    ELSE
        0
        0  k  FOR  j
            n  p  j  BNP  +
        NEXT
        k  0  ≥  *
    END
"

```

Exemple: (en trois secondes)



On obtient ainsi, par exemple, la probabilité d'obtenir au plus 15 fois le résultat 6 si on lance 100 fois de suite un dé honnête.

LOI HYPERGEOMETRIQUE H(n,a,p).

=====

'HNAP' calcule la probabilité $p(X=k)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi hypergéométrique de paramètres n, a, p .

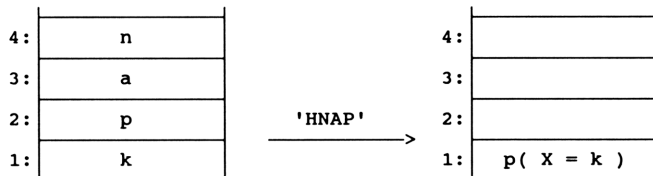
Pour prendre un exemple, $p(X=k)$ est la probabilité d'obtenir, lors d'un tirage simultané de a individus différents dans une population totale d'effectif n , k individus ayant une propriété P donnée, si on suppose que la proportion d'individus ayant cette propriété dans la population entière est égale à p .

Nécessairement k, a, n sont entiers avec $0 \leq k \leq a \leq n$, et $0 \leq p \leq 1$.

La formule utilisée est:

$$p(X=k) = \frac{C_{n,p}^k C_{n(1-p)}^{a-k}}{C_n^a}$$

Le schéma fonctionnel est:

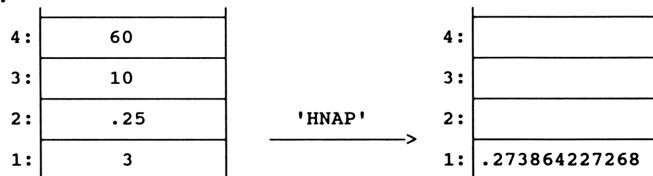


'HNAP': (Checksum: # 40776d, Taille: 136.5 octets)

```

"  →  n  a  p  k
    "  n  p  *  k  CNP
      n  1  p  -  *  a  k  -  CNP  *
      IF  DUP  THEN  n  a  CNP  /  END
    "
  
```

Exemple:



On peut dire que ce résultat représente la probabilité d'obtenir exactement 3 boules blanches, si on effectue un tirage exhaustif (ou encore simultané, ou encore sans remise) de 10 boules dans une urne qui en contient 60, dont 15 blanches (la proportion $15/60 = .25$ est au niveau 2 de la pile).

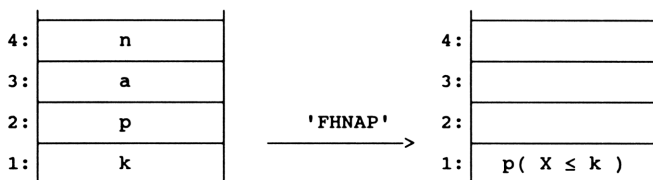
FONCTION DE REPARTITION DE LA LOI HYPERGEOMETRIQUE H(n,a,p).

=====

'FHNAP' calcule la probabilité $p(X \leq k)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi hypergéométrique de paramètres n, a, p . On obtient ainsi ce qu'on appelle la fonction de répartition de X .

Pour prendre un exemple, $p(X \leq k)$ est la probabilité d'obtenir, lors d'un tirage simultané de a individus différents dans une population totale d'effectif n , au plus k individus ayant une propriété P donnée, si on suppose que la proportion d'individus ayant cette propriété dans la population entière est égale à p .

Le schéma fonctionnel est:



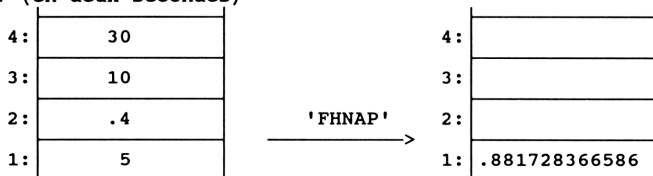
'FHNAP': (Checksum: # 47720d, Taille: 223 octets)

```

"
  " → n a p k
    " 'a-n*(1-p)' EVAL 0 MAX a n p * MIN
      → mink maxk
        " IF k mink ≥ k maxk ≤ AND THEN
          0
          mink k FOR i
            n a p i H NAP +
          NEXT
        ELSE 0 END
      "
    "
  "

```

Exemple: (en deux secondes)



autrement dit,
si on effectue un tirage exhaustif de 10 boules dans une urne qui en contient 30 (dont 12 blanches, la proportion de boules blanches étant .4), la probabilité d'en tirer au plus 5 vaut environ 0,88.

LOI DE POISSON P(L).

=====

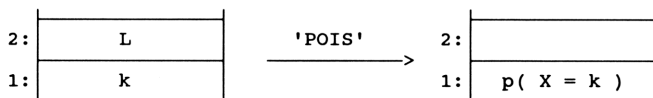
'POIS' calcule la probabilité $p(X=k)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi de Poisson de paramètre L . L est un réel strictement positif et k est un entier naturel.

la formule utilisée est:

$$p(X=k) = \exp(-L) \frac{L^k}{k!}$$

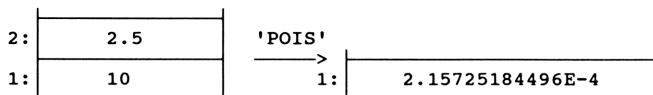
Les lois de Poisson permettent de modéliser des problèmes de trafic et de files d'attente.

Le schéma fonctionnel est:



'POIS': (Checksum: # 32314d, Taille: 68 octets)

“ → L k 'EXP(-L)*L^k/k!' ”

Exemple:

FONCTION DE REPARTITION DE LA LOI DE POISSON P(L).

=====

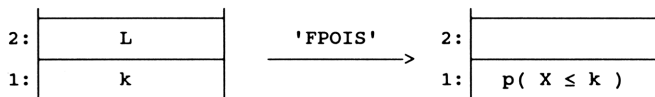
'FPOIS' calcule la probabilité $p(X \leq k)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi de Poisson de paramètre L . L est un réel strictement positif et k est un entier naturel.

la formule utilisée est:

$$p(X \leq k) = \exp(-L) \sum_{i=0}^{i=k} \frac{L^i}{i!}$$

Les lois de Poisson permettent de modéliser des problèmes de trafic et de files d'attente.

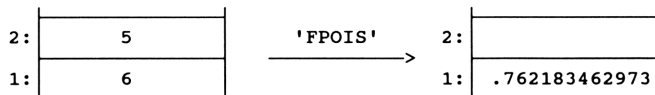
Le schéma fonctionnel est:



'FPOIS': (Checksum: # 45358d, Taille: 86.5 octets)

“ → L k 'Σ(j=0,k,POIS(L,j))'
”

Exemple: (en moins d'une seconde)



LOI GEOMETRIQUE G(p).

=====

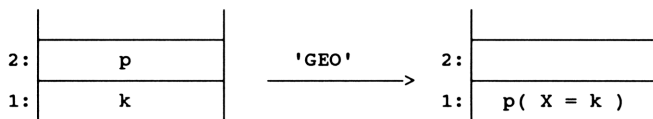
'GEO' calcule la probabilité $p(X=k)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi géométrique de paramètre p . p est un réel compris entre 0 et 1, et k est un entier strictement positif.

la formule utilisée est:

$$p(X=k) = p \cdot (1-p)^{(k-1)}.$$

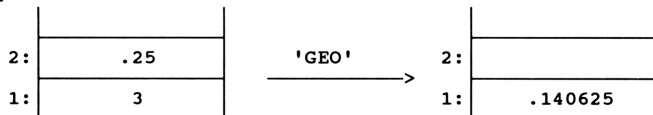
La loi géométrique de paramètre p donne le temps d'attente du premier succès dans une suite d'épreuves indépendantes, la probabilité de succès dans chacune de ces épreuves étant égale à p .

Le schéma fonctionnel est:



'GEO': (Checksum: # 63475d, Taille: 74.5 octets)

" → p k 'p*(1-p)^(k-1)*(k>0)' "

Exemple:

Autrement dit, si on effectue des tirages avec remise d'une carte dans un jeu "normal", la probabilité pour que le premier trèfle apparaisse au troisième tirage est égale à 0,140625 (la probabilité de succès à chaque tirage est égale à .25).

FONCTION DE REPARTITION DE LA LOI GEOMETRIQUE G(p).

=====

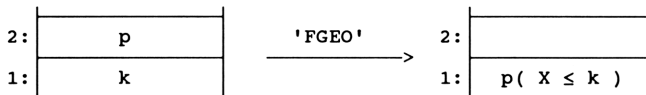
'FGEO' calcule la probabilité $p(X \leq k)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi géométrique de paramètre p. p est un réel compris entre 0 et 1, et k est un entier strictement positif.

la formule utilisée est:

$$p(X \leq k) = 1 - (1-p)^k.$$

La loi géométrique de paramètre p donne le temps d'attente du premier succès dans une suite d'épreuves indépendantes, la probabilité de succès dans chacune de ces épreuves étant égale à p. Il s'agit donc de savoir qu'elle est la probabilité pour que ce premier succès arrive au plus tard lors de la k-ième tentative.

Le schéma fonctionnel est:



'FGEO': (Checksum: # 2016d, Taille: 68.5 octets)

“ → p k '(1-(1-p)^k)*(k>0)'
”

Exemple:



Autrement dit, si on lance plusieurs fois de suite une pièce honnête, la probabilité d'obtenir pile au plus tard au 4ème lancer est 0,9375.

LOI DE PASCAL P(r,p).

=====

'PRP' calcule la probabilité $p(X=k)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi de Pascal de paramètres r, p . p est un réel compris entre 0 et 1, r et k sont deux entiers ($1 \leq r \leq k$).

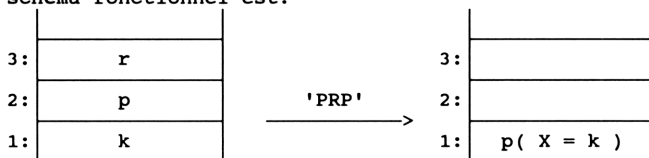
La formule utilisée est:

$$p(X=k) = C_{k-1}^{r-1} p^r (1-p)^{(k-r)}.$$

La loi de Pascal de paramètres r, p donne le temps d'attente du r -ième succès dans une suite d'épreuves indépendantes, la probabilité de succès dans chacune de ces épreuves étant égale à p .

La loi géométrique de paramètre p n'est autre que la loi de Pascal de paramètres $1, p$.

Le schéma fonctionnel est:



N.B: le programme 'PRP' appelle le programme 'CNP'.

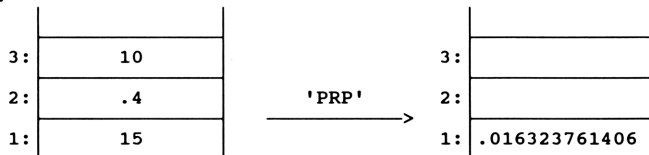
'PRP': (Checksum: # 13969d, Taille: 126.5 octets)

```

«   →   r   p   k
«   k   1   -   r   1   -   CNP
      IF   DUP   THEN   'p^r*(1-p)^(k-r)'   EVAL   *   END
»
»

```

Exemple:



Si on effectue par exemple des tirages, avec remise, d'une boule dans une urne qui contient 40% de boules blanches, la probabilité pour que la 10ème boule blanche apparaisse au 15ème tirage est donc 0,016323761406.

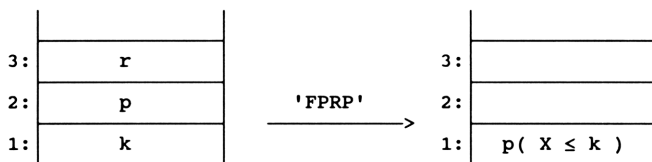
FONCTION DE REPARTITION DE LA LOI DE PASCAL $P(r,p)$.

=====

'FPRP' calcule la probabilité $p(X \leq k)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi de Pascal de paramètres r, p . p est un réel compris entre 0 et 1, r et k sont deux entiers ($1 \leq r \leq k$).

La loi de Pascal de paramètres r, p donne le temps d'attente du r -ième succès dans une suite d'épreuves indépendantes, la probabilité de succès dans chacune de ces épreuves étant égale à p . On cherche donc ici la probabilité pour que le r -ième succès intervienne au plus tard lors de la k -ème tentative.

Le schéma fonctionnel est le suivant:



'FPRP': (Checksum: # 53656d, Taille: 95.5 octets)

« → r p k 'Σ(j=r,k,PRP(r,p,j))' »

Exemple:



Si on jette plusieurs fois de suite une pièce équilibrée, la probabilité pour que le 10-ème résultat "pile" intervienne au plus tard à la 15ème tentative est donc 0,15087890625.

LOI BINOMIALE NEGATIVE J(r,p).

=====

'JRP' calcule la probabilité $p(X=k)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi binomiale négative de paramètres r, p . p est un réel compris entre 0 et 1, r et k sont deux entiers ($1 \leq r$, $0 \leq k$).

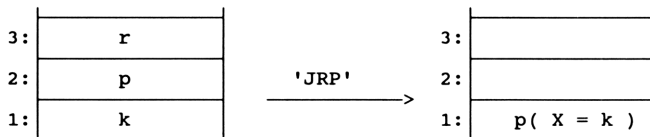
la formule utilisée est:

$$p(X=k) = C_{k+r-1}^k p^r (1-p)^k.$$

La loi binomiale négative de paramètres r, p donne le nombre d'échecs précédant le r -ième succès dans une suite d'épreuves indépendantes, la probabilité de succès dans chacune de ces épreuves étant égale à p .

Remarque: dire que X suit la loi binomiale négative $J(r, p)$, de paramètres n, p , c'est dire que $X+r$ suit la loi de Pascal $P(r, p)$ de paramètres r, p .

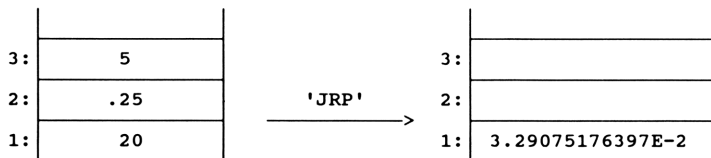
Le schéma fonctionnel est:



N.B: le programme 'JRP' appelle le programme 'PRP'.

'JRP': (Checksum: # 32211d, Taille: 31.5 octets)
 " 3 PICK + PRP "

Exemple:



Si on effectue des tirages avec remise d'une carte dans un jeu normal, la probabilité de tirer exactement 20 cartes qui ne soient pas des trèfles avant de tirer le cinquième trèfle est donc environ égale à 0,0329.

FONCTION DE REPARTITION DE LA LOI BINOMIALE NEGATIVE J(r,p).

=====

'FJRP' calcule la probabilité $p(X \leq k)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi binomiale négative de paramètres r, p . p est un réel compris entre 0 et 1, r et k sont deux entiers ($1 \leq r, 0 \leq k$).

La loi binomiale négative de paramètres r, p donne le nombre d'échecs précédant le r -ième succès dans une suite d'épreuves indépendantes, la probabilité de succès dans chacune de ces épreuves étant égale à p .

Il s'agit donc de connaître la probabilité pour que le nombre d'échecs précédant le r -ème succès soit au plus égal à k .

Le schéma fonctionnel est:



N.B: le programme 'FJRP' appelle le programme 'FPRP'.

'FJRP': (Checksum: # 24952d, Taille: 33.5 octets)

“ 3 PICK + FPRP ”

Exemple:



Si on effectue des tirages, avec remise, d'une boule dans une urne qui contient 60% de boules blanches, la probabilité que le nombre de boules non blanches tirées avant la 5ème boule blanche soit au plus égal à 4 est égale à 0,73343232.

FONCTION DE REPARTITION
DE LA LOI EXPONENTIELLE.

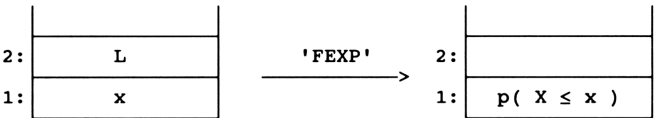
=====

'FEXP' donne la probabilité $p(X \leq x)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi exponentielle de paramètre L , et où x est un réel donné.

La formule utilisée est: $p(X \leq x) = 0$ si $x \leq 0$.

et, si $x \geq 0$:
$$p(X \leq x) = \int_0^x L \exp(-Lt) \, dt = 1 - \exp(-Lx)$$

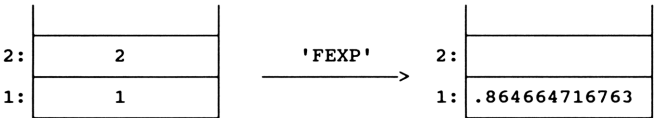
Le schéma fonctionnel est:



'FEXP': (Checksum: # 6632d, Taille: 68.5 octets)

“ → L x ' (1 - EXP (-L * x)) * (x ≥ 0) ' ”

Exemple:



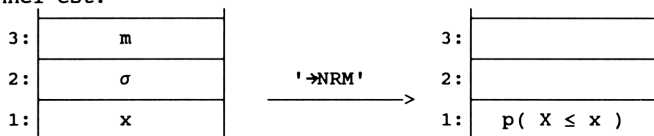
FONCTION DE REPARTITION DE LA LOI NORMALE N (m,σ).

=====

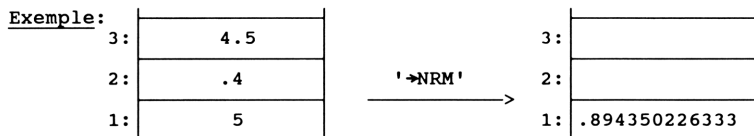
'→NRM' calcule la probabilité $p(X \leq x)$, où X est une v.a.r. (variable aléatoire réelle) suivant la loi normale de paramètres m , σ . m et σ sont deux réels (m représente l'espérance de X , et σ est l'écart-type de X . Bien sûr, $\sigma > 0$). x est un réel quelconque. 1 a formule utilisée est:

$$p(X \leq x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{(t-m)^2}{2\sigma^2}\right) dt$$

Les lois normales permettent de modéliser des problèmes liés à l'observation de populations d'effectif important. Le schéma fonctionnel est:



'→NRM': (Checksum: # 64285d, Taille: 36 octets)
 " SWAP SQ SWAP UTPN NEG 1 + "

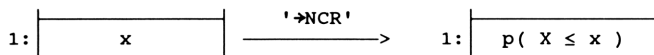


FONCTION DE REPARTITION DE LA LOI NORMALE CENTREE REDUITE.

=====

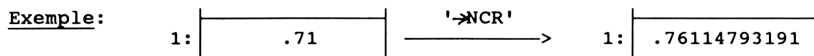
'→NCR' calcule $p(X \leq x)$, où X est une v.a.r. suivant la loi normale centrée réduite $N(0, 1)$. Il s'agit donc d'un cas particulier de ce qui précède: ici $m=0$ et $\sigma=1$.

Le schéma fonctionnel est donc plus simple:



le programme '→NCR' appelle le programme '→NRM'.

'→NCR': (Checksum: # 52295d, Taille: 33.5 octets)
 " 0 1 ROT →NRM "

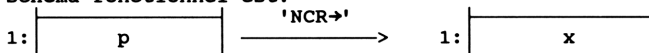


INVERSE DE LA FONCTION DE REPARTITION DE LA LOI NORMALE N (0 , 1).

=====

'NCR→' permet d'inverser la fonction de répartition de la loi normale centrée réduite $N(0,1)$. Si X est une variable aléatoire suivant cette loi et si p est un nombre compris entre 0 et 1, on calcule ainsi l'unique réel x tel que $p(X \leq x) = p$.

Le schéma fonctionnel est:



Remarque: 'NCR→' utilise l'instruction ROOT pour résoudre l'équation $p(X \leq x) = p$. Il est absolument nécessaire que p soit élément de l'intervalle $]0,1[$.

'NCR→': (Checksum: # 53511d, Taille: 93.5 octets)

```

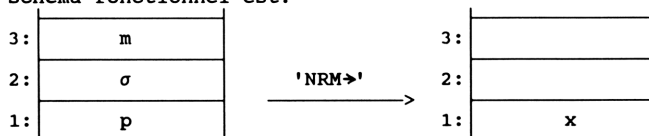
"  →  p
  "    " 1 p - 0 1 x UTPN - "
    " 'x' 0 ROOT 'x' PURGE
  "
"
```

INVERSE DE LA FONCTION DE REPARTITION DE LA LOI NORMALE N (m , σ).

=====

'NRM→' permet d'inverser la fonction de répartition de la loi normale $N(m, \sigma)$. Si X est une variable aléatoire suivant cette loi et si p est un nombre compris entre 0 et 1, on calcule ainsi l'unique réel x tel que $p(X \leq x) = p$.

Le schéma fonctionnel est:



Remarque: 'NRM→' appelle 'NCR→'. Il est absolument nécessaire que p soit élément de l'intervalle $]0,1[$.

'NRM→': (Checksum: # 5370d, Taille: 31 octets)

```

"  NCR→ * + "
```

Exemple: pour calculer le 3ème quartile de la loi $N(2,1.5)$, on tape:
2 1.5 .75 NRM→ et on obtient 3.01173462529 en 3 secondes.

AJUSTEMENT PAR UNE LOI NORMALE $N(m, \sigma)$.

=====

'AJUST' permet de calculer le meilleur ajustement possible, par une loi normale $N(m, \sigma)$, d'une loi dont on connaît au moins deux valeurs de la fonction de répartition (ou d'une statistique simple dont on connaît au moins deux points du polygone des fréquences cumulées).

On suppose donc connus, pour une variable aléatoire X , un certain nombre de points (a,b) , (c,d) , (x,y) tels que :

$$p(X \leq a) = b, p(X \leq c) = d, \dots, p(X \leq x) = y,$$

c'est-à-dire un certain nombre de valeurs de la fonction de répartition de X .

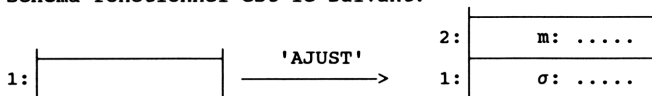
Le programme 'AJUST' calcule les paramètres m (moyenne) et σ (écart-type) de la loi normale $N(m, \sigma)$ qui approche au mieux la loi de la variable X .

Avant d'appeler 'AJUST', il convient de placer les données dans la matrice ΣDAT de la manière suivante,

$$\Sigma DAT = \begin{bmatrix} a & b \\ c & d \\ \dots & \dots \\ x & y \end{bmatrix}$$

l'ordre des lignes dans ΣDAT n'a ici aucune importance.

Le schéma fonctionnel est le suivant:



N.B.: A la sortie de 'AJUST', le contenu de ΣDAT est inchangé.

'AJUST' appelle 'NCR→'.

On veillera à ce que les éléments de la seconde colonne de ΣDAT soient tous compris entre 0 et 1.

'AJUST': (Checksum: # 21415d, Taille: 118 octets)

```

"  ΣDAT  DUP
  1  NE   FOR   i
      i   { 2 }  +  DUP2  GET  NCR→  PUT
NEXT
STOE  2  1  COLE  LR  ROT  STOE
SWAP  DTAG  "m"  →TAG  SWAP  DTAG  "σ"  →TAG
"
```

PROGRAMME 'AJUST': EXEMPLES D'UTILISATION.

=====

Exemple 1:

Une variable aléatoire X suit une loi normale. On sait que $p(X \leq 1) = .1635$ et $p(X \leq 5) = .8053$.
Calculer la moyenne m et l'écart-type σ de X .
Ici le problème admet une solution exacte, et 'AJUST' permet de la trouver.
On place la matrice $\begin{bmatrix} 1 & .1635 \\ 5 & .8053 \end{bmatrix}$ dans ΣDAT ,
et on appelle 'AJUST'.
On obtient, au bout de neuf secondes, et en mode 4 FIX:
Ainsi, X suit la loi normale $N(m, \sigma)$ de moyenne $m = 3.1298$, et d'écart-type $\sigma = 2.1729$.
A titre de vérification, en partant de cette pile, la séquence 1 $\rightarrow$ NRM donne bien le résultat 0.1635....

2:	m: 3.1298
1:	σ : 2.1729

Exemple 2:

On considère une population de 100 individus classés en fonction de leur taille, exprimée en cm.

Taille	Effectif
[150,160[	6
[160,165[	11
[165,170[	20
[170,175[	25
[175,180[	20
[180,185[	11
[185,190[	5
[190,200[	2

On désire approcher la variable T égale à la taille d'un individu quelconque de cette population par une loi normale.

On commence par calculer le vecteur des effectifs cumulés [6 17 37 .. 100].
On le divise par l'effectif total 100 pour avoir le vecteur des fréquences cumulées:

[0.06 0.17 0.37 0.62 0.82 0.93 0.98 1]

On place dans ΣDAT la matrice:

$\begin{bmatrix} 160 & 0.06 \\ 165 & 0.17 \\ 170 & 0.37 \\ 175 & 0.62 \\ 180 & 0.82 \\ 185 & 0.93 \\ 190 & 0.98 \end{bmatrix}$ (Remarquez qu'on laisse tomber le point (200,1) qui bloquerait 'AJUST')

On appelle le programme 'AJUST'.

Au bout de 37 secondes, on obtient, en mode 2 FIX:
La taille T d'un individu de la population décrite ici est donc approchable au mieux par la loi normale $N(172.75, 8.26)$.

2:	m: 172.75
1:	σ : 8.26

STATISTIQUES SIMPLES

Le répertoire 'STAT1' contient un ensemble de programmes permettant d'effectuer l'étude d'une statistique simple, discrète ou donnée par classes, et connue par les effectifs (ou fréquences) correspondant à chaque valeur ou à chaque classe du caractère X à étudier.

Dans le cas d'une statistique discrète (X_i, N_i), où les X_i sont les valeurs du caractère X, et les N_i sont les effectifs (ou fréquences), on place les couples (X_i, N_i) dans la matrice ΣDAT (la première colonne pour les X_i , la seconde pour les N_i).

Dans le cas d'une statistique donnée par classes ($[A_i, A(i+1)[, N_i$), on place dans une variable nommée DATA une liste formée elle-même de deux sous-listes: la première pour les extrémités de classes, la deuxième pour les effectifs (ou fréquences).

Par exemple la statistique:

X_i	[0,2[	[2,6[	[6,8[	[8,14[
N_i	22	25	15	10

sera représentée dans la variable DATA par la liste:

{ { 0 2 6 8 14 } { 22 25 15 10 } }

Une variable $\Sigma DT1$ est utilisée par certains programmes du répertoire 'STAT1' pour sauvegarder ΣDAT ou pour présenter les calculs intermédiaires amenant à un résultat donné (le calcul du coefficient de GINI par exemple).

Les programmes du répertoire 'STAT1' sont:

G+D : transformation d'une statistique par classes en stat. discrète.
MK : calcul du moment d'ordre k.
MCK : calcul du moment centré d'ordre k.
MCRK : calcul du moment centré réduit d'ordre k.
YULE : coefficients de Yule, Kelley, et Pearson.
QTLE : calcul des quantiles.
ECAR : calcul de l'écart arithmétique moyen.
MGE0 : calcul de la moyenne géométrique.
MHAR : calcul de la moyenne harmonique.
MDLE : calcul de la médiale (tableau de calcul dans $\Sigma DT1$).
GINI : calcul de l'indice de Gini (tableau de calcul dans $\Sigma DT1$).
LRTZ : tracé de la courbe de Lorentz (ou de concentration).
HIST : tracé de l'histogramme.
PFC : tracé du polygone des fréquences cumulées.
***CUM** : création d'un tableau de cumuls.
CUM+ : retour d'un tableau de cumuls au tableau initial.
COL.N: affiche une colonne quelconque de $\Sigma DT1$ (pour GINI & MDLE).
C.COL: calcul d'une colonne dépendante des 2 colonnes de ΣDAT .
MODE : modification d'une ou des deux colonnes de ΣDAT .

TRANSFORMATION D'UNE STATISTIQUE PAR CLASSES EN STATISTIQUE DISCRETE.

=====

'G→D' transforme une statistique simple donnée par classes en statistique discrète. La transformation s'effectue en concentrant l'effectif d'une classe en son milieu.

La statistique groupée initiale doit se trouver dans 'DATA'. Le résultat est envoyé dans EDAT.

La pile n'est pas affectée par 'G→D'. Toutefois l'appel à 'G→D' se solde par un message d'erreur dans le cas où la première sous-liste de 'DATA' ne contient pas exactement un élément de plus que la seconde.

'G→D': (Checksum: # 15895d, Taille: 111 octets)

```
«   DATA 1 GET OBJ→
   → n
   «   2   n   START
       OVER + 2 / n ROLLD
       NEXT
       DROP DATA 2 GET OBJ→
       { 2 } SWAP + →ARRY TRN STOZ
   »
»
```

Exemple:

La statistique:

Xi	[0,2[	[2,6[	[6,8[	[8,14[
Ni	22	25	15	10

représentée dans la variable DATA par la liste:
{ { 0 2 6 8 14 } { 22 25 15 10 } },

est transformée en la matrice:

```
[ [ 1 22 ]
  [ 4 25 ]
  [ 7 15 ]
  [ 11 10 ] ]
```

et cette matrice est placée
dans EDAT.

MOMENT D'ORDRE K D'UNE STATISTIQUE SIMPLE.

=====

'MK' calcule le moment d'ordre k (noté Mk) de la statistique discrète contenue dans la variable ΣDAT. Dans le cas d'une statistique donnée par classe (et contenue dans la variable 'DATA'), il convient d'appeler tout d'abord le programme 'G→D', pour la transformer en statistique discrète.

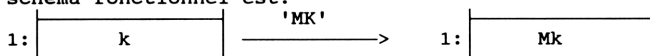
La formule donnant le moment d'ordre k de la statistique (Xi, Ni) est:

$$M_k = \frac{1}{N} \sum Ni Xi^k \quad (N = \text{effectif total} = \sum Ni).$$

Si k=1, on obtient la moyenne arithmétique ($\sum Ni Xi$) / N.

Avec k=2, on obtient le carré ($\sum Ni Xi^2$) / N de la moyenne quadratique.

Le schéma fonctionnel est:



'MK': (Checksum: # 17889d, Taille: 80.5 octets)

```

"      → k
      "  EDAT  OBJ→  DROP  0
        1  NE  START
          ROT  k  ^  ROT  *  +
        NEXT
        TOT  2  GET  /
      "
"
"

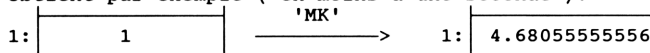
```

Exemple: Avec la statistique :

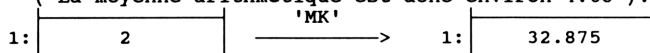
Xi	1	4	7	11
Ni	22	25	15	10

représentée dans ΣDAT par la matrice à deux colonnes: $\begin{bmatrix} 1 & 22 \\ 4 & 25 \\ 7 & 15 \\ 11 & 10 \end{bmatrix}$,

on obtient par exemple (en moins d'une seconde):



(La moyenne arithmétique est donc environ 4.68).



(La moyenne quadratique est donc $\sqrt{32.875} \approx 5.73$).

MOMENT CENTRE D'ORDRE K D'UNE STATISTIQUE SIMPLE.

=====

'MCK' calcule le moment centré d'ordre k (noté μ_k) de la statistique discrète contenue dans la variable ΣDAT . Dans le cas d'une statistique donnée par classe (et contenue dans la variable 'DATA'), il convient d'appeler tout d'abord le programme 'G+D', pour la transformer en statistique discrète.

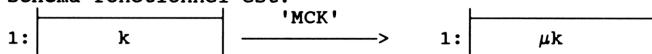
La formule donnant le moment centré d'ordre k de la statistique (X_i, N_i) est:

$$\mu_k = \frac{1}{N} \sum N_i (X_i - \bar{X})^k \quad (N = \sum N_i, \bar{X} = \text{moyenne arithmétique}).$$

Si k=1, on obtient 0.

Avec k=2, on obtient la variance: $(\sum N_i (X_i - \bar{X})^2) / N$ (c'est-à-dire le carré de l'écart-type).

Le schéma fonctionnel est:



N.B: Le programme 'MCK' appelle le programme 'MK'.

'MCK':(Checksum: # 54860d, Taille: 108.5 octets)

```

"      → k
      "  ΣDAT  OBJ→  DROP  1  MK
      → m
      "    0    1  NE  START
        ROT  m  -  k  ^  ROT  *  +
      NEXT
      TOT  2  GET  /
      "
"
"

```

Exemple: Avec la statistique :

X_i	1	4	7	11
N_i	22	25	15	10

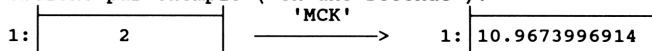
représentée dans ΣDAT par la matrice à deux colonnes:

```

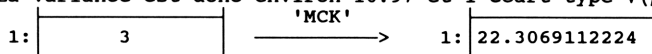
[ [ 1  22 ]
  [ 4  25 ]
  [ 7  15 ]
  [ 11 10 ] ],

```

on obtient par exemple (en une seconde):



(La variance est donc environ 10.97 et l'écart type $\sqrt{\mu_2} \approx 3.31$)



MOMENT CENTRE REDUIT D'ORDRE K D'UNE STATISTIQUE SIMPLE.

=====

'MCRK' calcule le moment centré réduit d'ordre k (noté α_k) de la statistique discrète contenue dans la variable Σ DAT. Dans le cas d'une statistique donnée par classe (et contenue dans la variable 'DATA'), il convient d'appeler tout d'abord le programme 'G-D', pour la transformer en statistique discrète.

La formule donnant le moment centré réduit d'ordre k de la statistique (Xi, Ni) est:

$$\alpha_k = \mu_k / (\sigma^k),$$

où μ_k est le moment centré d'ordre k (voir le programme 'MCK') et où σ est l'écart-type.

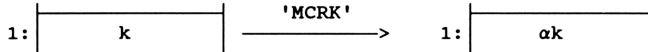
Avec k=1, on obtient 0.

Avec k=2, on obtient 1.

Avec k=3, on obtient le coefficient d'assymétrie.

Avec k=4, on obtient le coefficient d'aplatissement.

Le schéma fonctionnel est:



N.B. Le programme 'MCRK' appelle le programme 'MCK'.

'MCRK': (Checksum: # 57467d, Taille: 46.5 octets)

“ **DUP MCK 2 MCK ✓ ROT ^ /**
”

Exemple:

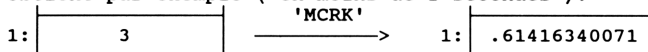
Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	10

[[1 22]
[4 25]
[7 15]
[11 10]],

représentée dans Σ DAT par la matrice à deux colonnes:

on obtient par exemple (en moins de 2 secondes) :



COEFFICIENTS DE YULE, KELLEY, ET PEARSON.

=====

'YULE' calcule les coefficients de Yule, Kelley, et de Pearson d'une statistique donnée par classes, et contenue dans la variable 'DATA'.

Les formules utilisées sont les suivantes:

Coefficient de Yule: $s = (q_3 + q_1 - 2M) / (q_3 - q_1)$.

Coefficient de Kelley: $k = 2 * (q_3 - q_1) / (d_9 - d_1)$.

1er coefficient de Pearson: $p_1 = 3 * (m - M) / \sigma$.

2ème coefficient de Pearson: $p_2 = (m - md) / \sigma$ (pour une série statistique unimodale de mode md).

Avec: q_1, q_3 : premier et troisième quartiles.

d_1, d_9 : premier et neuvième déciles.

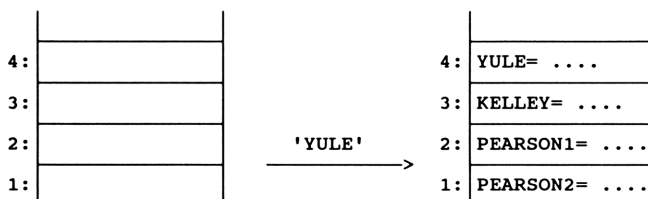
m : moyenne arithmétique.

M : médiane. σ : écart-type.

Remarque:

Le 2ème coefficient de Pearson n'a de sens que pour des classes de même amplitude. Si la statistique n'a pas un mode unique, ce coefficient n'est pas affiché.

Le schéma fonctionnel est:



N.B: 'YULE' appelle les programmes 'QTLE', 'G>D', 'MK' et 'MCK'.

TEXTE DU PROGRAMME 'YULE'
ET EXEMPLE D'UTILISATION.
=====

```
'YULE':( Checksum: # 63367d, Taille: 535.5 octets )
  «
    { .1 .25 .5 .75 .9 } QTLE EVAL G→D
    1 MK 2 MCK ✓ MAXE 2 GET 0 0
    → d1 q1 med q3 d9 moy sig max num lig
    « 1 NE FOR i
      'ΣDAT(i,2)' EVAL
      IF max == THEN
        'num' 1 STO+ i 'lig' STO
      END
    NEXT
    q3 q1 + med 2 * -
    q3 q1 - / "YULE" →TAG
    q3 q1 - d9 d1 -
    / 2 * "KELLEY" →TAG
    moy med - sig / 3 *
    "PEARSON1" →TAG
    IF num 1 == THEN
      moy 'ΣDAT(lig,1)' EVAL - sig /
      "PEARSON2" →TAG
    END
  »
»
```

Exemple:
Avec la statistique

Xi	[0,5[	[5,10[	[10,15[	[15,20[
Ni	22	25	15	10

représentée dans la variable 'DATA' par la liste:
{ { 0 5 10 15 20 } { 22 25 15 10 } },
L'appel du programme 'YULE' provoque au bout de 4 secondes
l'affichage suivant:

4:	YULE: 9.99999999953E-2
3:	KELLEY: 1.11658456486
2:	PEARSON1: .355182651996
1:	PEARSON2: .177318528263

QUANTILES D'UNE STATISTIQUE SIMPLE.

=====

'QTLE' calcule les quantiles de la statistique groupée (donnée par classes) qui est contenue dans la variable 'DATA'.

Rappel:

si q est un réel compris entre 0 et 1, le quantile correspondant à q est la valeur du caractère X pour laquelle la proportion d'effectif cumulé est égale à q . Le calcul du quartile est effectué par interpolation linéaire dans la classe qui permet de dépasser cette proportion.

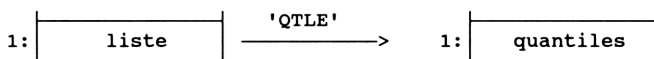
Par exemple:

Si $q = .5$, on obtient la médiane de la statistique (valeur du caractère pour laquelle est atteinte la moitié de l'effectif).

Si $q = k/10$ (k entier, $1 \leq k \leq 9$), on obtient le k ième décile dk .

Si $q = k/4$ (k entier, $1 \leq k \leq 3$), on obtient le k ième quartile qk .
On définit de même les déciles, centiles, milliles...

Le schéma fonctionnel est le suivant:



où "liste" est la liste des proportions q de $]0,1[$ pour lesquelles sont demandés les quantiles.

où "quantiles" est la liste correspondante des quantiles.

Exemples:

liste= { .5 } si on ne veut que la médiane.

liste= { .25 .75 } si on veut le premier et le 3ème quartile.

N.B:

Si une proportion q dans "liste" n'est pas comprise entre 0 et 1, le quantile correspondant (qui n'existe donc pas), est remplacé dans la liste obtenue par la chaîne représentant q .

TEXTE DU PROGRAMME 'QTLE' ET EXEMPLE D'UTILISATION.

=====

'QTLE':(Checksum: # 20526d, Taille: 243 octets)

```

« 0 DATA 2 GET + 0 + →CUM DUP DUP SIZE GET
→ e k
« 1 OVER SIZE FOR j
j DUP2 GET
IF DUP FLOOR THEN
→STR
ELSE
k * e 2
WHILE GETI 4 PICK ≤ REPEAT END
2 - DUP 3 ROLL GETI 3 ROLL
GET OVER 5 ROLL - 3 ROLL - /
DATA 1 GET ROT GETI 3 ROLL
GET OVER - ROT * +
END
PUT
NEXT
»
»
»

```

Exemple:

Avec la statistique:

Xi	[0,2[	[2,6[	[6,8[	[8,14[
Ni	22	25	15	10

qui sera représentée dans la variable DATA par la liste:
{ { 0 2 6 8 14 } { 22 25 15 10 } }.

On met { .1 .25 .5 .75 .9 } au niveau 1 et on appelle 'QTLE':

Au bout de 2 à 3 secondes on obtient, au niveau 1 de la pile, la liste
(en mode 3 FIX):
{ 0.655 1.636 4.240 6.933 9.680 }.

Ce qui signifie que, par exemple:

le premier décile d1 ≈ .655

la médiane Me ≈ 4.240 et le 3ème quartile ≈ 6.933.

ECART ARITHMETIQUE MOYEN D'UNE STATISTIQUE SIMPLE.

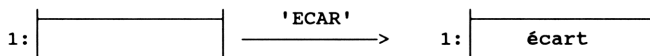
=====

'ECAR' calcule l'écart arithmétique moyen de la statistique discrète contenue dans la variable ΣDAT. Dans le cas d'une statistique donnée par classe (et contenue dans la variable 'DATA'), il convient d'appeler tout d'abord le programme 'G+D', pour la transformer en statistique discrète.

La formule donnant l'écart arithmétique moyen de la statistique (Xi, Ni) est: $\frac{1}{N} \sum Ni | Xi - \bar{X} |$

où N est l'effectif total ΣNi, et où $\bar{X}$ est la moyenne arithmétique.

Le schéma fonctionnel est le suivant:



N.B: le programme 'ECAR' appelle le programme 'MK'.

```

'ECAR':( Checksum: # 54392d, Taille: 93 octets )
"  ΣDAT  OBJ→  DROP  1  MK
  →  m
  "  0  1  NE  START
      ROT  m  -  ABS  ROT  *  +
      NEXT
      TOT  2  GET  /
  "
"

```

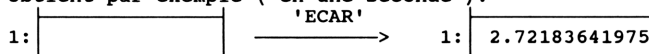
Exemple:

Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	10

représentée dans ΣDAT par la matrice à deux colonnes: $\begin{bmatrix} 1 & 22 \\ 4 & 25 \\ 7 & 15 \\ 11 & 10 \end{bmatrix}$,

on obtient par exemple (en une seconde):



MOYENNE GEOMETRIQUE D'UNE STATISTIQUE SIMPLE.

=====

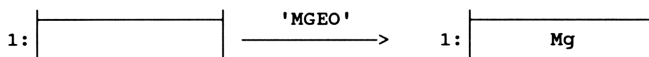
'MGEO' calcule la moyenne géométrique Mg de la statistique discrète contenue dans la variable ΣDAT. Dans le cas d'une statistique donnée par classe (et contenue dans la variable 'DATA'), il convient d'appeler tout d'abord le programme 'G→D', pour la transformer en statistique discrète.

La formule donnant la moyenne géométrique de la statistique (Xi, Ni) est:

$$\left[\prod (Xi)^{Ni} \right]^{(1/N)}$$

où N est l'effectif total ΣNi.

Le schéma fonctionnel est le suivant:



```

'MGEO':( Checksum: # 12712d, Taille: 80 octets )
"  ΣDAT  OBJ→  DROP  TOT  2  GET
  →  t
  "  1  1  NE  START
      3  ROLLD  t  /  ^  *
      NEXT
  "
"
  
```

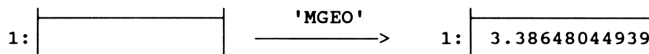
Exemple:

Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	10

représentée dans ΣDAT par la matrice à deux colonnes: $\begin{bmatrix} 1 & 22 \\ 4 & 25 \\ 7 & 15 \\ 11 & 10 \end{bmatrix}$,

on obtient par exemple (en moins d'une seconde):



MOYENNE HARMONIQUE D'UNE STATISTIQUE SIMPLE.

=====

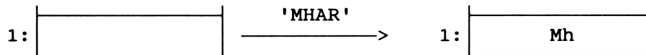
'MHAR' calcule la moyenne harmonique Mh de la statistique discrète contenue dans la variable ΣDAT. Dans le cas d'une statistique donnée par classe (et contenue dans la variable 'DATA'), il convient d'appeler tout d'abord le programme 'G+D', pour la transformer en statistique discrète.

La formule donnant la moyenne harmonique de la statistique (Xi, Ni) est:

$$\frac{1}{Mh} = \frac{1}{N} \sum \frac{Ni}{Xi}$$

où N est l'effectif total ΣNi.

Le schéma fonctionnel est le suivant:



'MHAR': (Checksum: # 20323d, Taille: 82.5 octets)

```

«   EDAT  OBJ->  DROP  TOT  2  GET
  →  t
«   0  1  NE  START
    SWAP  ROT  /  t  /  +
    NEXT
    INV
  »
»

```

Exemple:

Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	10

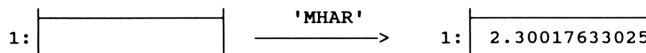
```

[ [ 1 22 ]
  [ 4 25 ]
  [ 7 15 ]
  [ 11 10 ] ],

```

représentée dans ΣDAT par la matrice à deux colonnes:

on obtient par exemple (en moins d'une seconde):



MEDIALE D'UNE STATISTIQUE SIMPLE.

=====

'MDLE' calcule la médiale M1 de la statistique groupée X (donnée par classes) qui est contenue dans la variable 'DATA'.

Rappel:

la médiale est la valeur du caractère X qui permet d'atteindre 50% des "masses" cumulées du caractère.

Si la statistique X est donnée par les couples $([A_i, A_{i+1}], N_i)$, la masse correspondant à la classe $[A_i, A_{i+1}]$ d'effectif N_i est:

$$N_i * (A_{i+1} + A_i) / 2.$$

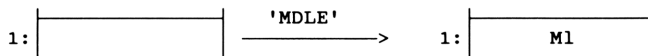
Le calcul de la médiale est effectué par interpolation linéaire dans la classe qui permet de franchir le cap des 50% de masse cumulée.

Le programme 'MDLE' place dans 'EDT1' le tableau nécessaire au calcul de la médiale.

La ligne N°i de ce tableau a la forme suivante (et il y a p lignes s'il y a p classes):

centre des classes	effectifs	masses	masses cumulées
X_i	N_i	$N_i X_i$	$\sum_{j \leq i} N_j X_j$

Le schéma fonctionnel est:



N.B.: le programme 'MDLE' appelle le programme 'G→D'.

**TEXTE DU PROGRAMME 'MDLE'
ET EXEMPLE D'UTILISATION.**

=====

'MDLE':(Checksum: # 34820d, Taille: 273 octets)

```

«  G→D  EX*Y  0
   → m  md
   «  0  1  NE  FOR  i
       EDAT  i { 1 } + GETI  3  ROLL  GET
       DUP2  *  4  PICK  OVER  +
       IF  5  PICK  m  2  /  <  OVER  m  2  /  ≥  AND
       THEN
           DUP  m  2  /  -  3  PICK  /  DATA  1  GET
           i  GETI  3  ROLL  GET  DUP  ROT  -  ROT  *  -
           'md' STO
       END
       NEXT
       NE { 4 } + →ARRY
       'EDT1' STO  DROP  md
   »
»

```

Exemple:

Avec la statistique:

Xi	[0,2[	[2,6[	[6,8[	[8,14[
Ni	22	25	15	10

qui sera représentée dans la variable DATA par la liste:
{ { 0 2 6 8 14 } { 22 25 15 10 } }.

On obtient en 2 secondes:

1: —————> 'MDLE' —————> 1: 6.88571428571

INDICE DE GINI D'UNE STATISTIQUE SIMPLE.

=====

'GINI' calcule l'indice de Gini (ou de concentration) de la statistique discrète contenue dans la variable ΣDAT . Dans le cas d'une statistique donnée par classes (et contenue dans la variable 'DATA'), il convient d'appeler tout d'abord le programme 'G→D', pour la transformer en statistique discrète.

Cet indice est toujours compris entre 0 et 1. il est d'autant plus proche de 1 que la statistique est concentrée (c'est à dire que les masses correspondant à cette distribution statistique se répartissent pour une grande part sur un nombre relativement faible d'individus).

Les calculs intermédiaires nécessaires au calcul de l'indice de Gini (et également au tracé de la courbe de Lorentz, voir le programme 'LRTZ') sont placés dans un tableau sauvegardé dans la variable 'EDT1'.

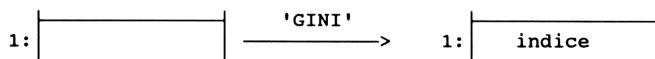
La ligne N°i de ce tableau a la forme suivante (et il y a autant de lignes que de valeurs différentes du caractère étudié):

centres classes	effectifs	effectifs cumulés	% Ui des eff. cumulés	masses	masses cumulées	
X_i	N_i	$\sum_{j \leq i} N_j$	$\frac{100}{N} \sum_{j \leq i} N_j$	$N_i X_i$	$\sum_{j \leq i} N_j X_j$	
colonne1	colonne2	colonne3	colonne4	col.5	colonne6	

	% Vi des masses cum.	base du trapèze	hauteur du trapèze	aire du trapèze
....	$\frac{100}{M} \sum_{j \leq i} N_j X_j$	$U_i - U(i-1)$	$\frac{V_i + V(i+1)}{2}$	base * hauteur
....	colonne7	colonne8	colonne9	colonne10

L'indice de Gini est égal à $1 - (\text{aire totale})/5000$, où "aire totale" représente la somme des coefficients de la dixième colonne de 'EDT1'.

Le schéma fonctionnel de 'GINI' est:



TEXTE DU PROGRAMME 'GINI' ET EXEMPLE D'UTILISATION.

=====

'GINI':(Checksum: # 27729d, Taille: 335.5 octets)

```

«  TOT  2  GET  EX*Y  0
  →  n  m  a
  «  0  0  0  0  0  0  0  0
    1  NE  FOR  i
      EDAT  i  { 1 } + GETI 3 ROLLD GET DUP2 *
      OVER 12 PICK + DUP 100 * n / ROT DUP
      11 PICK + DUP 100 * m / 4 PICK 15 PICK
      - OVER 13 PICK + 2 / DUP2 * DUP
      'a' STO+
    NEXT
    NE { 10 } + →ARRY 'EDT1' STO 8 DROPN
    1 a 5000 / -
  »
»

```

Exemple:

Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	10

, représentée dans EDAT par la matrice à deux colonnes: $\begin{bmatrix} 1 & 22 \\ 4 & 25 \\ 7 & 15 \\ 11 & 10 \end{bmatrix}$,

on obtient par exemple (en 2 secondes):

1: → 'GINI' → 1: .383489943952

COURBE DE LORENTZ D'UNE STATISTIQUE SIMPLE.

=====

'LRTZ' trace la courbe de Lorentz (ou de concentration, ou de Gini), de la statistique discrète contenue dans la variable EDAT. Dans le cas d'une statistique donnée par classes (et contenue dans la variable 'DATA'), il convient d'appeler tout d'abord le programme 'G→D', pour la transformer en statistique discrète.

Important:

Pour que le programme 'LRTZ' fonctionne, il faut préalablement avoir exécuté le programme 'GINI' (et ceci parce qu'on utilise le contenu du tableau 'EDT1').

Avec les notations du programme 'GINI', la courbe de Lorentz est la ligne polygonale, inscrite dans le carré [0,100] X [0,100], et joignant le point (0,0) au point (100,100), via les points (Ui,Vi). Elle est située en dessous de la première bissectrice.

Le tracé remplit tout l'écran de la HP48 ; ce qui signifie que l'échelle horizontale est "131 pixels=100 unités", et que l'échelle verticale est "64 pixels=100 unités". Ceci ne correspond pas à la tradition qui veut que la courbe de Gini soit visualisée dans un carré, mais permet une meilleure lisibilité à l'écran.

N.B: les cotés verticaux des trapèzes sont tracés.

Les coordonnées Ui, Vi des points nécessaires au tracé sont dans les colonnes 4 et 7 de 'EDT1'.

A la fin du tracé, on passe dans l'environnement graphique GRAPH. On revient à l'affichage de la pile en appuyant sur "ON".

N.B: Le programme 'LRTZ':

- * Passe dans le répertoire 'MATR', où il utilise 'GETC'.
- * Revient ensuite dans le répertoire 'STAT1'.

'LRTZ': (Checksum: # 5051d, Taille: 280.5 octets)

```

« (0,0) EDT1 MATR DUP 4 GETC SWAP 7 GETC STAT1
  R→C OBJ→ 1 GET →LIST +
  0 100 XRNG 0 100 YRNG ERASE
  { #0d #0d } PVIEW DRAI
  1 OVER SIZE 1 - FOR i
    DUP i 1 + GET OVER i GET OVER LINE
    DUP RE 0 R→C SWAP LINE
  NEXT DROP
  (0,0) (100,100) LINE GRAPH
»

```

Exemple: Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	10

représentée dans EDAT par la matrice à deux colonnes:
le tracé est effectué en 4 secondes.

```

[ [ 1 22 ]
  [ 4 25 ]
  [ 7 15 ]
  [ 11 10 ] ],

```

HISTOGRAMME D'UNE STATISTIQUE SIMPLE.

=====

'HIST' trace l'histogramme de la statistique groupée X (donnée par classes) qui est contenue dans la variable 'DATA'.

Le tracé est cadré de manière à occuper l'écran au maximum.

A la fin du tracé, on passe dans l'environnement graphique GRAPH. On revient à l'affichage de la pile en appuyant sur "ON".

'HIST': (Checksum: # 40551d, Taille: 249 octets)

```

«   DATA   EVAL
    1 OVER SIZE FOR i
      i DUP2 GET 4 PICK i GETI
      3 ROLLD GET SWAP - / PUT
    NEXT
    OVER 1 GET 0 R→C PMIN OVER
    DUP SIZE GET OVER OBJ→
    2 SWAP START MAX NEXT
    R→C PMAX ERASE { #0 #0 } PVIEW DRAX
    0 + 1 OVER SIZE 1 - FOR i
      DUP i GET 3 PICK
      i GETI 3 ROLLD GET
      0 R→C 3 ROLLD SWAP R→C BOX
    NEXT
    DROP2 GRAPH
»

```

Exemple:

Avec la statistique:

Xi	[0,2[	[2,6[	[6,8[	[8,14[
Ni	22	25	15	10

qui est représentée dans la variable 'DATA' par la liste:
{ { 0 2 6 8 14 } { 22 25 15 10 } },
le tracé est obtenu en 4 secondes.

POLYGONE DES FREQUENCES CUMULEES D'UNE STATISTIQUE SIMPLE.

=====

'PFC' trace le polygone des fréquences cumulées de la statistique groupée X (donnée par classes) qui est contenue dans la variable 'DATA'.

Le tracé est cadré de manière à occuper l'écran au maximum.

A la fin du tracé, on passe dans l'environnement graphique GRAPH. On revient à l'affichage de la pile en appuyant sur "ON".

N.B.: Le programme 'PFC' utilise le programme '→CUM'

'PFC':(Checksum: # 56583d, Taille: 204.5 octets)

```

“ DATA EVAL →CUM OVER 1 GET 0 R→C PMIN
1 2 START OVER DUP SIZE GET NEXT
R→C PMAX ERASE { #0 #0 } PVIEW
DRAX SWAP OBJ→ →ARRY
0 ROT + OBJ→ →ARRY R→C
2 OVER SIZE 1 GET FOR i
DUP i GET OVER i 1 - GET OVER LINE
DUP RE 0 R→C LINE
NEXT
DROP GRAPH
”

```

Exemple:

Avec la statistique:

Xi	[0,2[	[2,6[	[6,8[	[8,14[
Ni	22	25	15	10

qui est représentée dans la variable 'DATA' par la liste:
{ { 0 2 6 8 14 } { 22 25 15 10 } },
le tracé est obtenu en 4 secondes.

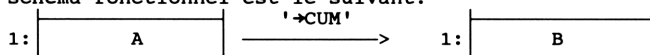
CREATION D'UN TABLEAU DE CUMULS.

=====

'→CUM' crée, à partir d'un tableau A, un tableau B de même format que A et formé des cumuls des éléments de A d'indice inférieur. A peut aussi bien être un vecteur qu'une matrice à une ou plusieurs colonnes. Les cumuls sont constitués dans l'ordre naturel des coefficients (de gauche à droite sur une ligne, et d'une ligne à la ligne située en dessous).

La possibilité de créer de tels tableaux est très utile dans l'étude des statistiques simples.

Le schéma fonctionnel est le suivant:



Remarque: '→CUM' peut également cumuler les éléments d'une liste.

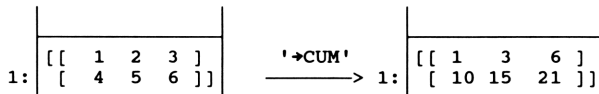
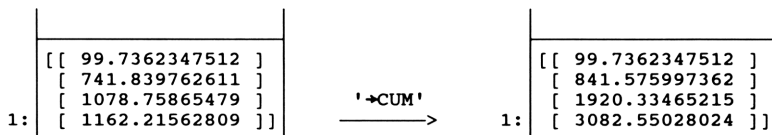
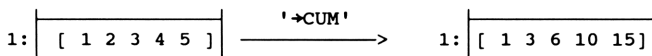
'→CUM': (Checksum: # 20274d, Taille: 92 octets)

```

« 2 OVER SIZE { } + OBJ>
  IF 2 == THEN * END
  FOR i
    i DUP2 1 - GET 3 PICK i GET + PUT
  NEXT
»

```

Exemples:



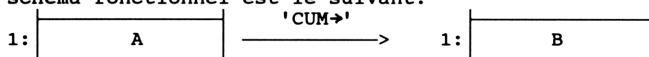
RETOUR D'UN TABLEAU DE CUMULS AU TABLEAU INITIAL.

=====

'CUM→' permet de 'décumuler' un tableau, c'est-à-dire d'effectuer l'opération inverse de celle de '→CUM'.

Si le programme '→CUM' permet de créer un tableau de cumuls, le programme 'CUM→' permet de revenir au tableau initial.

Le schéma fonctionnel est le suivant:



Ici A est le tableau initial (matrice ou vecteur) et B est le tableau final (A représente donc le tableau des cumuls de B).

Remarque: CUM→ peut également "dé-cumuler" une liste.

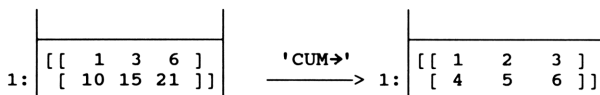
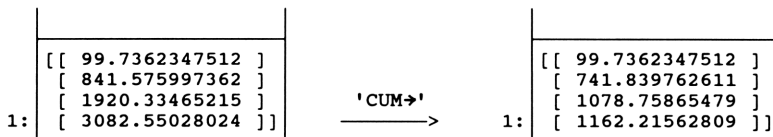
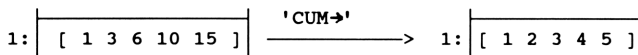
'CUM→': (Checksum: # 10763d, Taille: 94.5 octets)

```

«   DUP   SIZE   { }   +   OBJ→
   IF   2   ==   THEN   *   END
   2   FOR   i
       i   DUP2   GET   3   PICK   i   1   -   GET   -   PUT
   -1   STEP
»

```

Exemples:



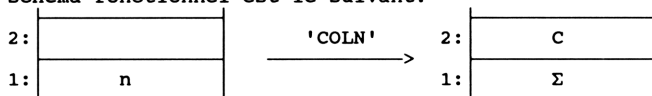
EXTRACTION D'UNE DES COLONNES DE **ΣDT1**.

=====

'COL.N' extrait la colonne de numéro n de la matrice 'EDT1' du répertoire 'STAT1'.

Le résultat est obtenu sous forme d'une matrice colonne C. On obtient également, au niveau 1 de la pile, la somme Σ des termes de la colonne extraite.

Le schéma fonctionnel est le suivant:



N.B.: le programme 'COL.N' appelle le programme 'GETC' du répertoire 'MATR'.

Le programme 'COL.N' est conçu pour être un complément aux programmes 'MDLE' et 'GINI'. En effet le calcul de la médiale et de l'indice de Gini d'une statistique discrète par ces deux programmes s'accompagne du chargement dans la variable 'EDT1' du tableau des calculs intermédiaires. Il est alors très utile de pouvoir consulter ce tableau colonne par colonne ainsi que de connaître la somme de ces colonnes.

```
'COL.N':( Checksum: # 15970d, Taille: 75.5 octets )
"      EDT1  SWAP  MATR  GETC  STAT1  DUP  OBJ→
      1   GET   2   SWAP  START  +  NEXT
"
```

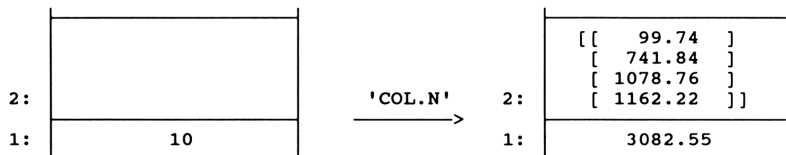
Exemple:

Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	10

```
[ [ 1 22 ]
  [ 4 25 ]
  [ 7 15 ]
```

représentée dans Σ DAT par la matrice à deux colonnes: $\begin{bmatrix} 11 & 10 \end{bmatrix}$,
Si on appelle le programme 'GINI', le tableau des calculs intermédiaires est placé dans 'EDT1'. On peut obtenir: (mode 2 FIX)



On trouve ainsi la valeur des aires des trapèzes qui interviennent dans la construction de la courbe de Gini. Le calcul de l'indice de Gini s'ensuit immédiatement:

$$I \approx 1 - 3082.55/5000 \approx 0.38$$

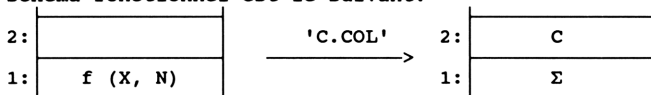
CREATION D'UNE COLONNE FONCTION DE CELLES DE ΣDAT

=====

Lamatrice ΣDAT doit contenir les deux colonnes représentatives d'une statistique discrète (la première colonne pour les valeurs du caractère, ou pour les centres de classes, la seconde pour les effectifs).

'C.COL' permet de calculer une colonne fonction des deux colonnes de ΣDAT.

Le schéma fonctionnel est le suivant:



où f(X,N) est la fonction utilisée (expression algébrique ou programme), X et N désignant respectivement les éléments de la première et de la colonne de ΣDAT (majuscules obligatoires).

où C est la matrice colonne obtenue et où Σ est la somme des coefficients de la colonne C.

Le programme 'C.COL' est très utile dans l'étude des statistiques simples. Il permet par exemple de créer colonne par colonne, très rapidement, les tableaux de calculs des moments de la statistique (calcul de la variance, etc...).

'C.COL':(Checksum: # 3810d, Taille: 176 octets)

```

" 0 → p s
  " p ΣDAT OBJ→ DROP
    NE 2 * 1 - NE FOR i
      'N' STO 'X' STO p EVAL DUP
      's' STO+ i ROLL
    -1 STEP
    NE { 1 } + →ARRY SWAP DROP s
  "
  { X N } PURGE

```

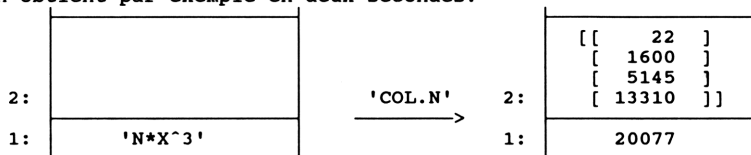
Exemple: Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	10

[[1 22]
[4 25]
[7 15]
[11 10]],

représentée dans ΣDAT par la matrice à deux colonnes:

On obtient par exemple en deux secondes:

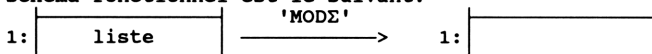


MODIFICATION DES COLONNES DE ΣDAT.

=====

La matrice ΣDAT doit en principe contenir les deux colonnes représentatives d'une statistique discrète (la première colonne pour les valeurs du caractère, ou pour les centres de classes, la seconde pour les effectifs). 'MODE' permet de modifier l'une ou l'autre (ou les deux) des colonnes de ΣDAT.

Le schéma fonctionnel est le suivant:



où "liste" est une liste ayant le format { prog1 prog2 } avec:

prog1: programme s'appliquant à la première colonne.

prog2: programme s'appliquant à la seconde colonne.

prog1 et prog2 doivent être écrits avec la logique polonaise inverse et donner un résultat numérique fonction de l'élément présent sur la pile. Si on veut que la première colonne reste inchangée, il suffit de poser prog1 = « ».

Si prog2 est absent de la liste la deuxième colonne est inchangée.

Le contenu de ΣDAT est sauvegardé dans 'ΣDT1'.

'MODE':(Checksum: # 313d, Taille: 152.5 octets)

```

«   ΣDAT 'ΣDT1' STO OBJ→
    → n
    «   'ΣDAT'
        1  NE   FOR   j
          j   { 1 } +
          n   1  FOR   i
            DUP2 GET  3  i  +  PICK  EVAL  PUTI
            -1 STEP
            DROP
        NEXT
        n  1  +  DROPN
    »
»

```

Exemple: Avec la statistique :

Xi	55	65	75	85
Ni	220	250	150	100

représentée dans ΣDAT par la matrice à deux colonnes:
On place { « 5 / 14 - » « 10 / » } au niveau 1 de la pile et on appelle 'MODE'. Au bout de 2 à 3 s, la matrice
ΣDAT contient:
et le contenu précédent de ΣDAT est sauvegardé dans ΣDT1.

```

[ [ 55 220 ]
  [ 65 250 ]
  [ 75 150 ]
  [ 85 100 ] ],
[ [ -3  22 ]
  [ -1  25 ]
  [  1  15 ]
  [  3  10 ] ]

```

Remarque: 'MODE' est utile pour changer d'échelle et/ou d'origine.

STATISTIQUES DOUBLES

Le répertoire 'STAT2' contient un certain nombre de programmes permettant d'étudier les statistiques doubles discrètes (X,Y) , chaque valeur (X_i,Y_j) étant connue par son effectif ou par sa fréquence. On notera N_{ij} cet effectif (ou cette fréquence).

Les différentes valeurs de X (première statistique marginale du couple (X,Y)) doivent être placées, sous forme d'un vecteur, dans une variable nommée ' X '.

De même, les différentes valeurs de Y (deuxième statistique marginale du couple (X,Y)) doivent être placées, sous forme d'un vecteur, dans une variable nommée ' Y '.

Le tableau des effectifs N_{ij} doit être placé dans une variable ' ΣEFF ', sous forme de matrice, avec les conventions suivantes:

La statistique X est donc celle qui figure en première colonne. Les valeurs de Y figurent donc sur la 1ère ligne du tableau. La matrice ' ΣEFF ' contient le tableau à n lignes et p colonnes de terme général N_{ij} , effectif de la valeur (X_i,Y_j) du couple (X,Y) .

	Y1	Y2	Y3		Yp
X1	N1,1	N1,2	N1,3	...	N1,p
X2	N2,1	N2,2	N2,3	...	N2,p
..					
Xn	Nn,1	Nn,2	Nn,3	...	Nn,p

Remarque:

Il est courant que l'on ait à étudier une statistique double (X,Y) consistant en un ensemble de points (X_i,Y_i) tous d'effectif 1, c'est-à-dire se présentant sous la forme:

X	X1	X2	X3		Xn
Y	Y1	Y2	Y3		Yn

condition de placer dans ' ΣEFF ' la matrice identité d'ordre n .

Un certain nombre de programmes du répertoire 'STAT2' s'appliquent bien à ce cas simple. Néanmoins, les programmes utilisant ' ΣEFF ' pourront encore être utilisés à

Les programmes du répertoire 'STAT2' sont les suivants:

'EFFX' : effectifs de la première statistique marginale.
'EFFY' : effectifs de la deuxième statistique marginale.
'MX' : moyenne arithmétique de la première statistique marginale.
'MY' : moyenne arithmétique de la deuxième statistique marginale.
'VX' : variance de la première statistique marginale.
'VY' : variance de la deuxième statistique marginale.
'NXY' : création de la matrice des $N_{ij} \cdot X_i \cdot Y_j$, très utile pour le calcul de la covariance.
'CV' : calcul de la covariance du couple (X,Y).
'COR' : coefficient de corrélation linéaire du couple (X,Y).
'DX→Y' : équation de la droite de régression de X en Y.
'DY→X' : équation de la droite de régression de Y en X.
'KX↑A' : ajustement par une fonction puissance $Y = k \cdot X^a$.
'KA↑X' : ajustement par une fonction exponentielle $Y = k \cdot a^X$.
'TOTAL' : somme des coefficients d'un vecteur ou d'une matrice.
'MULT' : produit terme à terme des coefficients de 2 vecteurs.
(utile pour rechercher une variance ou une covariance)
'MODT' : modification, selon une certaine formule, des termes d'un tableau (matrice ou vecteur). 'MODT' est utile dans la recherche d'ajustements au moyen de fonctions puissances ou exponentielles.

Les programmes 'TOTAL', 'MULT', 'MODT', sont destinés à permettre à l'utilisateur de mener les calculs "pas à pas". Ils peuvent s'avérer utiles dans d'autres répertoires que 'STAT2'.

EFFECTIFS DES STATISTIQUES MARGINALES.

=====

'EFFX' et 'EFFY' permettent de connaître les effectifs respectifs des statistiques marginales X et Y. Ce résultat est obtenu par sommation:

ligne par ligne pour le calcul des effectifs de X.
colonne par colonne pour le calcul des effectifs de Y.
(voir signification de la matrice 'ΣEFF' dans l'introduction).

Le résultat est obtenu sous forme d'un vecteur au niveau 1:

1: 'EFFX' ou 'EFFY' → 1: vecteur

'EFFX': (Checksum: # 28201d, Taille: 46 octets)
" ΣEFF DUP SIZE 2 2 SUB 1 CON * "

'EFFY': (Checksum: # 21169d, Taille: 48.5 octets)
" ΣEFF TRN DUP SIZE 2 2 SUB 1 CON * "

Exemple:

Avec la statistique:

On place dans 'ΣEFF'

la matrice: $\begin{bmatrix} 1 & 7 & 11 & 12 \\ 3 & 5 & 0 & 1 \\ 0 & 4 & 8 & 8 \end{bmatrix}$,

X\Y	10	20	30	40
1	1	7	11	12
5	3	5	0	1
10	0	4	8	8

dans 'X' le vecteur [1 5 10],
et dans 'Y' le vecteur [10 20 30 40].
On obtient alors: (en une seconde)

1: 'EFFX' → 1: [31 9 20]

1: 'EFFY' → 1: [4 16 19 21]

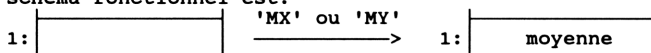
MOYENNES ARITHMETIQUES DES STATISTIQUES MARGINALES.

=====

'MX' et 'MY' permettent de connaître les moyennes arithmétiques respectives des statistiques marginales X et Y.

(voir signification de 'ΣEFF' dans l'introduction).

Le schéma fonctionnel est:



N.B: le programme 'MX' appelle 'EFFX' et 'TOTAL'. De même 'MY' appelle 'EFFY' et 'TOTAL'.

'MX': (Checksum: # 47044d, Taille: 44.5 octets)

“ EFFX DUP TOTAL / X DOT ”

'MY': (Checksum: # 17541d, Taille: 44.5 octets)

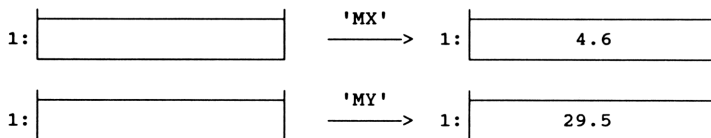
“ EFFY DUP TOTAL / Y DOT ”

Exemple:

Avec la statistique:		X\Y	10	20	30	40
			1	7	11	12
On place dans 'ΣEFF'		5	3	5	0	1
la matrice: [[[1 7 11 12]		10	0	4	8	8
[3 5 0 1]						
[0 4 8 8]],						

dans 'X' le vecteur [1 5 10],
et dans 'Y' le vecteur [10 20 30 40].

On obtient alors: (en une seconde)



Les moyennes arithmétiques de X et Y sont donc $\bar{X} = 4.6$ et $\bar{Y} = 29.5$

Remarque: dans le cas d'une statistique double (X,Y) consistant en un ens. de points d'effectif 1:

X	X1	X2	X3		Xn
Y	Y1	Y2	Y3		Yn

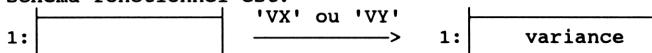
on peut encore utiliser 'MX' et 'MY', à condition de mettre la matrice identité d'ordre n dans 'ΣEFF'. Le plus simple est tout même d'appliquer le programme 'TOTAL' à X et à Y, et de diviser par n.

VARIANCES DES STATISTIQUES MARGINALES.

=====

'VX' et 'VY' permettent de connaître les variances respectives des statistiques marginales X et Y.

(voir signification de la matrice 'SEFF' dans l'introduction).
Le schéma fonctionnel est:



N.B.: le programme 'VX' appelle 'EFFX', 'TOTAL', 'MULT' et 'MX'.
De même 'VY' appelle 'EFFY', 'TOTAL', 'MULT' et 'MY'.

'VX':(Checksum: # 49445d, Taille: 65 octets)
" EFFX DUP TOTAL / X DUP MULT DOT MX SQ - "

'VY':(Checksum: # 27450d, Taille: 65 octets)
" EFFY DUP TOTAL / Y DUP MULT DOT MY SQ - "

Exemple:

Avec la statistique:

On place dans 'SEFF'

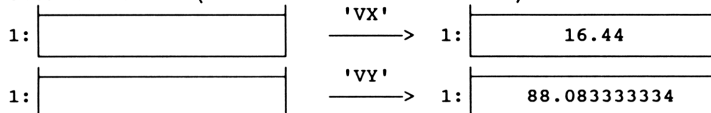
la matrice: $\begin{bmatrix} 1 & 7 & 11 & 12 \\ 3 & 5 & 0 & 1 \\ 0 & 4 & 8 & 8 \end{bmatrix}$,

X\Y	10	20	30	40
1	1	7	11	12
5	3	5	0	1
10	0	4	8	8

dans 'X' le vecteur [1 5 10],

et dans 'Y' le vecteur [10 20 30 40].

On obtient alors: (en moins de deux secondes)



Les variances de X et Y sont donc $\text{var}(X) = 16.44$ et $\text{var}(Y) \approx 88.08$; on en déduit leurs écarts-types: $\sigma(X) = \sqrt{16.44} \approx 4.05$, et $\sigma(Y) \approx 9.39$.

Remarque: dans le cas d'une statistique double (X,Y) consistant en un ens. de points d'effectif 1:

on peut encore utiliser 'MX' et 'MY', à condition de mettre la matrice identité d'ordre n dans 'SEFF'. Le plus simple est tout

même d'utiliser les programmes 'MODT' (pour calculer les vecteurs des X_i^2 et celui des Y_i^2) et 'TOTAL', en se basant sur la formule de Huyghens:

X	X1	X2	X3		Xn
Y	Y1	Y2	Y3		Yn

$$\text{var}(X) = \overline{X^2} - (\bar{X})^2 \quad (\bar{X} = \text{moyenne arithmétique de } X).$$

CREATION DE LA MATRICE DE TERME GENERAL $N_{ij} \cdot X_i \cdot Y_j$.

=====

'NXY' constitue la matrice de même format que 'ΣEFF' (voir l'introduction pour les conventions utilisées) et de terme général $N_{ij}X_iY_j$.

Le résultat est placé dans une variable nommée 'ΣNXY'.

La pile n'est pas concernée par le programme 'NXY'.

'NXY':(Checksum: # 18989d, Taille: 181.5 octets)

```

«   ΣEFF   SIZE
  → d
  «   1   d   1   GET   FOR   i
      1   d   2   GET   FOR   j
      'ΣEFF(i,j)*X(i)*Y(j)'   EVAL
      NEXT
      NEXT
      d   →ARRY   'ΣNXY'   STO
  »
»

```

Exemple:

Avec la statistique:

On place dans 'ΣEFF'

la matrice: $\begin{bmatrix} 1 & 7 & 11 & 12 \\ 3 & 5 & 0 & 1 \\ 0 & 4 & 8 & 8 \end{bmatrix}$,

dans 'X' le vecteur $[1 \ 5 \ 10]$,

et dans 'Y' le vecteur $[10 \ 20 \ 30 \ 40]$.

X\Y	10	20	30	40
1	1	7	11	12
5	3	5	0	1
10	0	4	8	8

Le programme 'NXY' place alors, en 3 à 4 secondes, la matrice:

$\begin{bmatrix} 10 & 140 & 330 & 480 \\ 150 & 500 & 0 & 200 \\ 0 & 800 & 2400 & 3200 \end{bmatrix}$ dans 'ΣNXY'. On a par exemple:
'ΣNXY(2,1)=150' car $X(2)=5$, $Y(1)=10$
et 'ΣEFF(2,1)'=3.

Remarque: dans le cas d'une statistique double (X,Y) consistant en un ens. de points d'effectif 1:

on peut encore utiliser 'NXY', à condition de mettre la matrice identité d'ordre n dans 'ΣEFF', mais c'est sans grand intérêt:

le programme 'MULT', appliqué aux vecteurs 'X' et 'Y', est en effet tout désigné dans ce cas (car il fournit le vecteur des X_iY_i).

X	X1	X2	X3		Xn
Y	Y1	Y2	Y3		Yn

COVARIANCE ET COEFFICIENT DE CORRELATION LINEAIRE.

=====

'CV' calcule la covariance $\text{cov}(X,Y)$ de la statistique double (X,Y) .
'COV' calcule leur coefficient de corrélation linéaire $\text{ro}(X,Y)$.

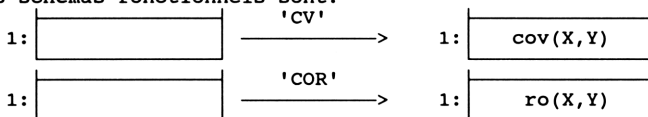
Rappelons que:

$$\text{cov}(X,Y) = \frac{1}{N} \sum N_{ij} (X_i - \bar{X})(Y_j - \bar{Y}) = \frac{1}{N} \sum N_{ij} X_i Y_j - \bar{X} \bar{Y}$$

et $\text{ro}(X,Y) = \frac{\text{cov}(X,Y)}{\sigma(X) \sigma(Y)}$

où $\bar{X}$, $\bar{Y}$ désignent les moyennes arithmétiques de X et Y et $\sigma(X)$, $\sigma(Y)$ leurs écarts-types.

Les schémas fonctionnels sont:



N.B: 'CV' appelle les programmes 'EFFX', 'MX', 'MY', 'TOTAL'.
'COR' appelle les programmes 'CV', 'VY', 'VX'.

'CV':(Checksum: # 29645d, Taille: 72.5 octets)

“ X ΣEFF Y * DOT EFFX TOTAL / MX MY * - ”

'COR':(Checksum: # 20309d, Taille: 44 octets)

“ CV VY ✓ / VX ✓ / ”

Exemple:

Avec la statistique:

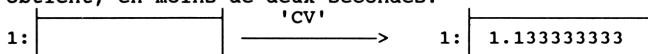
On place dans 'ΣEFF'

la matrice: $\begin{bmatrix} 1 & 7 & 11 & 12 \\ 3 & 5 & 0 & 1 \\ 0 & 4 & 8 & 8 \end{bmatrix}$,

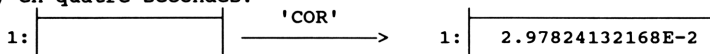
X\Y	10	20	30	40
1	1	7	11	12
5	3	5	0	1
10	0	4	8	8

dans 'X' le vecteur [1 5 10],
et dans 'Y' le vecteur [10 20 30 40].

On obtient, en moins de deux secondes:



et, en quatre secondes:



(X et Y sont donc ici faiblement corrélées).

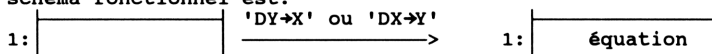
DROITE DE REGRESSION LINEAIRE

"DE Y EN X" ET "DE X EN Y".

=====

'DY→X' et 'DX→Y' calculent respectivement les équations des droites de régression linéaire "de Y en X" et "de X en Y" (appelées encore droites des moindres carrés) de la statistique double (X,Y).

Le schéma fonctionnel est:



où "equation" est l'équation cherchée, sous la forme:

'y=a*x+b' dans le cas du programme 'DY→X',

'x=a*y+b' dans le cas du programme 'DX→Y'.

Rappelons l'équation de la droite des moindres carrés de Y en X:

$$y = \frac{\text{cov}(X,Y)}{\text{var}(X)} (x - \bar{X}) + \bar{Y}$$

et celle des moindres carrés de X en Y est: $x = \frac{\text{cov}(X,Y)}{\text{var}(Y)} (y - \bar{Y}) + \bar{X}$

(avec les notations habituelles).

N.B: 'DY→X' appelle 'CV', 'VX', 'MY', 'MX';

'DX→Y' appelle 'CV', 'VY', 'MY', 'MX'.

'DY→X': (Checksum: # 56991d, Taille: 82 octets)

```

"  'Y'  CV  VX  /  DUP  'X'  *
  MY  MX  4  ROLL  *  -  +  =
"
  
```

'DX→Y': (Checksum: # 33808d, Taille: 82 octets)

```

"  'X'  CV  VY  /  DUP  'Y'  *
  MX  MY  4  ROLL  *  -  +  =
"
  
```

Exemple:

Avec la statistique:

On place dans 'ΣEFF'

la matrice: $\begin{bmatrix} 1 & 7 & 11 & 12 \\ 3 & 5 & 0 & 1 \\ 0 & 4 & 8 & 8 \end{bmatrix}$,

X\Y	10	20	30	40
1	1	7	11	12
5	3	5	0	1
10	0	4	8	8

[1 5 10] dans 'X', [10 20 30 40] dans 'Y'.

On obtient, en quatre secondes, en mode 3 FIX:

```

1: [ ] --'DY→X'--> 1: [ 'y=0.069*x+29.183' ]
  
```

et:

```

1: [ ] --'DX→Y'--> 1: [ 'x=0.013*y+4.220' ]
  
```

AJUSTEMENT PAR UNE FONCTION

'Y=k*a^X' ou 'Y=k*X^a'

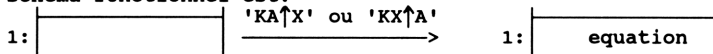
=====

'KA↑X' et 'KX↑A' calculent l'ajustement de la statistique double (X,Y) par , respectivement:

Une fonction exponentielle $y=k*a^x$. (programme 'KA↑X')

Une fonction puissance $y=k*x^a$. (programme 'KX↑A')

Le schéma fonctionnel est:



où "equation" est l'équation de la courbe cherchée, sous la forme:

' $y=k*a^x$ ' dans le cas du programme 'KA↑X',

' $y=k*x^a$ ' dans le cas du programme 'KX↑A'.

N.B: 'KA↑X' et 'KX↑A' appellent: 'MODT', 'CV', 'VX', 'MX', et 'MY'.

Important: Pendant le fonctionnement de 'KX↑A', le contenu de X et de Y est modifié (seul Y est modifié dans 'KA↑X'). les variables X et Y retrouvent leurs valeurs initiales à la fin du programme.

On veillera donc à ne pas arrêter le fonctionnement du programme par 'ON' avant qu'il ne soit réellement terminé. Dans le cas contraire, les valeurs initiales de X et de Y sont placées sur la pile.

```

'KA↑X':( Checksum: # 35728d, Taille: 159 octets )
"  Y  DUP  « LN »  MODT  'Y'  STO  CV  VX  /  EXP
  →  a
  «  'Y'  a  MX  NEG  ^  MY  EXP  *  a  'x'  ^  *  =
    SWAP  'Y'  STO
  »
»
  
```

```

'KX↑A':( Checksum: # 14153d, Taille: 207.5 octets )
"  X  Y  DUP2
  « LN »  MODT  'Y'  STO  « LN »  MODT  'X'  STO
  CV  VX  /
  →  a
  «  'Y'  MY  a  MX  *  -  EXP  'x'  a  ^  *  =
    3  ROLLD  'Y'  STO  'X'  STO
  »
»
  
```

Exemple: avec les données utilisées dans l'exemple décrivant le fonctionnement de 'DX→Y' et de 'DY→X', on trouve:
(en 4 secondes, en mode 3 FIX)

```

1: [ ] --'KA↑X'--> 1: [ 'y=27.239*1.003^x' ]
  
```

et:

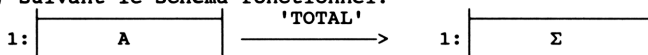
```

1: [ ] --'KX↑A'--> 1: [ 'y=28.360*x^(-0.024)' ]
  
```

SOMME DES COEFFICIENTS D'UN TABLEAU.

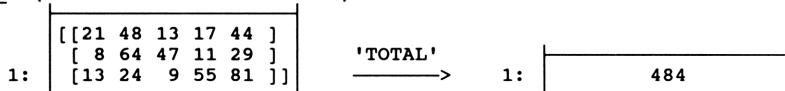
=====

'TOTAL' calcule la somme Σ des termes du tableau A (vecteur ou matrice) suivant le schéma fonctionnel:



```
'TOTAL':( Checksum: # 7618d, Taille: 52 octets )
«  OBJ→  OBJ→  IF  2  ==  THEN  *  END
  2  SWAP  START  +  NEXT
»
```

Exemple: (en moins d'une seconde)

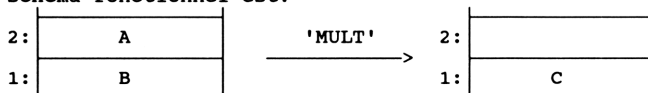


PRODUIT TERME A TERME DE DEUX VECTEURS.

=====

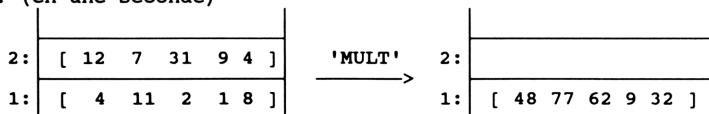
'MULT' effectue le produit terme à terme de deux vecteurs A=[X1,..Xn] et B=[Y1,..Yn]. Le résultat est le vecteur C=[X1*Y1, X2*Y2,..., Xn*Yn].

Le schéma fonctionnel est:



```
'MULT':( Checksum: # 27673d, Taille: 121 octets )
«  DUP  SIZE  1  GET
  →  a  b  n
  «  1  n  FOR  i  'a(i)*b(i)'  EVAL  NEXT
    n  →ARRAY
  »
»
```

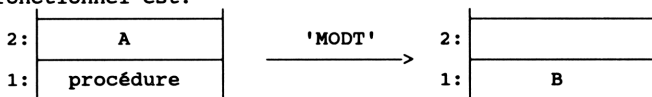
Exemple: (en une seconde)



APPLICATION D'UNE MEME PROCEDURE AUX COEFFICIENTS D'UN TABLEAU.

=====

'MODT' permet de modifier les coefficients d'un tableau A en leur appliquant une même transformation. Si B est le tableau obtenu, le schéma fonctionnel est:



où "procédure" est un programme destiné à s'appliquer à chacun des éléments de A. Ce programme est censé s'appliquer à l'élément figurant au niveau 1 de la pile, et retourner un résultat numérique à ce même niveau.

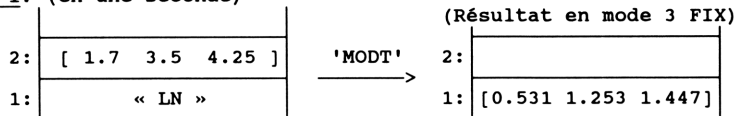
'MODT': (Checksum: # 48215d, Taille: 110 octets)

```

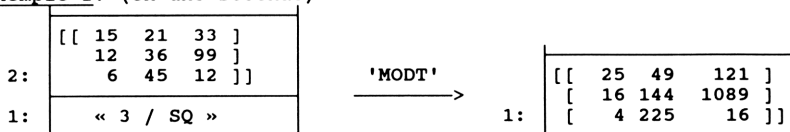
"      → p
"      " OBJ→ DUP → d
"      " OBJ→ IF 2 == THEN * END
"      " → n
"      " 1 n START p EVAL n ROLLD NEXT
"      " d →ARRY
"
"
"
"
"
"
"
"

```

Exemple 1: (en une seconde)



Exemple 2: (en une seconde)



(tous les coefficients ont été divisés par 3 puis élevés au carré).

Remarque: 'MODT' est utile lorsqu'il faut rechercher entre deux statistiques X et Y une relation de type $Y = K \cdot X^\alpha$ ou $Y = K \cdot A^X$. On est alors amené à transformer, au moyen du programme 'MODT', le vecteur des X_i et/ou des Y_i en celui des $LN(X_i)$ (des $LN(Y_i)$) et à rechercher une corrélation linéaire entre ces deux nouvelles variables (programmes 'CV', 'COR', 'DY→X', etc...).

BASES DE DONNEES

La capacité mémoire importante de la HP48SX permet d'envisager la gestion de bases de données. Bien sûr, il ne sera pas possible de rentrer un nombre grandiose d'enregistrements. Mais, pour une calculatrice de poche, le résultat obtenu est très impressionnant.

Une base de données est un ensemble d'enregistrements (ayant en général certains rapports entre eux ...) sur lequel on peut effectuer l'une des opérations suivantes:

- ajout d'un enregistrement.
- modification d'un enregistrement.
- suppression d'un enregistrement.
- lecture de l'ensemble des enregistrements.
- tri de l'ensemble des enregistrements.
- sélection d'enregistrements.

Dans la mémoire de la HP48S, une base de données se présentera sous la forme d'une liste ayant la forme suivante:

{ C1 C2 C3 Cn } ,

où n est un entier représentant le nombre d'éléments de la liste et où C1 , C2 ,, Cn sont les différents enregistrements.

Un enregistrement est une chaîne de caractères.

Pour être totalement visible à l'écran, cette chaîne doit à priori être formée d'au plus 4 lignes, chaque ligne ayant au plus 22 caractères (On peut insérer des sauts de lignes par NEWLINE).

Néanmoins on verra qu'il est possible par l'option VISIT du programme 'LIRE' d'utiliser des chaînes plus larges.

Une base de données sera placée dans une variable, et l'accès à la base de données se fera en utilisant ce nom de variable.

Les programmes du répertoire 'DATA' sont les suivants:

'AJOUT': ajouter un enregistrement.

'LIRE' : lecture de la base de données, avec possibilité de modifier ou d'effacer des enregistrements.

'TRI' : tri de la base de données.

'FIND' : recherche du premier enregistrement contenant une sous-chaîne donnée (et lecture de la base de données à partir de celui-ci).

'SELEC': sélection des enregistrements contenant une sous-chaîne donnée, puis lecture de la base de données créée avec

AJOUT D'UN ENREGISTREMENT.

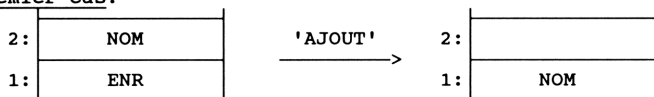
=====

'AJOUT' permet d'ajouter un enregistrement à une base de données existante, ou de créer une nouvelle base de données en y entrant un nouvel enregistrement.

Il y a 2 schémas fonctionnels possibles:

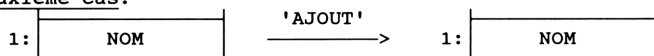
NOM désigne le nom de la base de données à compléter ou à créer.

Premier cas:



Ici ENR est la chaîne de caractères représentant le nouvel enregistrement.

Deuxième cas:



Ici un message "Nouvel Objet:" est affiché pour vous permettre de rentrer le nouvel enregistrement.

Le clavier est alors placé en mode Alpha: l'objet que vous validez par ENTER est transformé en chaîne (vous n'avez donc pas à taper les ").

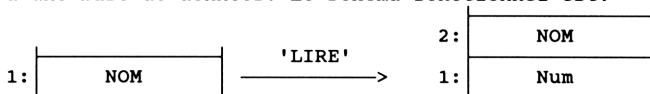
Si la base de données est correctement triée, alors le nouvel enregistrement s'insère dans la base de données de telle manière que celle-ci soit encore triée.

```
'AJOUT': ( Checksum: # d, Taille: octets )
« IF DUP TYPE 2 ≠ THEN
  "Nouvel Objet:" { "" α } INPUT
END
→ c
« IFERR DUP RCL THEN
  DROP c 1 →LIST
ELSE
  "" SWAP OBJ→ → n
  « c 1
  WHILE
    DUP n ≤ OVER 3 + PICK c > AND
    REPEAT 1 +
  END
  ROLLD n 1 + →LIST SWAP DROP
  »
END OVER STO
» »
```


LECTURE ET MODIFICATION DE LA BASE DE DONNEES.

=====

'LIRE' permet de consulter, et éventuellement de modifier le contenu d'une base de données. Le schéma fonctionnel est:



où NOM est le nom de la base de données, et où Num est le numéro d'ordre de l'enregistrement sur lequel s'est arrêtée la lecture (On peut alors faire **GET** pour placer l'enregistrement sur la pile).

N.B: On peut également appeler 'LIRE' en plaçant le nom de la base de données au niveau 2 et un numéro d'ordre au niveau 1. La lecture commence alors par l'enregistrement correspondant à ce numéro d'ordre (C'est intéressant pour les bases de données importantes, si on veut accéder rapidement à un enregistrement assez éloigné du début)

Dès l'appel du programme 'LIRE', celui-ci est suspendu, le contenu de l'enregistrement apparaît à l'écran et le menu suivant est affiché.

→	←	→→	VISIT	EFFAC	OK
---	---	----	-------	-------	----

- 1) Un appui sur la touche "→" affiche l'enregistrement suivant.
Si on est sur le dernier enregistrement, on revient au début.
- 2) Un appui sur "←" affiche l'enregistrement précédent.
Si on est sur le premier enregistrement, on passe à la fin.
- 3) Un appui sur "→→" provoque un défilement automatique. Quand on arrive en fin de base de données, le défilement continue depuis le début. Le défilement est interrompu par l'appui sur une touche quelconque (éviter ON).
- 4) Un appui sur "VISIT" provoque le passage de l'enregistrement en ligne de commande. Il peut ainsi être consulté (notamment si ses dimensions ne lui permettent pas d'être entièrement visible à l'écran).
On peut en profiter pour modifier cet enregistrement.
Cette consultation/modification prend fin par ENTER.
S'il y a eu modification, le programme en demande confirmation par O (oui) ou N (non).
Si vous confirmez, l'enregistrement ainsi modifié prend la place de l'ancien.
Sinon la modification est annulée.
Dans les 2 cas, on revient en lecture de la base de données.

- 5) Un appui sur "EFFAC" provoque l'effacement de l'enregistrement en cours de lecture. Toutefois, pour éviter une erreur, le programme demande une confirmation par O (oui) ou N (non). Dans les 2 cas, on revient en lecture de la base de données.
- 6) Un appui sur "OK" termine la lecture. On a alors le nom de la base au niveau 2, et le N° d'enregistrement au niveau 1. Un nouvel appel de 'LIRE' permettrait donc de reprendre la lecture du point exact où on l'a quittée.

Remarques:

Si on efface le dernier enregistrement d'une base de données, celle-ci est purgée du répertoire.
Dans le défilement automatique, on utilise une temporisation de 0.2 secondes (voir instruction .2 WAIT ci-dessous, que vous pouvez modifier).

```
'LIRE': ( Checksum: # 44705d, Taille: 669 octets )
« IF DUP TYPE 6 == THEN 1 END OVER RCL SIZE
→ i n
« « i + n MOD 1 + 'i' STO DUP
i GET CLLCD 1 DISP 3 FREEZE
»
« "Confirmation ? (O/N) : " { "" α }
INPUT "O" SAME
» → ii ok
« -1 ii EVAL
{ { "→" « 0 ii EVAL » }
{ "←" « -2 ii EVAL » }
{ "↔"
« DO 0 ii EVAL .2 WAIT
UNTIL KEY END DROP
» }
{ "VISIT"
« DUP i DUP2 GET "" OVER
{ -1 } + INPUT
IF DUP ROT SAME THEN 0
ELSE ok EVAL
END
IF THEN PUT ELSE 3 DROPN END
-1 ii EVAL
» }
{ "EFFAC"
« IF ok EVAL THEN
IF n 1 > THEN
DUP RCL DUP 1 i 1 - SUB
SWAP i 1 + OVER SIZE SUB
+ OVER STO 'n' 1 STO-
ELSE PURGE 2 MENU KILL
END
END -1 ii EVAL
» }
{ "OK" « CONT » }
} TMENU HALT 2 MENU i
» » »
```

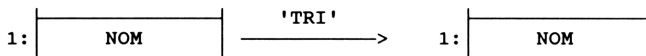
TRI DE LA BASE DE DONNEES.

=====

'TRI'effectue le tri d'une base de données. Celui-ci s'effectue de manière alphabétique croissante (plus précisément suivant les codes ASCII croissants).

Le tri ne s'impose que si on a modifié certains enregistrements (option VISIT dans le programme 'LIRE'), détruisant l'ordre qui s'établit automatiquement si on n'utilise que 'AJOUT'.

Le schéma fonctionnel est simple:



où NOM représente le nom de la base de données.

'TRI': (Checksum: # 28921d, Taille: 141.5 octets)

```

«   DUP   RCL   OBJ→
    →   n
    «   IF   n   1   >
        THEN
            2   n   FOR   i
                i   ROLL   i   1   +
                WHILE   DUP   PICK   3   PICK   <
                REPEAT   1   -   END
                1   -   ROLL   D
            NEXT
        END
        n   →LIST   OVER   STO
    »
  
```

Remarque: Il n'est pas possible de donner d'indication de durée de la procédure de tri. Celle-ci est tant fonction de l'importance de la base de données que de son degré de "désordre". Une base de 50 enregistrements peut ainsi être triée en environ 20 secondes.

RECHERCHE D'UN ENREGISTREMENT.

=====

'FIND' effectue une recherche séquentielle, dans une base de données, du premier enregistrement contenant une sous-chaine donnée.

Il y a 2 schémas fonctionnels possibles:

NOM désigne le nom de la base de données.

Premier cas:

2:	NOM	Ici <u>CHN est la chaîne de caractères</u> <u>à rechercher.</u>
1:	CHN	

Deuxième cas:

1:	NOM	Dans ce cas un message "Chercher le premier: " vous invite à donner l'objet à chercher (et qui est automatiquement transformé en chaîne) Le clavier est en mode Alpha: Vous entrez l'objet cherché (pas besoin de ") et vous validez par ENTER).
----	-----	--

Le message "En cours..." est affiché pendant la recherche.

Si la sous-chaine est trouvée dans un enregistrement, alors on passe en lecture de la base de données, à partir de celui-ci.

Sinon un message "Pas d'occurrence" est affiché.

'FIND': (Checksum: # 43288d, Taille: 240.5 octets)

```

" IF DUP TYPE 2 ≠ THEN
  "Chercher le premier:" { "" α } INPUT
END
OVER RCL SIZE
→ ch n
" CLLCD "En cours..." 1 DISP
0
DO 1 +
UNTIL
  DUP2 n MIN GET ch POS OVER n > OR
END
IF DUP n > THEN
  DROP "Pas d'occurrence" DOERR
ELSE
  LIRE
END
"
"

```

SELECTION D'ENREGISTREMENTS.

=====

'SELEC' effectue une recherche séquentielle, dans une base de données, de tous les enregistrements contenant une sous-chaine donnée.

Il y a 2 schémas fonctionnels possibles:

NOM désigne le nom de la base de données.

Premier cas:

2:	NOM	Ici <u>CHN</u> est la chaîne de caractères à rechercher.
1:	CHN	

Deuxième cas:

1:	NOM	Dans ce cas un message "Trouver les :" vous invite à donner l'objet à chercher (et qui est automatiquement transformé en chaîne). Le clavier est en mode Alpha: Vous entrez l'objet cherché (pas besoin de ") et vous validez par ENTER).
----	-----	---

Le message "En cours..." est affiché pendant la recherche.

Si la sous-chaine est trouvée dans un enregistrement, alors le programme 'SELEC' crée une nouvelle base de données, nommée '°', avec les enregistrements qui contenaient cette sous-chaine.

On passe alors en lecture de la base de données '°'.

A l'issue de cette lecture, la variable '°' n'est pas purgée.

Si la sous-chaine n'est pas trouvée, un message "Pas d'occurrence" est affiché.

```
'SELEC': ( Checksum: # 51383d, Taille: 265 octets )
" IF DUP TYPE 2 ≠ THEN
  "Trouver les : " { "" α } INPUT
END
OVER RCL SIZE → ch n
" CLLCD "En cours..." 1 DISP { }
  1 n FOR j
    OVER j GET
    IF DUP ch POS THEN +
      ELSE DROP
    END
  NEXT
  IF DUP SIZE THEN
    '°' STO '°' LIRE
  ELSE DROP "Pas d'occurrence..." DOERR
  END
" "
```


INDEX DES NOMS DE PROGRAMME.

Nom	Programme	Page
ABCUV	Résolution de $AU+BV=C$ (A,B,C,U,V polynômes).....	66
ABCXY	Résolution de $ax+by=c$ (a,b,c,x,y entiers ≥ 0 ou ≤ 0)...	18
ACS	Arc-cosinus d'un DL et DL de $ACOS(X)$	160
ADDID	Border un tableau à droite par la matrice identité...	106
ADDF	Addition de deux fractions rationnelles.....	71
ADDL	Addition de deux entiers longs.....	206
ADDP	Addition de deux polynômes.....	43
AIRE	Calculs d'aires planes.....	180
AJOUT	Ajouter un enregistrement à une base de données.....	272
AJUST	Ajustement par une loi normale $N(m,\sigma)$	233
ALU	Décomposition "A=LU" d'une matrice carrée.....	84
ANGL	Calculs d'écarts angulaires.....	171
ANIM	Successions d'images-écran.....	203
ANP	Arrangements.....	216
APMC	Approximations par la méthode des moindres carrés....	121
ASH	Arg-sinus hyperbolique d'un DL et DL de $ASINH(X)$	162
ASNS	Arc-sinus d'un DL et DL de $ASIN(X)$	159
ATG	Arc-tangente d'un DL et DL de $ATAN(X)$	161
ATH	Arg-tan. hyperbolique d'un DL et DL de $ATANH(X)$	163
AX	Exponentiation d'un D.L. et D.L. de $A^{\pm X}$	150
BINO	Liste des coefficients du binôme.....	217
BNP	Loi binômiale $B(n,p)$	218
BRST	Zéros d'un polynôme par la méthode de Bairstow.....	53
CALC	Evaluation d'expressions rationnelles.....	20
CALCC	Calculs sur les colonnes d'une matrice.....	92
CALCL	Calculs sur les lignes d'une matrice.....	90
CB	Changement de matrice par changement de base.....	81
CERC	Equation d'un cercle ou d'une sphère.....	168
CH	Cosinus hyperbolique d'un D.L. et D.L. de $COSH(X)$	157
CH→L	Transformation d'une chaîne de car. en entier long...	212
CNP	Combinaisons sans répétitions.....	216
COL.N	Extraction d'une des colonnes de $\Sigma DAT1$	256
COMP	Composition de deux polynômes.....	48
COR	Coeff. de corrélation de la stat. double (X,Y).....	265
CPDL	Composition de deux développements limités.....	141
CRBRE	Centre et rayon de courbure d'une courbe plane.....	182
CRTAB	Créer un tableau avec une formule donnée.....	94
CS	Cosinus d'un D.L. et D.L. de $COS(X)$	154
→CUM	Création d'un tableau de cumulés.....	254
CUM→	Retour d'un tableau de cumulés au tableau initial....	255
CURV	Calculs d'intégrales curvilignes.....	178
CV	Covariance de la statistique double (X,Y).....	265
CYCLO	Calcul des polynômes cyclotomiques.....	22
C.COL	Création d'une colonne fonction de celles de ΣDAT ...	257
DECPF	Décomposer 1 fraction rationnelle en élt.s simples....	75
DEFL	Valeurs et vecteurs propres (par déflation).....	98
DEG2	Zéros complexes ou réels d'un polynôme de degré 2....	50
DEG3	Zéros complexes ou réels d'un polynôme de degré 3....	51
DEG4	Zéros complexes ou réels d'un polynôme de degré 4....	52
DELTA	Laplacien d'une fonction de plusieurs variables.....	191
DERDL	Dérivation d'un développement limité.....	147
DERIV	Dérivation d'un polynôme.....	57
DERVF	Dérivée d'une fraction rationnelle.....	74
DEVLP	Développements d'expressions polynômiales.....	62

INDEX DES NOMS DE PROGRAMME (suite).

Nom	Programme	Page
DEVPP	Développer un produit de puissances de polynômes....	63
DIFF	Différentielle d'1 fonction de plusieurs variables...	193
DIM	Ordre et valuation d'un développement limité.....	137
DIST	Calculs de distances.....	169
DIV	Division euclidienne de deux polynômes.....	46
DIVL	Division de deux entiers longs.....	208
DIVPC	Diviser 2 polynômes suivant les puissances croiss....	47
DIVPR	Division de tableaux avec précision accentuée.....	105
DIVRG	Divergence d'un champ de vecteurs.....	185
DLN	Puissances d'un développement limité.....	144
DRTE	Equation d'une droite du plan.....	166
DX→Y	Droite des moindres carrés de X en Y.....	266
DY→X	Droite des moindres carrés de Y en X.....	266
ECAR	Ecart arithmétique moyen d'une stat. discrète.....	244
ECHC	Echange de deux colonnes dans une matrice.....	89
ECHL	Echange de deux lignes dans une matrice.....	89
EFFX	Effectifs en X de la statistique double (X,Y).....	261
EFFY	Effectifs en Y de la statistique double (X,Y).....	261
ELMD	Eliminer coefficients nuls à droite d'un vecteur.....	68
ELMG	Eliminer coefficients nuls à gauche d'un vecteur.....	68
ELMG	Eliminer coeff. nuls à gauche (entiers longs).....	214
ENV	Tracé de l'enveloppe d'une famille de droites.....	200
EQRL	Recherche de relations ou d'équations linéaires.....	114
EULER	Indicateur d'Euler d'un nombre entier.....	24
EX	Exponentielle d'un D.L. et D.L. de exp(± X).....	149
EXPO	Puissances entières d'un polynôme.....	45
EXPOF	Puissances d'une fraction rationnelle.....	73
EXPOL	Puissances entières d'un entier long.....	207
EXTRE	Extremums d'une courbe y=f(x).....	119
FACTL	Factorielle d'un entier (résultat: entier long).....	210
FAMT	Tracé d'une famille de courbes dépendant de T.....	202
FBNP	Fonction de répartition de la loi binômiale B(n,p)...	219
FCTR	Factorisation en produit de facteurs premiers.....	19
FEXP	Fonction de répartition de la loi exponentielle.....	230
FGEO	Fonction de répartition de la loi géométrique G(p)...	225
FHNAP	Fonction de répartition de la loi H(n,a,p).....	221
FIND	Chercher un enregistrement d'une base de données....	276
FJRP	Fonction de répartition de la loi J(r,p).....	229
FOUR	Sommes partielles de séries de Fourier.....	131
FPOIS	Fonction de répartition de la loi de Poisson P(L)...	223
FPRP	Fonction de répartition de la loi de Pascal P(r,p)...	227
FRCN	Calcul de fractions continues.....	29
FRMT	Gestion des débordements sur les entiers longs.....	214
F→Q	Ecriture rationnelle des coeff. d'1 fract. ration....	72
GANP	Combinaisons avec répétitions.....	217
GEO	Loi géométrique G(p).....	224
GETC	Extraire une colonne d'une matrice.....	88
GETL	Extraire une ligne d'une matrice.....	87
GINI	Indice de Gini d'une statistique discrète.....	249
GRADI	Gradient d'une fonction de plusieurs variables.....	189
G→D	Transformer une stat. groupée en stat. discrète.....	236
HIST	Histogramme d'une statistique groupée.....	252
HNAP	Loi Hypergéométrique H(n,a,p).....	220
INFL	Points d'inflexion d'une courbe y=f(x).....	120

INDEX DES NOMS DE PROGRAMME (suite).

Nom	Programme	Page
INTDL	Intégration d'un développement limité.....	148
INTP	Intégration d'un polynôme sur un segment.....	58
INTZ	Intégration dans le plan complexe.....	39
INVDL	Inverse d'un D.L. et D.L. de $1/(1 \pm X)$	142
INVN	Inversion d'une matrice à coefficients entiers.....	83
INVPR	Inversion de matrice avec précision accentuée.....	105
ITER	Calculs itératifs.....	34
I→J	Calculs avec le nombre complexe j.....	33
JRP	Loi binômiale négative J(r,p).....	228
KA↑X	Ajustement par une fonction $Y=k*a^X$ (stat double)....	267
KX↑A	Ajustement par une fonction $Y=k*X^a$ (stat double)....	267
LAGR	Polynôme intrepoleteur de Lagrange.....	123
LIRE	Lire une base de données.....	273
LG	Log népérien d'un D.L. et D.L. de $LN(1 \pm X)$	151
LPRM	Liste des diviseurs premiers d'un entier.....	21
LRTZ	Courbe de Lorentz d'une statistique discrète.....	251
L→CH	Transformation d'un entier long en chaîne de car....	211
L→R	Transformation d'un entier long en réel.....	213
MCK	Moment centré d'ordre k d'une stat. discrète.....	238
MCRK	Moment centré réduit d'ordre k d'1 stat. discrète....	239
MDLE	Médiale d'une statistique groupée.....	247
MGE0	Moyenne géométrique d'une stat. discrète.....	245
MHAR	Moyenne harmonique d'une stat. discrète.....	246
MK	Moment d'ordre k d'une statistique discrète.....	237
MODT	Appliquer un même programme aux coeff d'un tableau...	269
MODE	Modifications des colonnes de ΣDAT.....	258
MOEB	Valeurs de la fonction de Moëbius.....	25
MTCL	Méthode de Monte-Carlo pour visualiser $F(X,Y)=0$	201
MULT	Vecteur des produits terme à terme de 2 vecteurs....	268
MX	Moyenne en X de la statistique double (X,Y).....	262
MY	Moyenne en Y de la statistique double (X,Y).....	262
NCR→	Inverse de la fonction de répartition de N(0,1).....	232
→NCR	Fonction de répartition de la loi Normale N(0,1)....	231
NRM→	Inverse de la fonction de répartition de N(m,σ).....	232
→NRM	Fonction de répartition de la loi Normale N(m,σ)....	231
NXY	Création de la matrice des $N_{ij}X_iY_j$ (stat. double)....	264
PAR	Tracé d'une courbe paramétrée.....	196
PCAR	Polynôme caractéristique d'une matrice carrée.....	96
PFC	Polygone des fréquences cumulées (stat. groupée)....	253
PGCD	Plus grand commun diviseur de deux entiers.....	16
PGDL	Pgcd de deux entiers longs.....	209
PGDP	Pgcd de deux polynômes.....	59
PIVOT	Exécuter la méthode du pivot de Gauss.....	109
PLAN	Equation d'un plan.....	167
POIS	Loi de Poisson P(L).....	222
POL	Tracé d'une courbe en polaires.....	197
POLM	Polynôme minimal d'une matrice carrée.....	107
POLP	Tracé d'une courbe en polaires paramétrées.....	198
PPCM	Plus petit commun multiple de deux entiers.....	16
PPCS	Primitive symbolique de $P(x)\cos(ax)+Q(X)\sin(ax)$	64
PPEX	Primitive symbolique de $P(x)\exp(ax)$	65
PPML	Ppcm de deux entiers longs.....	210
PPMP	Ppcm de deux polynômes.....	59
PRIM	Primitivation d'un polynôme.....	58

INDEX DES NOMS DE PROGRAMME (suite).

Nom	Programme	Page
PROD	Calcul de produits partiels.....	28
PRODF	Produit de deux fractions rationnelles.....	71
PRODL	Produit de deux entiers longs.....	206
PRODP	Produit de deux polynômes.....	44
PRP	Loi de Pascal $P(r,p)$	226
PUISS	Puissances positives ou négatives d'une matrice.....	82
PUTC	Placer une colonne dans une matrice.....	86
PUTL	Placer une ligne dans une matrice.....	85
QDL	Quotient de deux développements limités.....	143
QTLE	Quantiles d'une statistique groupée.....	242
RCNC	Racines n-èmes d'un nombre complexe.....	36
RANG	Calcul du rang d'une matrice.....	100
RECTG	Longueur d'un arc de courbe gauche.....	176
RECTP	Longueur d'un arc de courbe plane.....	174
RENV	Inversion de l'ordre des composantes d'un vecteur....	56
RK4	Equa. diff. $Y'=F(X,Y)$ par Runge-Kutta d'ordre 4.....	129
ROTA	Rotationnel d'un champ de vecteurs.....	187
R→L	Transformation d'un réel en entier long.....	213
SELEC	Sélection d'enregistrements d'une base de données....	277
SH	Sinus hyperbolique d'un D.L. et D.L. de $\sinh(X)$	156
SIMP	Simplification de fraction.....	17
SMPF	Simplification d'une fraction rationnelle.....	70
SN	Sinus d'un D.L. et D.L. de $\sin(X)$	153
SXY	Résolution de systèmes $F(X,Y)=0$, $G(X,Y)=0$	125
SXYZ	Systèmes $F(X,Y,Z)=0$, $G(X,Y,Z)=0$, $H(X,Y,Z)=0$	127
SYST	Résolution symbolique d'un système linéaire.....	102
TCHBE	Calcul des polynômes de Tchebichev.....	61
TG	Tangente d'un D.L. et D.L. de $\tan(X)$	155
TGTE	Equation de la tangente en un point d'une courbe....	118
TH	Tangente hyperbolique d'un D.L. et D.L. de $\tanh(X)$...	158
TOTAL	Somme des coefficients d'un tableau.....	268
TR	Trace d'une matrice carrée.....	81
TRACE	Tracé d'une famille de points et d'une courbe.....	124
TRACE	Module de tracé utilisé par 'PAR', 'POL', 'POLP'.....	199
TRI	Trier une base de données.....	275
TRIG	Calculs trigonométriques (ex: linéarisations).....	37
TRNS	Translaté d'un polynôme.....	49
T→Q	Approximations rationnelles dans un tableau.....	31
VALF	Valeur d'une fraction rationnelle en un point.....	73
VALP	Valeur d'un polynôme en un point.....	56
VECTP	Equations de sous-espaces propres d'une matrice.....	104
VISU	Visualisation algébrique d'un développement limité...	138
VP234	Valeurs propres d'une matrice d'ordre 2,3,4.....	97
VX	Variance en X de la statistique double (X,Y).....	263
VY	Variance en Y de la statistique double (X,Y).....	263
V→F	Ecriture algébrique d'une fraction rationnelle.....	72
V→P	Passage vecteur→expression pour un polynôme.....	60
XA	Puissances réelles d'un DL et DL de $(1 \pm X)^A$	152
X→AX	Transformation $X \rightarrow A \cdot X$ dans un développement limité...	146
X→XN	Transformation $X \rightarrow X^N$ dans un développement limité...	145
YULE	Coefficients de Yule, Kelley, et Pearson.....	240
πDL	Produit de deux développements limités.....	140
EDL	Somme de deux développements limités.....	139

NOTES

NOTES

NOTES

NOTES

NOTES

Achevé d'imprimer par

PARA 
GRAPHIC

31240 L'UNION (Toulouse)

Tél. (16) 61.74.27.67

Dépôt légal : Novembre 1990

TABLE DES MATIERES DU TOME 2

PREAMBULE

INTRODUCTION

ARITHMETIQUE

NOMBRES REELS ET COMPLEXES

POLYNOMES

FRACTIONS RATIONNELLES

CALCUL MATRICIEL

ANALYSE

DEVELOPPEMENTS LIMITES

GEOMETRIE

GEOMETRIE DIFFERENTIELLE

GRAPHISME

ENTIERS LONGS

PROBABILITES

STATISTIQUES SIMPLES

STATISTIQUES DOUBLES

BASES DE DONNEES

INDEX