HP-28S

PROGRAMMATION ET APPLICATIONS

Jean-Michel FERRARD

D2I DIFFUSION

HP-28S

PROGRAMMATION ET APPLICATIONS

Jean-Michel FERRARD

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

D2I DIFFUSION

NOTICE

Les informations contenues dans cet ouvrage peuvent être changées sans préavis.

D2I DIFFUSION et HEWLETT-PACKARD ne donnent aucune garantie en ce qui conserne l'usage qui pourrait être fait des formules et exemples qui sont décrits dans le présent livret.

Le livret et toutes les informations qu'il contient ne peuvent être copiés en tout ou partie sans notre consentement.

TABLE DES MATIERES

INTRODUCTION

L'ambition de ce livre est d'être la référence en matière de programmation de la calculatrice HP28S de Hewlett-Packard. Il s'adresse donc tout particulièrement à ceux et celles qui possèdent déjà ce bel objet ou qui pensent (et ils ont raison) l'acquérir bientôt.

Bien sûr un certain niveau mathématique est recommandé; celui d'un baccalauréat scientifique me semble un minimum. Mais le contenu de ce livre n'est pas exclusivement matheux...

L'étude des statistiques constitue par exemple une discipline quelque peu éloignée des mathématiques pures (et dans ce livre deux chapitres y sont consacrés); d'autre part, les programmes du répertoire 'DATA' permettent de constituer de véritables banques de données (dans lesquelles vous mettrez ce qui vous plaira ...).

Pour fixer les idées, disons que cet ouvrage, s'adresse aux étudiants et aux professeurs:

* des classes terminales C,D,E.
* 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.

Le but de ce livre n'est pas de vous apprendre à vous servir de la HP28S. Sur ce point les deux manuels fournis avec la machine sont amplement suffisants. J'ai plutôt voulu ici proposer une vaste bibliothèque de programmes où chacun pourrait trouver son bonheur. Ce livre n'est donc un apprentissage de la programmation sur HP28S que dans la mesure où il vous permettra de progresser par l'observation de programmes qui fonctionnent et qui, du moins je l'espère, sont plutôt correctement écrits.

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. Nombre de programmes de ce livre vous deviendront vite indispensables et je pense que leur utilité est indéniable dans le cadre de la préparation aux concours et examens de l'enseignement supérieur.

Le mérite en revient bien entendu à cette calculatrice extraordinaire qu'est la HP28S de Hewlett-Packard.

LA HP28S: BREF PORTRAIT:

La HP28S constitue en effet un progrès considérable dans le domaine des calculatrices. Non seulement cette machine dispose de fonctions scientifiques extrêmement performantes, mais dans sa conception, tout à été prévu pour en faciliter l'apprentissage et l'utilisation.
La HP28S est donc à la fois la calculatrice la plus puissante du marché, mais en même temps elle est d'une simplicité d'emploi étonnante.

Ce n'est pas une des moindres qualités de la HP28S que de proposer à l'utilisateur une mémoire libre de 32 K-octets (plus de 32000 caractères).
Encore faut-il pouvoir:
* occuper cette mémoire par des programmes.
* occuper cette mémoire de façon rationnelle.

Ce livre veut répondre à la première de ces deux exigences.

Hewlett-Packard a déjà répondu à la seconde en munissant la HP28S d'un "système d'exploitation" qui n'est pas sans rappeler ce que l'on trouve en micro-informatique.
Le concept de répertoire et de sous-répertoire permet en effet de répondre de manière élégante aux problèmes posés ici par la quantité de mémoire disponible. L'utilisateur est d'autre part "tenu par la main" par un système de nenus qui lui permettent d'accéder rapidement à n'importe quel objet situé en mémoire (programmes, variables, fonctions).
Déjà toutes les fonctions de la HP28S, qui sont accessibles en tapant leur intitulé au clavier, sont regroupées dans des menus spécifiques. L'écriture de programmes en est grandement facilitée.

Il revient alors à l'utilisateur de créer lui-même des répertoires dans lesquels il stockera les programmes et les variables dont il a besoin, chaque répertoire contenant des données ayant évidemment un rapport entre elles. C'est autour de cette idée que sont articulés les différents chapitres de ce livre.
Au niveau des fonctions scientifiques de la HP28S, il faut saluer plusieurs grandes nouveautés:

1) Le calcul symbolique:

La possibilité d'effectuer des calculs symboliques, c'est à dire où les expressions utilisées peuvent contenir des variables formelles, ouvre en effet des perspectives inabordées jusqu'ici sur de petits systèmes.
Il serait trop long de passer en revue tout ce que peut faire la HP28S en ce domaine. Signalons ce qui pour moi s'est avéré le plus utile à savoir la possibilité de calculer la dérivée d'une expression symbolique (éventuellement dérivée partielle) et d'obtenir encore le résultat sous forme symbolique. Les programmes du chapitre 'GEOMETRIE DIFFERENTIELLE' font un grand usage de cette possibilité.

2) La double logique (RPN et algébrique):

Hewlett-Packard s'est fait le spécialiste de la "notation polonaise inverse", dite encore logique "RPN", sur ses calculatrices. Cette logique diffère par exemple de la notation algébrique traditionnelle (la plus courante) par l'ordre dans lequel sont entrées les données lors d'un calcul numérique. Pour calculer par exemple '7*(5+3/11)' il faut :

En notation algébrique traditionnelle: entrer l'expression telle qu'elle apparaît sur le papier, puis valider (ENTREE, EXE, ...).

En notation polonaise inverse: on entre les opérandes avant les opérateurs, ce qui donne ici la séquence: 7 5 3 11 / + *.

On reproche à cette logique d'être éloignée de notre façon d'écrire les expressions algébriques. Elle a tout de même plusieurs avantages:

* Elle permet de visualiser le résultat de chaque étape du calcul.
* Elle permet de se passer du programme d'évaluation que nécessite l'utilisation de la notation algébrique.

Hewlett-Packard a résolu ce problème en nous permettant d'utiliser, au choix et à tout moment, l'une ou l'autre de ces deux logiques. Il y en a donc pour tous les goûts.

3) La pile:

Le concept de "pile" occupe une position centrale pour l'utilisateur de la HP28S. Cette pile est une zone de la mémoire où sont déposés les objets en cours d'utilisation ou qui attendent leur tour.

Il s'agit bien d'une pile dans la mesure où les objets sont disposés "les uns sous les autres" suivant l'ordre dans lequel ils ont été introduits. Un nouvel objet est ainsi placé au niveau 1 de la pile, les autres objets remontant alors tous d'un niveau.

L'écran de la HP28S fait apparaître les 3 ou quatre premiers niveaux de cette pile. Mais l'utilisateur peut à tout moment circuler dans cette pile et y visualiser les objets qu'il y a déposé. Enfin, et c'est une nouveauté par rapport aux modèles précédents de Hewlett-Packard, la pile peut être aussi longue que le souhaite l'utilisateur.

On peut pratiquement tout mettre sur la pile: des variables, des programmes, des nombres, des tableaux, des listes, etc....

Le principe est que tout est possible tant que l'utilisateur n'essaie pas de composer ensemble des objets qui ne peuvent l'être. Mais le nombre d'opérations que l'on peut réaliser est étonnant. L'exemple du calcul matriciel suffit à prouver les possibilités qu'offre ce concept de pile.

Considérons le problème suivant: on veut effectuer le produit de deux matrices A et B. Sur la plupart des calculatrices, il faut implanter soit-même un programme permettant d'effectuer ces produits. Sur certaines machines plus récentes, ce programme est déjà implanté, mais son utilisation est longue et fastidieuse. Sur la HP28S, il suffit de rentrer la première matrice sur la pile (la syntaxe est très simple), puis la seconde, puis d'appuyer sur la touche *.

On procède de même pour calculer l'inverse ou le déterminant d'une matrice carrée, ou pour résoudre un système d'équations linéaires. Le gain de temps est extraordinaire.

4) Le langage de programmation:

Le langage de programmation de la HP28S est extrêment puissant. Il s'apparante en bien des points à "Turbo-Pascal" que connaissent maintenant de nombreux étudiants. Parmi les points communs, on peut relever:

* absence de numéros de lignes.
* programmation récursive (possibilité pour un programme de s'appeler lui même): un certain nombre de programmes de ce livre utilisent la récursivité. Citons par exemple le programme 'FACTL' du répertoire 'LONG'. Les programmes 'CS' et 'SN' du répertoire 'DL' constituent par ailleurs un bel exemple de récursivité croisée (chacun d'eux appelle l'autre).
* possibilité de créer des variables locales (c'est à dire dont la durée de vie se limite à l'éxécution du programme où elles sont crées). Un sous-programme peut lui-même être placé dans une variable locale.
* toute la panoplie des instructions de test et de contrôle est présente:

'IF..THEN..END' 'IF..THEN..ELSE..END'
Boucles 'FOR (ou START)... NEXT', 'FOR (ou START)...STEP'
'DO..UNTIL..END', 'WHILE..REPEAT..END' etc....

D'autre part, il est particulièrement agréable de pouvoir appeler un programme uniquement en donnant son nom. Voyez par exemple le programme 'CALC' du menu 'ARIT'. Le texte de 'CALC' contient un appel au programme 'SIMP' qui appelle lui même le programme 'PGCD'.

Si vous êtes dans le bon répertoire, et si la ligne des menus est à l'écran, un appui sur une seule touche permet d'appeler le programme de son choix. On fait difficilement plus convivial.

Citons enfin l'aide considérable à la programmation que constituent l'instruction 'HALT' et le menu 'CONTRL', qui permettent d'éxécuter un programme "pas à pas". On peut ainsi facilement déceler les erreurs qui se nichent souvent dans le premier jet d'un programme.

5) Le graphique:

La HP28S possède un écran graphique de 32 poins sur 137. De nombreuses instructions permettent de gérer cet écran. Même si les dimensions de celui-ci sont modestes, il s'avère suffisant pour les application usuelles.

6) Enfin:

Sont présentes les deux fonctions qu'une calcultarice se doit de posséder aujourd'hui:

* une fonction de résolution d'équation.
* une fonction d'intégration (avec possibilité de régler la précision du résultat obtenu).

La conclusion de ce bref portrait est que la HP28S est aujourd'hui sans rivale sur le marché des calculatrices. Elle est l'outil idéal de tous les étudiants de l'enseignement supérieur à dominante scientifique. Encore faut-il lui donner de quoi satisfaire la soif de programmes que réclame sa belle capacité mémoire. Ce livre est fait pour çà.

L'ORGANISATION DE CE LIVRE:

Cet ouvrage est divisé en quatorze chapitres. Chacun d'eux correspond à une famille de programmes qui devront être installés dans un répertoire particulier. Les quatorze chapitres sont les suivants:

1) **ARITHMETIQUE**: Pgcd, Ppcm, simplification de fraction, décomposition en produit de facteurs premiers, estimation d'expressions rationnelles sous forme de fraction simplifiée, etc...

2) **NOMBRES REELS ET COMPLEXES**: Séries, produits infinis, fractions continues. Calculs itératifs. Calculs trigonométriques. Approximations rationnelles d'un réel. 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) **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.

5) **TECHNIQUES D'ANALYSE**: Polynôme de Lagrange. Approximation au sens des moindres carrés. Systèmes non linéaires. Série de Fourier. Equa diff. , etc..

6) **DEVELOPPEMENTS LIMITES**: Opérations sur les développements limités. Développements limités des fonction usuelles et compositions usuelles de D.L. , etc...

7) **GEOMETRIE AFFINE ET EUCLIDIENNE**: Equations. Distances, Ecarts angulaires. Changements de coordonnées.

8) **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..

9) **GRAPHISME**: Tracé d'une courbe paramétrée, en polaire, d'une famille de courbes. Enveloppe d'une famille de droites etc...

10) **ENTIERS LONGS**: Opérations sur les entiers longs ; arithmétique.

11) **PROBABILITES**: Lois de probabilité et fonction de répartition des lois usuelles. Dénombrements.

12) **STATISTIQUES SIMPLES**: Les différentes moyennes et caractéristiques d'une statistique simple. Courbe et indice de Gini. Histogramme. Polygône des fréquences cumulées. Médiale, etc...

13) **STATISTIQUES DOUBLES**: moyennes, variances des statistiques marginales. Corrélation. droites des moindres carrés, ect....

14) **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) Développements limités: 'DL'
5) Calcul matriciel: 'MATR'
6) Techniques d'analyse: 'ANALY'
7) Géométrie: 'GEOM'
8) Géométrie différentielle: 'GDIF'
9) Graphisme: 'GRAPH'
10)Entiers longs: 'LONG'
11)Probabilités: 'PROBA'
12)Statistiques simples: 'STAT1'
13)Statistiques doubles: 'STAT2'
14)Banques de données: 'DATA'

QUELS PROGRAMMES CHOISIR ?

Vous serez amenés à faire un choix dans les programmes que vous implanterez dans votre calculatrice. En effet ce livre contient environ pour 45 K-octets de programmes. La HP28S n'en possédant que (!) 32, tout ne tient pas dans une seule machine.
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', '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 'GRAPH', 'ANALY' ou 'GDIF'....

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

'ARIT': 2459	'R.C': 2606	'POLY': 5812
'DL' : 5844	'MATR': 3638	'ANALY': 2959
'GEOM': 1946	'GDIF': 2603	'GRAPH': 1710
'LONG': 2775	'PROBA': 2344	'STAT1': 5487
'STAT2': 1313	'DATA': 1932	

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? ...).

J'ai voulu que les listings soient particulièrement faciles à copier et à comprendre. Pour celà:

* 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.

Chaque programme est illustré par au moins un exemple.

Quelques conseils:

* Créer 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.
* Vérifier 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, vérifier 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).
* Faites attention au signe →: il apparaît dans de nombreux mots du langage de la HP28S qu'il vaut mieux écrire en utilisant les menus: passez par exemple dans le menu 'ARRY' pour obtenir les mots →ARRAY et ARRAY→ . 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 HP28S.
* J'ai utilisé des minuscules pour désigner les variables locales . Une confusion entre une minuscule et une majuscule entraînerait une erreur.

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.

NOTE: Le fait de désactiver la fonction LAST (récupération des arguments) peut conduire à des erreurs dans les programmes utilisant l'instruction IFERR.

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 ! (finies les mises au dénominateur commun fastidieuses).

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.
'**CALC**' : évaluation d'expressions rationnelles (exemple ci-dessus).

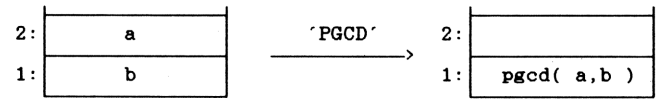
A coté, et si le coeur vous en dit, vous placerez les programmes suivants, qui sont plus spécialisés:

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

Répertoire 'ARIT' Programmes 'PGCD' et 'PPCM'

CALCUL DU P.G.C.D.

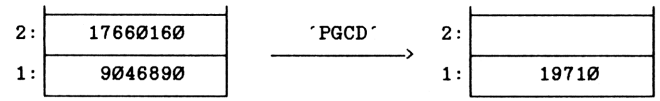
'PGCD' calcule le plus grand commun diviseur de deux entiers a et b suivant le schéma fonctionnel suivant:



'PGCD':

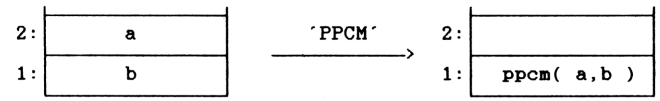
```
« → a b
  « WHILE a b DUP
    REPEAT
      DUP 'a' STO MOD 'b' STO
    END
    DROP ABS
  »
»
```

Exemple:



CALCUL DU P.P.C.M.

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

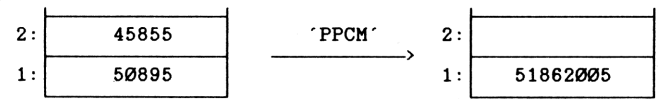


N.B: le programme 'PPCM' appelle le programme 'PGCD':

'PPCM':

```
« DUP2 → a b
  « PGCD b SWAP / a *
  »
»
```

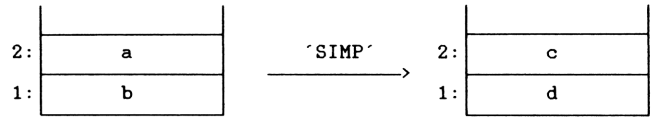
Exemple:



Répertoire ´**ARIT**´ Programme ´**SIMP**´

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):

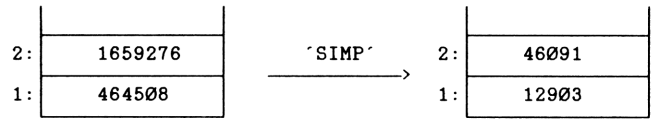


N.B: ´SIMP´ appelle le programme ´PGCD´.

´SIMP´:

```
«   DUP2  →   a   b
    «  PGCD  →   p
        «   a   p   /   b   p   /
        »
    »
»
```

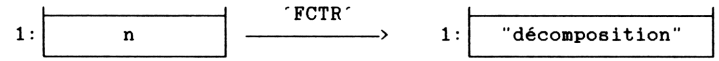
Exemple: (en moins d´une seconde)



Répertoire ´ARIT´ Programme ´FCTR´

FACTORISATION D´UN ENTIER.

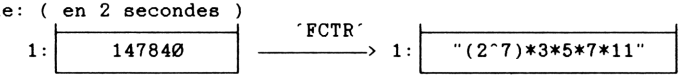
´FCTR´ effectue la décompostion d´un nombre entier n en un produit de facteurs premiers, selon le schéma fonctionnel:



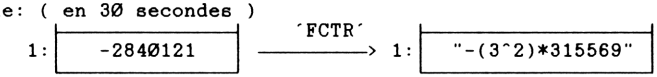
où "décomposition" est une chaîne de caractères représentant le résultat.

```
´FCTR´:
    «   IF  DUP  Ø  <  THEN   ABS   "-"   ELSE   ""   END   SWAP  2
        →   n   k
        «   ""
            WHILE   n   1   >
            REPEAT
               IF   n   k   MOD   THEN
                     IF   k   k   *   n   <   THEN
                        k   DUP   2   ≠   +   1   +   ELSE   n
                     END
                     ´k´   STO
                ELSE
                   Ø   →   x
                   «   n
                       WHILE   DUP   k   MOD   Ø   ==
                       REPEAT
                          k   /   x   1   +   ´x´   STO
                       END
                       ´n´   STO
                       IF   x   1   ==   THEN
                             "*"   +   k   →STR   +
                          ELSE
                             "*("   +   k   →STR   +   "^"
                               +   x   →STR   +   ")"   +
                       END
                   »
               END
            END
            DUP   SIZE   2   SWAP   SUB   +
        »
    »
```

Exemple: (en 2 secondes)



Exemple: (en 30 secondes)

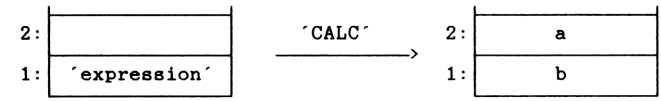


Répertoire 'ARIT' Programme 'CALC'

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'.

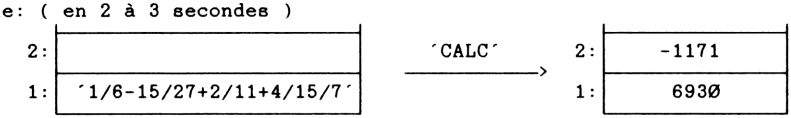
schéma fonctionnel:



'CALC':

```
«   DUP   EVAL   SWAP   →STR   1
    →  v   c   d
    «  WHILE   c   "/"   POS
       REPEAT
          c   DUP   "/"   POS   1   +   c
          SIZE   SUB   DUP   1   1   SUB   NUM
          IF   1Ø   ==   THEN
             DUP   SIZE   2   SWAP   SUB
          END
          'c'   STO   Ø
          WHILE   "Ø123456789"   c   1   1   SUB   POS   DUP
          REPEAT
             1   -   SWAP   1Ø   *   +   c   DUP   SIZE
             2   SWAP   SUB   'c'   STO
          END
          DROP   d   *   'd'   STO
       END
       v   d   *   .5   +   FLOOR   d   SIMP
    »
»
```

Exemple: (en 2 à 3 secondes)



Attention !

Pour que le programe 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 élévé à une puissance quelconque.

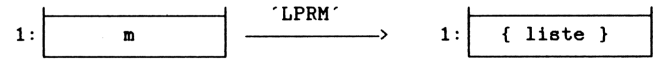
Ainsi '3/2^5 doit être remplacé par '3/32'.

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.

Répertoire 'ARIT' Programme 'LPRM'

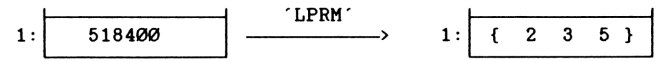
DIVISEURS PREMIERS D'UN ENTIER.

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




```
'LPRM':
   «   ABS   2  →  m   k
       «   { }
           WHILE   m   1   >
           REPEAT
              IF   m   k   MOD   THEN
                     IF   k   k   *   m   <   THEN
                            k   k   2   MOD   +   1   +
                        ELSE
                            m
                     END
                     'k'   STO
                  ELSE
                     k   +   m
                     WHILE   DUP   k   MOD   Ø   ==
                     REPEAT   k   /   END
                     'm'   STO
              END
           END
       »
   »
```

Exemple: (une seconde)



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

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

Répertoire 'ARIT' Programme 'CYCLO'

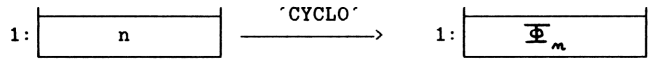
POLYNOMES CYCLOTOMIQUES.

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

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

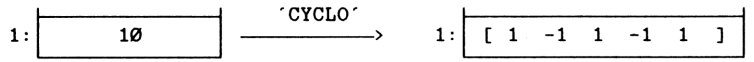
où k décrit les entiers compris entre 1 et n et premiers avec n.
Le degré du polynôme Φ_n est $\varphi(n)$, où φ est l'indicateur d'Euler.

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 X.

Exemple: (7 secondes)



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

Exemple: le polynôme Φ_{30} , de degré 8, est obtenu en 26 secondes.
le polynôme Φ_{105} , de degré 48, est obtenu en 1 mn 38 s.

Le programme 'CYCLO' utilise la formule:

$\Phi_m(x) = \prod_{d|m} (X^d - 1)^{\mu(\frac{m}{d})}$ 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'.

Répertoire 'ARIT' Programme 'CYCLO'

TEXTE DU PROGRAMME 'CYCLO'.

```
«
DUP   { }  Ø   Ø   [ 1 ]
→   n   m   dp   k   pr   pol
«   IF   n   1   ==   THEN   [ 1  -1 ]
    ELSE
       «   Ø   1   dp   SIZE   FOR   i
             IF   k   dp   i   GET   MOD   Ø   ==
               THEN   1   +   END
           NEXT
           2   MOD
       »
       «   →   w
           «   1   pol   SIZE   1   GET   n   k   /   -
               IF   w   -1   ==   THEN   SWAP   END
               FOR   i
                  pol   'i+n/k'   EVAL   'pol(i+n/k)+w*pol(i)'
                  EVAL   PUT   'pol'   STO
               w   STEP
           »
       »
       «   →   w
           «   pol   DUP   SIZE   1   GET   n   k   /
               w   *   -   1   →LIST   RDM   'pol'   STO
           »
       »
       →   impair   divmul   redim
       «   m   LPRM   DUP   'dp'   STO   LIST→   1   1   ROT
           START   *   NEXT   'pr'   STO
           pr   1   FOR   k
              IF   pr   k   MOD   Ø   ==   THEN
                 IF   impair   EVAL   NOT   THEN
                    -1   redim   EVAL   -1   divmul   EVAL
                 END
              END
           -1   STEP
           1   pr   FOR   k
              IF   pr   k   MOD   Ø   ==   THEN
                 IF   impair   EVAL   THEN
                    1   divmul   EVAL   1   redim   EVAL
                 END
              END
           NEXT
           pol
       »
    END
»
»
```

Répertoire 'ARIT' Programme 'EULER'

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.

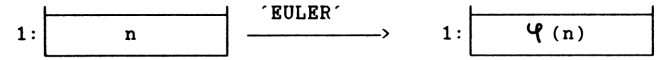
Si la décomposition de n en facteurs premiers s'écrit:

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

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

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:

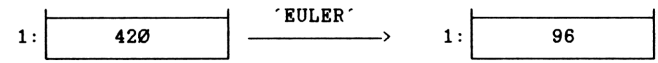


N.B: Le programme 'EULER' appelle le programme 'LPRM':

'EULER':

```
«   IF   DUP   1   >   THEN
    →      n
    «   n   LPRM   LIST→   n   1   ROT   START
            DUP   ROT   /   -
        NEXT
    »
»
```

Exemple: (1 seconde)



Répertoire ´**ARIT**´ Programme ´**MOEB**´

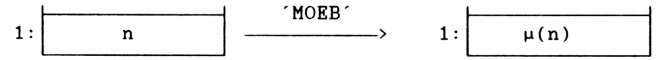
FONCTION DE MOEBIUS.

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

μ(n)= Ø 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 μ(n) suivant le schéma fonctionnel:



´MOEB´:

```
«   2  →  n  k
    «  1  WHILE  n  1  >
          REPEAT
             IF  n  k  MOD  THEN
                 IF  k  k  *  n  <  THEN
                     k  k  2  MOD  +  1  +
                 ELSE
                     n
                 END
                 ´k´  STO
             ELSE
                 NEG  n  k  /  ´n´  STO
                 IF  n  k  MOD  Ø  ==  THEN
                     Ø  *  ABORT  END
             END
          END
    »
»
```

On trouve par exemple:

μ(1)=1, μ(2)=-1, μ(4)=Ø, μ(6)=1

μ(3Ø)=-1 (en 1 seconde) μ(21Ø)=1 (en 1 seconde).

NOMBRES REELS ET NOMBRES COMPLEXES

Le répertoire consacré aux nombres réels ou complexes ne contient qu'un nombre limité de programmes. Non pas que le sujet soit trop mince, bien au contraire: la plupart des programmes que l'on utilise habituellement en mathématiques ne manipulent-ils pas des données numériques?

On trouvera ici 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:

- '**SERIE**': pour calculer les sommes partielles d'une série.
- '**PROD**' : pour calculer les produits partiels d'un produit infini.
- '**FRCON**': calcul des fractions continues et des réduites d'un nombre réel donné.
- '**R→Q**' : transformation d'un nombre réel en sa meilleure approximation rationnelle.
- '**ITER**' : permet d'effectuer des calculs récurrents (récurrence de pas 1, 2, ou 3).
- '**XP**' : calcul de la racine n-ème réelle d'un nombre réel.
- '**RACN**' : 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.

Répertoire 'R.C' Programme 'SERIE'

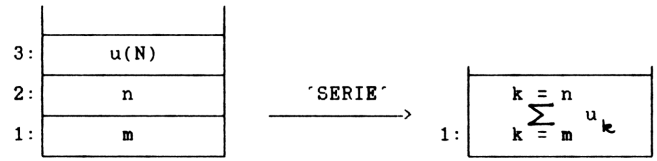
SOMMES PARTIELLES DE SERIES.

'SERIE' calcule la somme $\sum_{k=m}^{k=n} u_k$, où u_k est le terme général d'une série.

m et n sont les deux entiers représentant les valeurs extrémales de l'indice de sommation 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 de la sommation (pour que le calculateur utilise à plein sa précision, il vaut mieux lui faire additionner d'abord les plus petites quantités).

Le schéma fonctionnel est le suivant:



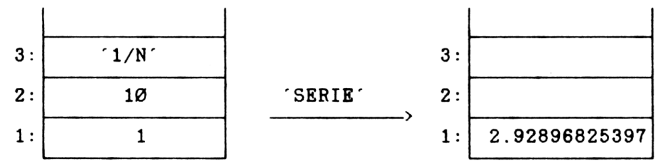
(où u(N) est une expression algébrique, ou un programme, permettant d'évaluer le terme u_N . Attention la majuscule "N" y est obligatoire) .

N.B: La variable N, si elle existe dans le répertoire, est effacée par le programme 'SERIE'.

'SERIE':

```
«   DUP2   »   2   *   1   -
    →  fonc   fin   deb   s
    «  Ø   deb   fin   FOR   i
           i   'N'   STO   fonc   EVAL   +
       s   STEP
      'N'  PURGE
    »
»
```

Exemple: (en 1 seconde)



(Ici le fait de permuter 1 et 1Ø aux niveaux 1 et 2 n'a aucune influence sur le résultat).

Répertoire 'R.C' Programme 'PROD'

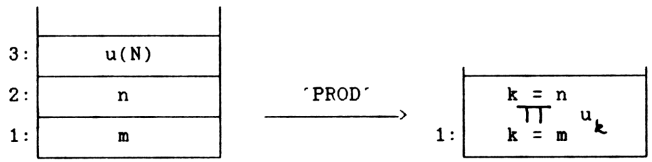
CALCULS DE PRODUITS PARTIELS.

'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 plus quantités les plus voisines de 1).

Le schéma fonctionnel est le suivant:



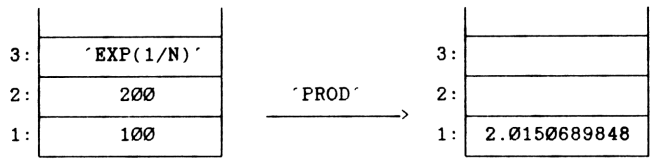
(où u(N) est une expression algébrique, ou un programme, permettant d'évaluer le terme u_N . Attention la majuscule "N" y est obligatoire) .

N.B: La variable N, si elle existe dans le répertoire, est effacée par le programme 'PROD'.

'PROD':

```
«   DUP2   >   2   *   1   -
    →  fonc   fin   deb   s
    «  1   deb   fin   FOR   i
          i  'N'   STO   fonc   EVAL   *
       s   STEP
      'N'  PURGE
    »
»
```

Exemple: (en 8 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 2.0150689848. Ce dernier résultat est sans doute plus précis).

Répertoire 'R.C' Programme 'FRCON'

CALCULS DE FRACTIONS CONTINUES.

Soit x un réel et a_1 sa partie entière. Si x n'est pas entier, on peut écrire:

$$x = a_1 + 1/x_1 \quad \text{avec } x_1 > 1.$$

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:

$$x = a_1 + \cfrac{1}{a_2 + 1/x_2}$$

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

$$x = a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \cfrac{1}{a_m + \cfrac{1}{x_m}}}}}$$

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

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

Les a_m sont appelés quotients partiels de cette fraction continue.

La fraction continue:

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \cfrac{1}{a_{m-1} + \cfrac{1}{a_m}}}}}$$

est notée $[a_1 / a_2 / \ldots / a_m]$ et est appelée n-iéme réduite de x

C'est un nombre rationnel r_m dont on notera $r_m = (p_m / q_m)$ l'expression sous forme de deux entiers premiers entre eux.

Les réduites r_m de x constituent 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_m et q_m convergent vers l'infini.
- x est toujours compris entre deux réduites consécutives.
- l'écart entre x et sa n_ième réduite r_m est tel que:

$$| x - r_m | \leqslant \frac{1}{q_{m-1}\, q_m}$$

Répertoire 'R.C' Programme 'FRCON' (suite)

FRACTIONS CONTINUES (suite).

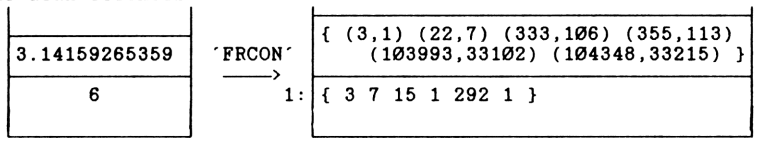
Le schéma fonctionnel de 'FRCON' est:

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_m / q_m.

'FRCON':

```
«   Ø   { }   (1,Ø)   (Ø,1)
    →   x   i   a   k   r   s
    «   1   i   START
            x   FLOOR   DUP   1   →LIST   k   SWAP   +   'k'   STO
            'a'   STO   x   a   -   INV   'x'   STO   r   a   *   s
            +   r   's'   STO   'r'   STO   r
        NEXT
        i   →LIST   k
    »
»
```

Exemple:
On veut connaître les réduites de π , jusqu'à l'ordre 6. On obtient, en moins de deux secondes:



Ce qui signifie que les réduites successives de π sont:

[3] = 3
[3 / 7]= 3 + 1/7 = 22/7 ≈ 3.14285714286.

$$[\ 3\ /\ 7\ /\ 15\ ] = 3 + \frac{1}{1 + 1/15} = 333\ /\ 106 \approx 3.14150943396.$$

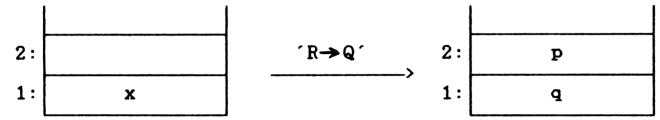
[3 / 7 / 15 / 1] = 355 / 113 ≈ 3.14159292035.
[3 / 7 / 15 / 1 / 292] = 103993 / 33102 ≈ 3.14159265301.
[3 / 7 / 15 / 1 / 292 / 1] = 104348 / 33215 ≈ 3.14159265392.

Répertoire 'R.C' Programme 'R→Q'

MEILLEURE APPROXIMATION RATIONNELLE D'UN REEL.

'R→Q' calcule le nombre rationnel p/q (p et q premiers entre eux) qui approche au mieux le réel x. Le sens de cette approximation est (sauf à modifier légèrement le programme) l'égalité en machine (autrement dit les douzes premiers chiffres significatifs de x doivent coïncider avec ceux du quotient p/q).

Le schéma fonctionnel est:

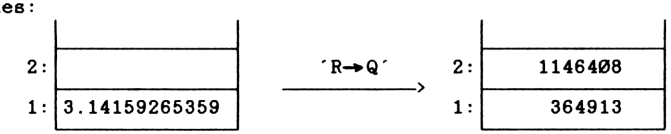


'R→Q':

```
«   DUP   (1,Ø)   (Ø,1)   Ø
    →  r   s   eps
    «  DO
          DUP   FLOOR   DUP   r   *   s   +
          r   's'   STO   'r'   STO   -
          IF   DUP   THEN
             INV   2   PICK   r   C→R   /   -   ABS
          ELSE   Ø   END
       UNTIL   eps   ≤   END
       DROP2   r   C→R
    »
»
```

Exemple:

Meilleure approximation rationnelle de π. On obtient en moins de deux secondes:



Effectivement, on a l'"égalité-machine": $\pi \approx 3.14159265359 \approx \frac{1146Ø8}{364913}$

Variante:

Le programme peut-être modifié, si on désire obtenir la meilleure approximation rationnelle de x (avec le plus petit dénominateur possible), en se donnant une marge d'erreur > Ø (donc en n'exigeant pas l'égalité machine). Il suffit pour celà de placer, à la première ligne du programme, la valeur de l'erreur à la place de Ø (valeur qui va dans la variable 'eps').

L'intérêt de cette variante est de lutter contre des erreurs d'arrondis qui auraient pu entacher la précision avec laquelle x est connu.

Répertoire 'R.C' Programme 'ITER'

CALCULS ITERATIFS.

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

Le problème est le suivant:

On se donne une relation de récurennce $U_n = F(U_{n-p}, U_{n-p+1}, \ldots, U_{n-1})$ permettant de calculer des "objets" U_n , la suite U étant initialisée par la donnée des p termes initiaux $U_0, U_1, \ldots, U_{p-1}$. 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 U_{N-p} U_{N-p+1} .... U_{N-1} $F(U_{N-p}, U_{N-p+1}, \ldots, U_{N-1})$ }.

Dès l'appel de 'ITER' celui-ci crée dans le répertoire les variables N, U0 U1 .... 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 crées.

```
'ITER':
    «   LISTE   SIZE
        «   "{ U"   SWAP   →STR   "}"   +   +   STR→   1   GET   »
        →   d   var
        «   'LISTE'   1   GET   'N'   STO   'LISTE'   { 2 }
            Ø   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
                     Ø   d   2   -   FOR   i
                         i   var   EVAL   PURGE
                     NEXT
                     ABORT
                ELSE
                     N   1   +   'N'   STO
                     1   d   2   -   FOR   i
                         i   var   EVAL   RCL   i   1   -   var   EVAL   STO
                     NEXT
                     d   3   -   var   EVAL   STO
                END
            UNTIL   Ø   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 $u_0=1$.

On place { 1, 1 ´UØ/2+2/UØ´} dans ´LISTE´ et on appelle ´ITER´.

Au bout de deux secondes (nécessaires aux créations des variables), le programme est suspendu et affiche la valeur de u_4 , c´est-à-dire 2.5; on fait CONT et on obtient la valeur de u_3 ,c´est à dire 2.Ø5; les valeurs suivantes sont: u_2 = 2.ØØØ6Ø975561 ,u_1 = 2.ØØØØØØØ9292, puis $u_0 \approx 2$.

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:

$$M_n = M_{n-1} - n*\left(M_{n-2}^{t}\right).$$

et par les termes initiaux

$$M_0 = \begin{bmatrix} 1 & Ø & Ø \\ Ø & 1 & Ø \\ Ø & Ø & 1 \end{bmatrix} \qquad M_1 = \begin{bmatrix} Ø & 1 & Ø \\ 1 & Ø & 1 \\ Ø & 1 & Ø \end{bmatrix}$$

On place { [[1 Ø Ø [Ø 1 Ø [Ø Ø 1]] [[Ø 1 Ø [1 Ø 1 [Ø 1 Ø]] « U1 UØ TRN N * - » } dans la variable ´LISTE´ et on appelle ´ITER´.

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

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

Exemple 3:

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

$$P_n(x) = nP_{n-1}(x) - x*P_{n-2}(x) + x^2P'_{n-3}(x).$$

et les termes initiaux:

$P_0(x)=Ø$, $P_1(x)=x$ et $P_2(x)=x^2$.

On place, dans la variable ´LISTE´:

{ 3 [Ø] [1 Ø] [1 Ø Ø] « N U2 U1 UØ POLY → n u2 u1 uØ « u2 n * u1 [-1 Ø] PRODP ADDP uØ DERIV [1 Ø Ø] PRODP ADDP R.C »}

(on remarque le passage dans le répertoire ´POLY´, où on crée des variables locales uØ,u1,u2,u3,n et le retour dans le répertoire R.C).

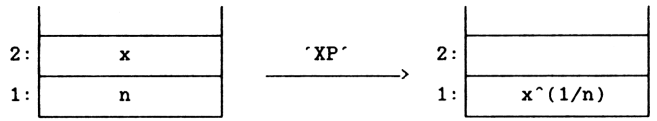
Répertoire 'R.C' Programme 'XP'

RACINE Nième D'UN REEL.

En mathématique, la valeur de (-8)^(1/3) (c'est-à-dire la racine cubique de -8) est égale à **-2** . Pourtant le résultat donné par la HP28 quand on lui demande d'évaluer (-8)^(1/3) est (1 , 1.732Ø5Ø8Ø757) c'est-à-dire -2j (nombre complexe dont la puissance troisième est effectivement égale à -8).

Le programme 'XP' apporte une solution à ce problème, en calculant la racine n-ième réelle du réel x (la solution est unique sauf si x est négatif et que l'entier n est pair, auquel cas il n'y a pas de solution).

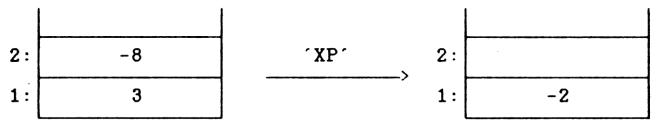
Le schéma fonctionnel est:



'XP':

```
«  →  n
   «  IF   DUP   Ø   ≥   THEN
            n   INV   ^
      ELSE
         IF   n   2   MOD   THEN
               ABS   n   INV   ^   NEG
         ELSE
            Ø   INV
         END
      END
   »
»
```

Exemple:



N.B: le programme 'XP' s'interromp sur une erreur quand n est nul, où quand n est pair et x négatif.

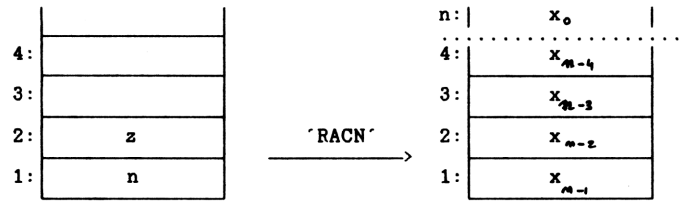
Application:

Si on veut visualiser la courbe représentative de $(X^2-1)^{(1/3)}$, il est préférable de tracer le programme « X SQ 1 - 3 XP » plutôt que l'expression '(X*X-1)^(1/3)' (car dans ce dernier cas la partie du graphe correspondant à $-1 \leq x \leq 1$ n'est pas tracée).

Répertoire 'R.C' Programme 'RACN'

RACINES Nièmes D'UN NOMBRE COMPLEXE.

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

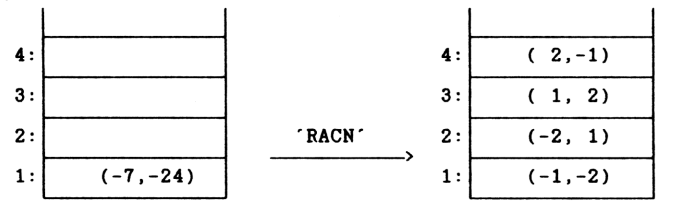


La formule utilisée est:
si $z = R \exp(i \theta)$, alors $x_k = \sqrt[n]{R} \exp(i(\theta/n + 2k\pi/n))$.

'RACN':

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

Exemple:



Répertoire 'R.C' Programme 'TRIG'

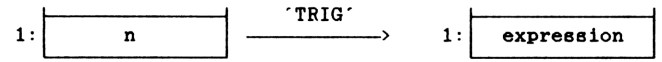
LINEARISATION ET OPERATION INVERSE DE LA LINEARISATION.

'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'.

Le schéma fonctionnel est:



où n est un entier et où "expression" est l'expression symbolique du résultat (c'est à dire, suivant les cas, de cos(t)^n, sin(t)^n, cos(nt), ou de sin(nt)).

Dès l'appel de 'TRIG', la chaîne de 23 caractères
"CS^ SN^ COS SIN" est affichée au niveau 1 de la pile.
CS^, SN^, COS, SIN se placent ainsi juste au dessus des touches INS, DEL, ◀ et ▶.
Un appui sur une de ces touches effectue alors l'opération correspondant respectivement 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, et appelé 'TRIG', on obtient, au niveau 1 de la pile:

'COS(T)^3=(COS(3*T)+3*COS(T))/4' si on a appuyé sur INS.
'SIN(T)^3=(-SIN(3*T)+3*SIN(T))/4' si on a appuyé sur DEL.
'COS(3*T)=4*COS(T)^3-3*COS(T)' si on a appuyé sur ◀.
'SIN(3*T)=-(4*SIN(T)^3)+3*SIN(T)' si on a appuyé sur ▶.

Exemple: avec n=15 , les résultats sont obtenus en 4 à 11 secondes.
on trouve par exemle:

'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)'

Répertoire 'R.C' Programme 'TRIG' (suite)

TEXTE DU PROGRAMME 'TRIG'.

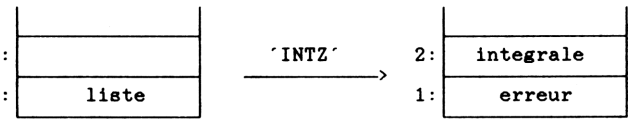
```
'TRIG':
    «    "CS^ SN^          COS SIN"   4   DISP
         DO    UNTIL   KEY   END   Ø
         →   n   t   k
         «   «   n   k   COMB   n   k   2   *   -   »
             «   IF   DUP   Ø   ==   THEN   DROP   2   /
                      ELSE   'T'   *   COS   *
                 END
             »
             «   2   n   1   -   ^   /   »
             «   IF   2   MOD   1   ==   THEN   NEG   END   »
             «   V→P   R.C   »
             →   p1   p2   p3   p4   p5
             «   IF   t   "INS"   ==   THEN
                    'COS(T)'   n   ^   Ø
                    Ø   n   2   /   FLOOR   FOR   k
                      p1   EVAL   p2   EVAL   +
                    NEXT   p3   EVAL
                 END
                 IF   t   "DEL"   ==   THEN
                    'SIN(T)'   n   ^   Ø
                    Ø   n   2   /   FLOOR
                    IF   n   2   MOD   Ø   ==   THEN
                       FOR   k
                          p1   EVAL   p2   EVAL   k
                          n   2   /   +   p4   EVAL   +
                       NEXT
                    ELSE
                       FOR   k
                          p1   EVAL   'T'   *   SIN   *   k   n
                          1   -   2   /   +   p4   EVAL   +
                       NEXT
                    END   p3   EVAL
                 END
                 IF   t   "LEFT"   ==   THEN
                    n   'T'   *   COS   n   POLY
                    1   TCHEB   'COS(T)'   p5   EVAL
                 END
                 IF   t   "RIGHT"   ==   THEN
                    n   'T'   *   SIN
                    IF   n   2   MOD   Ø   ==   THEN
                       IF   n   Ø   ==   THEN   Ø   ELSE
                          n   1   -   2   POLY   TCHEB   n   2   /   1
                          +   p4   EVAL   'SIN(T)'   p5   EVAL   'COS(T)'   *
                       END
                    ELSE
                       n   1   POLY   TCHEB   n   1   -   2   /   p4
                       EVAL   'SIN(T)'   p5   EVAL
                    END
                 END   =   CLMF
             »
         »
    »
```

Répertoire 'R.C' Programme 'INTZ'

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≤t≤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), "erreur" est un nombre complexe a+ib, a et b étant respectivement un majorant de l'erreur absolue commise sur la partie réelle et la partie imaginaire du résultat.

le format de "liste" est le suivant: { F X Y A B EPS }, avec:

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

X,Y: expressions symboliques, par rapport à la variable 'T', donnant la représentation paramétrique de l'arc Γ.

A,B: valeurs extrémales du paramètre T.

EPS: erreur relative permise sur le résultat.

'INTZ':

```
«  LIST→  'T'  PURGE  DROP
   →  a  b  eps
   «  i  *  +  DUP  'Z'  STO
      'T' ∂  *  DUP  RE  SWAP  IM
      1  2  START
         'T'  a  b  3  →LIST  eps  ∫  ROT
      NEXT
      R→C  SWAP  ROT  R→C  SWAP  {T Z}  PURGE
   »
»
```

Répertoire 'R.C' Programme 'INTZ'
(suite)

EXEMPLES D'UTILISATION DU PROGRAMME 'INTZ'.

Exemple 1:

Calculer $\int_\Gamma (1/Z)\,dZ$, où Γ est le cercle orienté $X(T)= COS(T)$, $Y(T)= SIN(T)$, $Ø \le T \le 2\pi$.

On met { '1/Z' 'COS(T)' 'SIN(T)' Ø '2*π' .1 } au niveau 1 de la pile et on appelle le programme 'INTZ'.
Au bout de 2Ø secondes, on obtient:

```
2 | ( -2.453Ø1266379E-13, 6.28318530718 ) |
1 | ( 4.54ØØØ675124E-14 , .628318530718 ) |
```

alors que le résultat exact est $2i\pi \approx (Ø, 6.28318530718)$.
On constate que le résultat obtenu sur la partie imaginaire est plus que précis, malgré le "Ø.1" d'incertitude relative acceptée. Le temps de calcul vient de la partie réelle du résultat, qui est nulle en théorie, et qui ici est calculée avec une incertitude absolue de 4.5 E -14.

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,Ø) et le point B(Ø,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 met au niveau 1 de la pile:
Dans le premier cas: { 'EXP(Z)' '1-T' 'T' Ø 1 .ØØØ1 },
Dans le deuxième cas: { 'EXP(Z)' 'COS(T)' 'SIN(T)' Ø 'π/2' .ØØØ1 },
et on appelle 'INTZ'.

Dans le premier cas, on obtient, en 16 secondes:

```
2 | (-2.17798298405, .841470884363) |
```

Dans le second cas, on obtient, en 33 secondes:

```
2 | (-2.17797993682, .84147123493) |
```

Alors que le résultat exact est:
exp(i)-exp(1) ≈ (-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 Ø Ø 1 Ø -2 Ø].

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 leur composantes selon les puissances croissantes de l'indéterminée.
- Dans le programme 'RENV' (qui effectue la bascule puissances croissantes/ puissances décroissantes).
- Dans le programme V→P (transformation d'un polynôme écrit sous forme de vecteur en une notation plus traditionnelle).

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

'**ADDP**', '**PRODP**', '**EXPO**', '**DIV**', '**DIVPC**', '**COMP**','**TRNSL**',qui permettent d'additionner, de multiplier, d'élever à une certaine puissance, de diviser (suivant les puissances croissantes ou décroissantes), de composer, ou de translater les polynômes.

'**DEG2**', '**DEG3**', '**DEG4**', '**BRST**', qui donnent les racines réelles ou complexes des polynômes de degré 2,3,4, ou plus (La méthode itérative de Bairstow est utilisée si le degré est supérieur à 4)

'**VALP**', '**DERIV**', '**PRIM**', '**INTP**', permettent d'évaluer un polynôme en un point, de le dériver, de le primitiver et de l'intégrer sur un segment.

'**PGCDP**', '**PPMP**' calculent le Pgcd et le Ppcm de deux polynômes.

'**V→P**' transforme un polynôme (écrit sous forme de vecteur) en une notation algébrique traditionnelle.

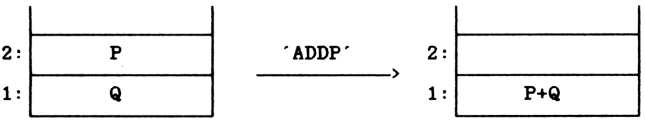
'**TCHEB**' calcule les polynômes de Tchébitchev de première ou de seconde espèce.

'**ELMG**', '**ELMD**' sont 2 routines utilisées lors des opérations sur les polynômes et dont l'objet est de neutraliser certaines erreurs d'arrondi.

Répertoire 'POLY' Programme 'ADDP'

ADDITION DE DEUX POLYNOMES.

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



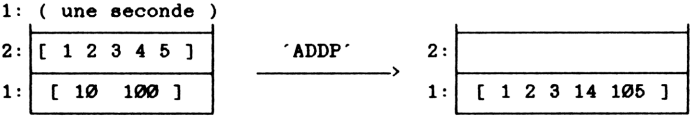
N.B: 'ADDP' est utile si les longueurs des vecteurs représentant P et Q sont différentes, ou si on ne connaît pas à priori ces longueurs (sinon il vaut mieux utiliser +).

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':

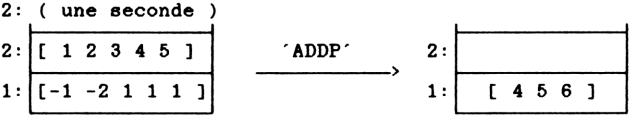
```
«  DUP2  SIZE  1  GET  SWAP  SIZE  1  GET  SWAP  -
   IF  DUP  Ø  ==  THEN
       DROP  +  ELMG
   ELSE
       IF  DUP  Ø  <  THEN  ABS  ROT  SWAP  END
       →  d
       «  ARRY→  1  GET  d  +
          →  n
          «  d  1  →LIST  Ø  CON  ARRY→  DROP
             1  d  START  n  ROLLD  NEXT
             n  1  →LIST  →ARRY  +
          »
       »
   END
»
```

Exemple 1: (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: (une seconde)

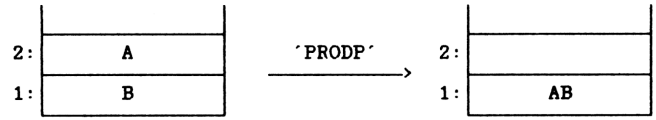


Ici on aurait pu tout simplement utiliser l'opération +, mais le résultat aurait été [Ø Ø 4 5 6].

Répertoire 'POLY' Programme 'PRODP'

PRODUIT DE DEUX POLYNOMES.

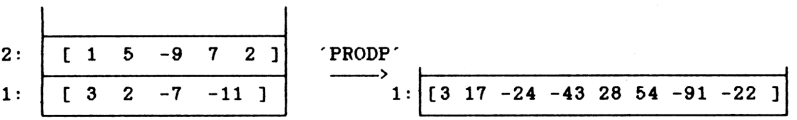
'PRODP' effectue le produit de deux polynômes A et B suivant le schéma:



'PRODP'

```
«  DUP2  SIZE  1  GET  SWAP  SIZE  1  GET
   IF  >  THEN  SWAP  END
   →  b  a
   «  a  SIZE  1  GET  DUP  b  SIZE  1  GET  +  DUP  1  →LIST
      →  la  lp  d
      «  a  { 1 }
         IF  la  1  ==  THEN
               GET  b  *
            ELSE
               1  la  FOR  i
                  GETI  b  *  d  RDM  ARRY→  DROP
                  1  i  START  lp  ROLLD  NEXT
                  lp  1  →LIST  →ARRY  3  ROLLD
               NEXT
               DROP2
               2  la  START  +  NEXT
               ARRY→  DROP  lp  ROLL  DROP
               lp  1  -  →ARRY
         END
      »
   »
»
```

Exemple: (en 3 secondes).



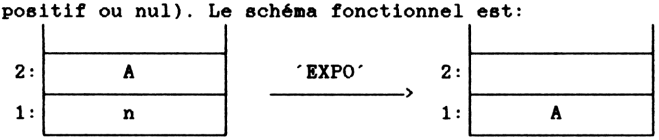
car si P = X^4 +5X^3 -9X^2 +7X +2 et Q = 3X^3 +2X^2 -7X -11, alors

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

Répertoire 'POLY' Programme 'EXPO'

PUISSANCES D'UN POLYNOME.

'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':

```
«  →  a  n
   «  [ 1 ]
      WHILE  n  Ø  >
      REPEAT
         IF  n  2  MOD  Ø  ≠  THEN  a  PRODP  END
         n  2  /  FLOOR  'n'  STO
         IF  n  Ø  >  THEN  a  DUP  PRODP  'a'  STO  END
      END
   »
»
```

Exemple:

On veut calculer (2X^2 -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 8 secondes. On obtient le vecteur:

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

ce qui signifie que:

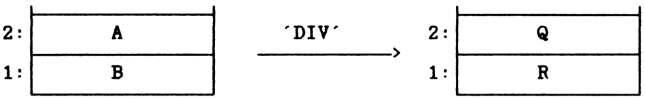
(2X^2 -3X +7)^5 = 32X^10 - 240X^9 + 1280X^8 ..... - 36015X + 16807.

Répertoire 'POLY' Programme 'DIV'

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)

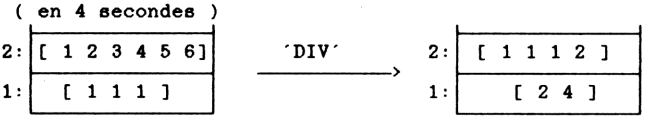
Le schéma fonctionnel est le suivant:



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

```
'DIV':
    «   Ø   Ø   Ø
        «   DUP   DUP   RE   -   ABS   IF   NOT   THEN   RE   END   »
        →   a   b   d   reel
        «   ELMG   'b'   STO   ELMG   DUP   'a'   STO   SIZE   1   GET
            b   SIZE   1   GET   DUP2   -   1   +
            →   m   n   k
            «   IF   m   n   ≥   n   1   >   AND   THEN
                    b   DUP   {1}   GET   DUP   'd'   STO   /   'b'   STO
                    k   1   →LIST   (Ø,Ø)   CON   {1}
                    1   k   START
                       a   DUP   {1}   GET   DUP   3   ROLLD   b   *
                       a   SIZE   RDM   -   ARRY→   1   GET   1   -
                       1   →LIST   →ARRY   SWAP   DROP   'a'   STO   PUTI
                    NEXT
                    DROP   d   /   reel   EVAL   a   reel   EVAL   ELMG
                ELSE
                    IF   n   1   ==   THEN
                        a   'b'   {1}   GET   /   [Ø]
                    ELSE
                        [Ø]   a
                    END
            END
        »
    »
»
```

Exemple: (en 4 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

Répertoire 'POLY' Programme 'DIVPC'

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:

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

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

Le schéma fonctionnel est:

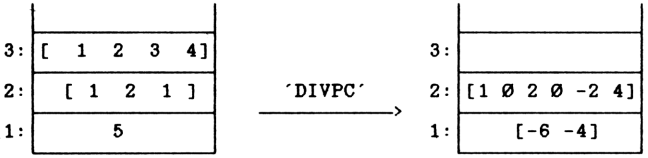
ATTENTION:

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 programe 'ELMD'.

```
'DIVPC':
    «   1   +   Ø   Ø
        «   DUP   DUP   RE   -   ABS   IF   NOT   THEN   RE   END   »
        →   a   b   n   d   m   reel
        «   a   SIZE   1   GET   b   SIZE   1   GET   MAX   1   →LIST
            DUP   DUP   'm'   STO   a   SWAP   RDM   'a'   STO   b   SWAP
            RDM   DUP   {1}   GET   DUP   'd'   STO   /   'b'   STO
            n   1   →LIST   (Ø,Ø)   CON   {1}
            1   n   START
               a   DUP   {1}   GET   DUP   3   ROLLD   b   *   -   ARRY→
               1   GET   1   -   1   →LIST   →ARRY   SWAP   DROP
               m   RDM   'a'   STO   PUTI
            NEXT
            DROP   d   /   ELMD   reel   EVAL   a   ELMD   reel   EVAL
        »
    »
```

Exemple: (en 5 secondes)



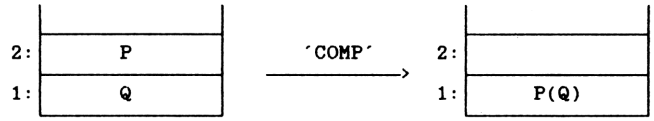
Ce qui signifie que:

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

Répertoire 'POLY' Programme 'COMP'

COMPOSITION DE DEUX POLYNOMES.

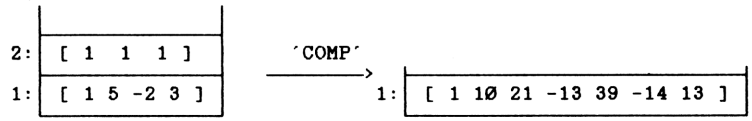
'COMP' effectue le calcul du composé P(Q) de deux polynômes P et Q selon le schéma fonctionnel:



N.B: 'COMP' appelle les programmes 'ELMG', 'PRODP' et 'ADDP'

```
'COMP':
    «   ELMG   SWAP   ELMG   DUP   SIZE   1   GET   [ 1 ]
        →  q   p   n   r
        «  p   n   1   →LIST   GET   { 1 }   →ARRY
           IF   n   1   >   THEN
              n   1   -   1   FOR   i
                 r   q   PRODP   DUP   'r'   STO
                 p   i   1   →LIST   GET   *   ADDP
              -1   STEP
           END
        »
    »
```

Exemple: (en 6 secondes)



Ce qui signifie qu'en posant:

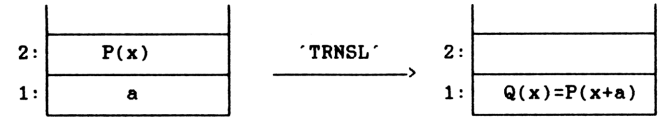
P(X) = X^2 +X +1 et Q(X) = X^3 +5X^2 -2X +3, alors

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

Répertoire 'POLY' Programme 'TRNSL'

TRANSLATE D'UN POLYNOME.

'TRNSL' transforme un polynôme x -> P(x) en le polynôme x -> Q(x)=P(x+a), suivant le schéma fonctionnel:

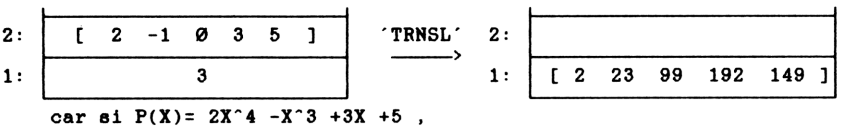


N.B: le programme 'TRNSL' appelle les programmes 'VALP', 'DERIV', 'RENV'.

'TRNSL':

```
«   SWAP   DUP   SIZE   1   GET
    →   a   p   n
    «   n   1   →LIST   Ø   CON   { 1 }
        Ø   n   1   -   FOR   i
            p   a   VALP   i   FACT   /   SWAP   DERIV   'p'   STO   PUTI
        NEXT
        DROP   RENV
    »
»
```

Exemple: (en 6 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 'TRNSL' donne également la décomposition du polynôme x -> P(x) suivant les puissances de (x-a).

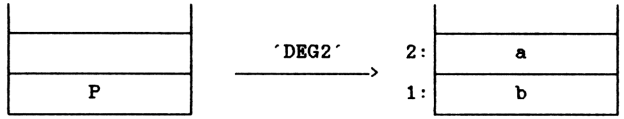
Ainsi l'exemple précédent peut-il être interprété en disant que:
si P(X)= 2X^4 -X^3 +3X +5 ,
alors P(X)= 2(X-3)^4 +23(X-3)^3 +99(X-3)^2 +192(X-3) +149

On peut encore dire que le programme 'TRNSL' 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.

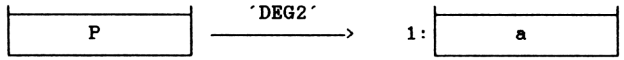
Répertoire 'POLY' Programme 'DEG2'

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:

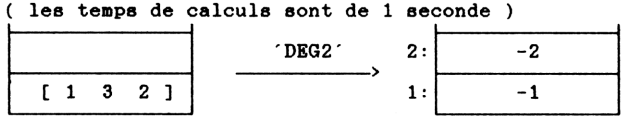


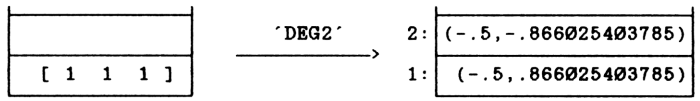
N.B: 'DEG2' appelle le programme 'ELMG'.

'DEG2':

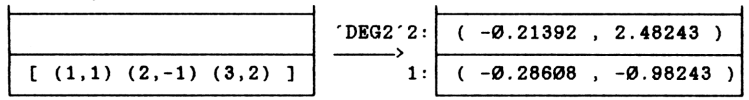
```
«   ELMG   DUP   SIZE
    →  p   s
    «  IF   s   { 3 }   ==   THEN
           'p(2)^2-4*p(1)*p(3)'   EVAL   √
           →  rd
           «  '-(p(2)+rd)/2/p(1)'   EVAL
              '(-p(2)+rd)/2/p(1)'   EVAL
           »
       ELSE
           IF   s   { 2 }   ==   THEN
              p   ARRY→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

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





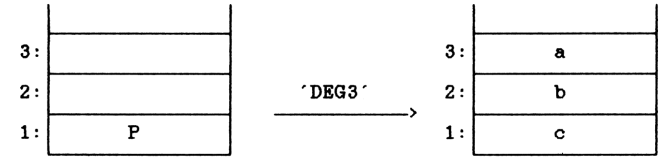
En mode 5 FIX,



Répertoire 'POLY' Programme 'DEG3'

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:

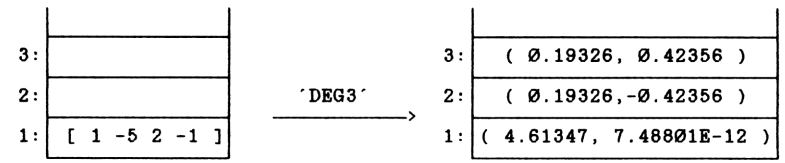


N.B: Celles des racines a,b ou c qui sont réelles sont cependant obtenues sous forme complexe (avec une partie imaginaire nulle ou presque compte tenu des erreurs d'arrondis).

'DEG3':

```
«   DUP  1  GET  /  ARRY→  DROP  RCLF  35  CF
    →  a  b  c  x
    «  DROP  'b-a^2/3'  EVAL  3  /
       '2*a^3/27-a*b/3+c'  EVAL  2  /
       →  p  q
       «  IF  p  ABS  Ø  ≠  THEN
             q  p  √  p  *  /  NEG  ASINH
             →  z
             «  Ø  2  FOR  k
                   '2*√p*SINH((z+2*i*k*π)/3)-a/3'  EVAL
                NEXT
             »
          ELSE
             Ø  2  FOR  k
                '(-2*q)^(1/3)*EXP(2*i*k*π/3)-a/3'  EVAL
             NEXT
          END
       »  x  STOF
    »
»
```

Exemple: (en Mode 5 FIX pour le résultat, et en 3 secondes)

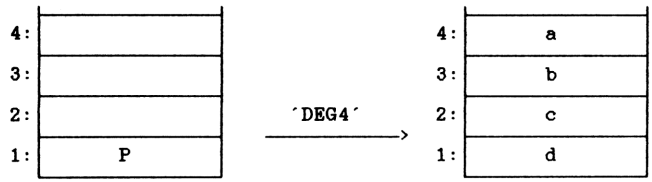


Dans ce cas particulier, le polynôme P possède deux racines réelles conjuguées, et une racine réelle égale sensiblement à 4.61347 .

Répertoire 'POLY' Programme 'DEG4'

ZEROS D'UN POLYNOME DE DEGRE 4.

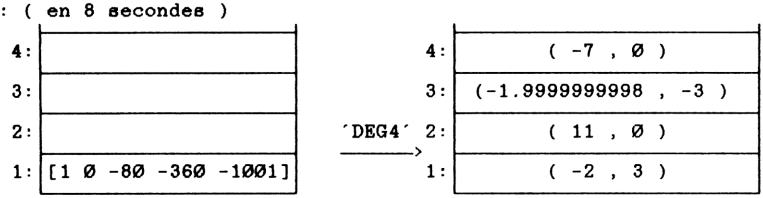
'DEG4' calcule les quatre zéros réels ou complexes a b c et d, d'un polynôme P de degré 4, à coefficients réels ou complexes. Le schéma fonctionnel est:



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':
    «   DUP   1   GET   /   ARRY→   DROP
        →   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))'  EVAL  THEN  2  'j'  STO  END
                IF  'ABS(t(3)) > ABS(t(j))'  EVAL  THEN  3  'j'  STO  END
                t   j   GET   √   SWAP   j   GET
                →   w   y
                «   IF   w   (Ø,Ø)   ==   THEN
                           a   4   /   NEG   DUP   DUP2
                       ELSE
                           Ø   w   'a*y/2-c'   EVAL   w   /   2   /   {3}
                           →ARRY   1   a   2   /   y   2   /  {3}   →ARRY
                           →   v1   v2
                           «   v1   v2   +   DEG2
                               v1   v2   -   DEG2
                           »
                       END
                »
            »
        »
    »
```

Exemple: (en 8 secondes)



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

Répertoire 'POLY' Programme 'BRST'

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 ,..., a_n ,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 P(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_0 ,p_0] puis on construit une suite [s_n ,p_n], 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 2Ø (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_n , p_n] (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.

Répertoire 'POLY' Programme 'BRST' (suite)

TEXTE DU PROGRAMME 'BRST'.

```
'BRST':
   «  DUP  SIZE  1  GET  Ø  Ø  Ø  Ø  Ø  Ø  .ØØØØØ1  2Ø
      →  a  n  v  s  p  b  d  k  eps  max
      «  IF  n  6  ≥  THEN
            [ 1  1 ]  'v'  STO
            DO
               Ø  'k'  STO
               DO
                  v  ARRY→  DROP  'p'  STO  's'  STO  Ø  'a(1)'  EVAL
                  2  n  FOR  i
                     DUP  s  *  3  PICK  p  *  +  'a(i)'  EVAL  +
                  NEXT
                  n  →ARRY  'b'  STO  DROP  v  Ø  Ø  'b(1)'  EVAL
                  2  n  1  -  FOR  i
                     ROT  DROP  DUP  s  *  3  PICK
                     p  *  +  'b(i)'  EVAL  +
                  NEXT
                  SWAP  DUP  4  ROLLD  { 2 2 }  →ARRY  INV
                  'b(n-1)'  EVAL  'b(n)'  EVAL  2  →ARRY  *
                  DUP  RNRM  v  RNRM  /  'd'  STO  -  'v'  STO
                  k  1  +  'k'  STO
               UNTIL  d  eps  <  k  max  ≥  OR  END
               IF  k  max  <  THEN
                  1  s  NEG  p  NEG  3  →ARRY  DEG2  n  2
                  'n'  STO  b  n  1  →LIST  RDM  'a'  STO
               END
               HALT
            UNTIL  n  6  <  k  max  <  AND  END
         END
         a  ELMG  DUP  SIZE  1  GET  'n'  STO
         IF  n  5  ==  THEN
            DEG4
         ELSE
            IF  n  4  ==  THEN
               DEG3
            ELSE  DEG2
            END
         END
      »
   »
```

Répertoire ´POLY´ Programme ´BRST´
(suite)

EXEMPLE D´UTILISATION DU PROGRAMME ´BRST´.

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

Au bout de 1 mn 13 s, on obtient sur la pile deux zéros complexes conjugués de P, à savoir : a_1 = (-1.26463Ø96509, -.357261654484)
et a_2 = (-1.26463Ø96509, .357261654484).

On fait alors CONT et au bout de de 32 s, on trouve à nouveau deux zéros complexes conjugués:
a_3 = (.442765764928, -1.17374Ø73Ø66)
et a_4 = (.442765764928, 1.17374Ø73Ø66).

On poursuit par CONT et au bout de 5Ø s, on trouve encore deux zéros complexes conjugués:
a_5 = (-.246722626138, -1.26288540248)
et a_6 = (-.246722626138, 1.26288540248).

On poursuit par CONT et au bout de 8 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 programe ´BRST´).
Ces zéros sont a_7 = (-.88465843109, -.959966681655)
a_8 = (-.88465843109, .959966681655)
a_9 = (.95324625739, .72511Ø51529)
et a_{10} = (.95324625739, -.72511Ø51529)

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^1Ø +2*X^9 +3*X^8 +..9*X^2 +1Ø*X +11´ (à l´aide du programme V X) et en utilisant le programme SOLVR de la HP28 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:

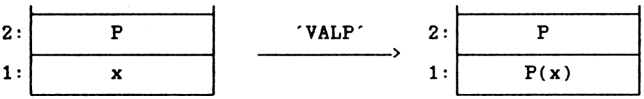
$$| a - \bar{a} | \approx | P(\bar{a}) | \ / \ | P'(\bar{a}) |$$

Pour effectuer ce calcul, on utilise les programmes ´DERIV´ (dérivation de P) et ´VALP´ (calcul des valeurs de P et de P´ en 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.Ø1E-9
Pour a_6 , on trouve $| a_6 - \bar{a}_6 | \approx$ 9.Ø5E-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-1Ø

Répertoire 'POLY' Programmes 'VALP' et 'RENV'

VALEUR D'UN POLYNOME EN UN POINT.

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



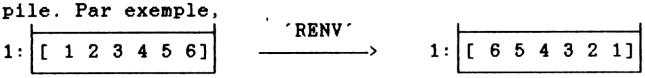

```
'VALP':
    «   Ø
        →   x   y
        «   DUP   SIZE   1   GET   { 1 }   1   3   ROLL   START
              GETI   y   x   *   +   'y'   STO
            NEXT
            DROP   y
        »
    »
```

Exemple: (en 1 seconde)

On peut aussi utiliser la fonction EVAL du HP28 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,



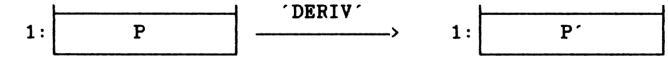

```
'RENV':
    «   DUP   SIZE   1   GET
        →   n
        «   IF   n   1   >   THEN
              ARRY→   DROP
              n   2   FOR   i   i   ROLLD   -1   STEP
              n   1   →LIST   →ARRY
            END
        »
    »
```

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

Répertoire 'POLY' Programme 'DERIV'

DERIVATION D'UN POLYNOME.

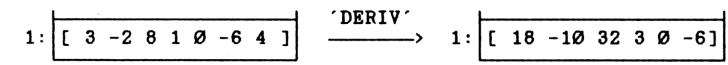
'DERIV' calcule le polynôme dérivé P' de P. Le schéma fonctionnel est:



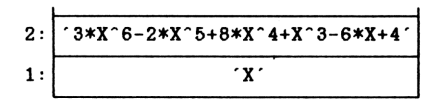
N.B: le programme 'DERIV' appelle le programme 'ELMG'.

```
'DERIV':
    «   ELMG   DUP   SIZE   1   GET   1   -
        →  p   d
        «  IF   d   Ø   ==   THEN
                 [ Ø ]
           ELSE
              p   { 1 }
              d   1   FOR   i
                  GETI   i   *   3   ROLLD
              -1   STEP
              DROP2    d    1    →LIST   →ARRY
           END
        »
    »
```

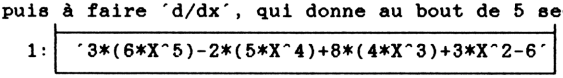
Exemple: (en un peu plus d'une seconde)



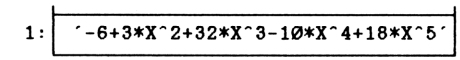
Remarque:
Cette méthode est beaucoup plus rapide que celle qui consiste à entrer:



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



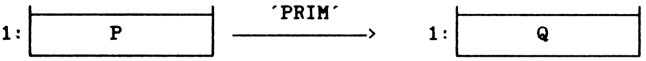
Il faut alors passer dans le menu ALGEBRA, faire COLCT pour obtenir au bout de 8 nouvelles secondes:



Répertoire 'POLY' Programmes 'PRIM' et 'INTP'

PRIMITIVATION D'UN POLYNOME.

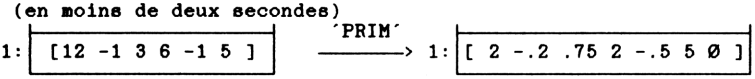
'PRIM' calcule la primitive Q s'annulant en Ø du polynôme P. Le schéma fonctionnel est:



N.B: Le programme 'PRIM' appelle le programme 'ELMG'.

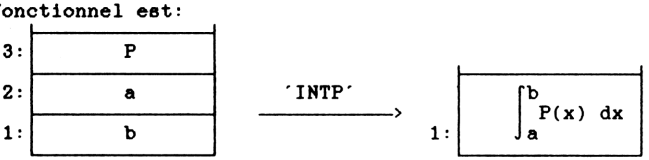
```
'PRIM':
    «   ELMG   DUP   SIZE   1   GET
        →  n
        «   IF   DUP   [ Ø ]   ≠   THEN
              { 1 }  n   2   FOR   i
                 DUP2   GET   i   /   PUTI
              -1   STEP
              DROP   n   1   +   1   →LIST   RDM
            END
        »
    »
```

Exemple: (en moins de deux secondes)



INTEGRALE D'UN POLYNOME SUR UN SEGMENT.

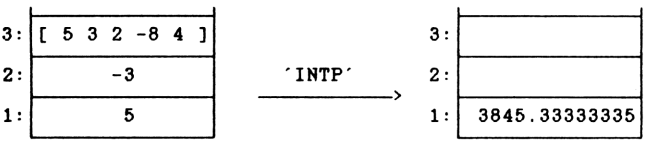
'INTP' calcule l'intégrale du polynôme P sur le segment [a,b]. Le schéma fonctionnel est:



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

```
'INTP':
    «   →   a   b
        «   PRIM   b   VALP   SWAP   a   VALP   SWAP   DROP   -
        »
    »
```

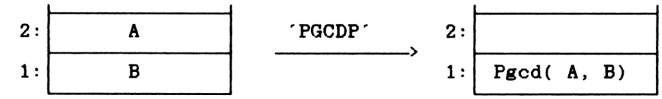
Exemple:



Répertoire 'POLY' — Programmes 'PGCDP' et 'PPMP'

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

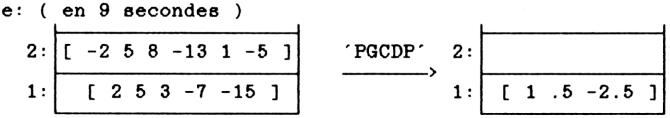
'PGCDP' calcule le pgcd (plus grand commun diviseur) de deux polynômes A et B, suivant le schéma fonctionnel:



Le polynôme obtenu est normalisé (son terme de plus haut degré est 1).
N.B: le programme 'PGCDP' appelle le programme 'DIV'.

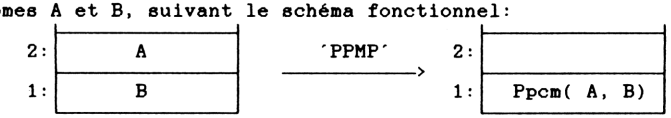
```
'PGCDP':
    «   →  a  b
        «  WHILE  a  b  DUP  [ Ø ]  ≠
           REPEAT
              DIV  b  'a'  STO  'b'  STO  DROP
           END
           DROP  DUP  { 1 }  GET  /
        »
    »
```

Exemple: (en 9 secondes)



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

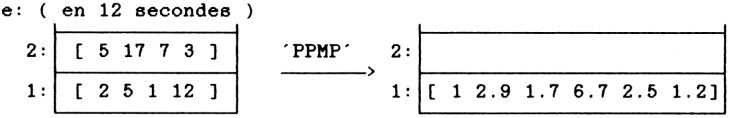
'PPMP' calcule le ppcm (plus petit commun multiple) de deux polynômes A et B, suivant le schéma fonctionnel:



Le polynôme obtenu est normalisé (son terme de plus haut degré est 1).
N.B: le programme 'PPMP' appelle les programmes 'PGCDP', 'DIV', 'PRODP'.

```
'PPMP':
    «   DUP2
        →  a  b
        «  PGCDP  b  SWAP  DIV  DROP
           a  PRODP  DUP  { 1 }  GET  /
        »
    »
```

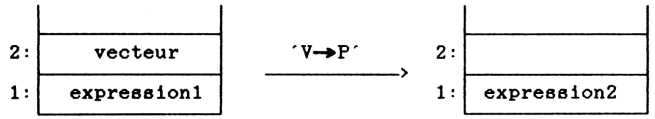
Exemple: (en 12 secondes)



Répertoire 'POLY' Programme 'V→P'

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:

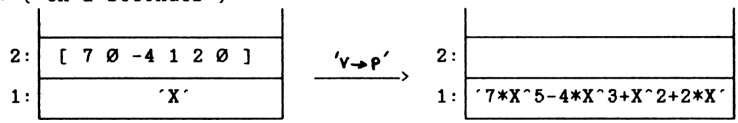


où "vecteur" est le vecteur représentant le polynôme P, où "expresssion1" est une expression algébrique devant se substituer à l'indéterminée du polynôme P, et où "expression2" est l'expression algébrique ainsi obtenue pour le polynôme P.

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

```
'V→P':
    «  →  var
       «  DUP  SIZE  1  GET
          →  p  n
          «  Ø  1  n  FOR  i
                'p'  i  GET  var  n  i  -  ^  *  +
             NEXT
          »
       »
    »
```

Exemple: (en 2 secondes)



On peut remplacer 'X' par d'autres expresssions 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.

Répertoire 'POLY' Programme 'TCHEB'

POLYNOMES DE TCHEBICHEV.

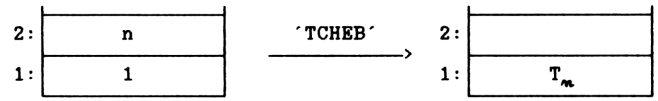
'TCHEB' calcule les polynômes de Tchebichev de première espèce T_m ,et de seconde espèce U_m .

Les polynômes T_m sont définis par:
$T_0(x)=1$, $T_1(x)=x$, et pour tout $n \geq 2$, $T_m(x) = 2xT_{m-1}(x) - T_{m-2}(x)$.

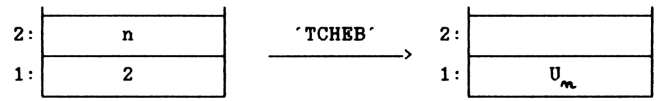
Les polynômes U_m sont définis par:
$U_0(x)=1$, $U_1(x)=2x$, et pour tout $n \geq 2$, $U_m(x) = 2xU_{m-1}(x) - U_{m-2}(x)$.

Le schéma fonctionnel est le suivant:

* pour obtenir les polynômes de Tchebichev de première espèce:

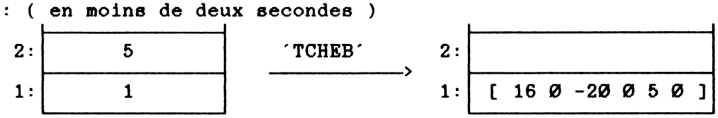


* pour obtenir les polynômes de Tchebichev de seconde espèce:




```
'TCHEB':
   «   IF   DUP   1   ==   THEN   DROP   [ 1 Ø ]
         ELSE   IF   2   ==   THEN   [ 2 Ø ]   ELSE   ABORT   END
       END  [ 1 ]
       →  n   b   a
       «   IF   n   Ø   ==   THEN   a
             ELSE   IF   n   1   ==   THEN  b  ELSE
               3   n   1   +   FOR   k
                 b   DUP   k   1   →LIST   RDM   2   *  Ø   Ø   a
                 ARRY→   DROP   k   →ARRY   -   'b'   STO   'a'   STO
               NEXT  b
             END
           END
       »
   »
```

Exemple: (en moins de deux secondes)



Car $T_5(X) = 16*X^5 -2Ø*X^3 +5*X$.

Répertoire 'POLY' Programmes 'ELMG' et 'ELMD'

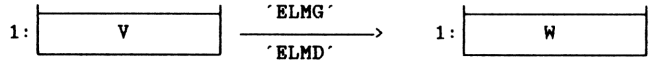
ELIMINATION DES COEFFICIENTS NULS A DROITE OU A GAUCHE.

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

Ces deux programmes sont appelés par plusieurs programmes du répertoire 'POLY'. Ils sont en particulier très utiles dans les programmes de division 'DIV' et 'DIVPC'.

Pour combattre les erreurs d'arrondis (en particulier dans 'DIV' et 'DIVPC'), 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:



où W est le vecteur résultant du tronquage éventuel de V.

'ELMG':

```
«    DUP   RNRM
     IF   .ØØØØ1   <   THEN   DROP   [ Ø ]
          ELSE   ARRY→   1   GET
          →   k
          «   WHILE   k   ROLL   DUP   ABS   .ØØØØ1   <
              REPEAT   DROP   k   1   -   'k'   STO   END
              k   ROLLD   k   1   →LIST   →ARRY
          »
     END
»
```

'ELMD'

```
«    DUP   RNRM
     IF   .ØØØØ1   <   THEN   DROP   [ Ø ]
          ELSE   ARRY→   1   GET
          →   k
          «   WHILE   DUP   ABS   .ØØØØ1   <
              REPEAT   DROP   k   1   -   'k'   STO   END
              k   1   →LIST   →ARRY
          »
     END
»
```

Exemple:

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

CALCUL MATRICIEL

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

Dans ce dommaine, la HP28S 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 . Les programmes du répertoire 'MATR' sont articulés autour des problèmes suivants:

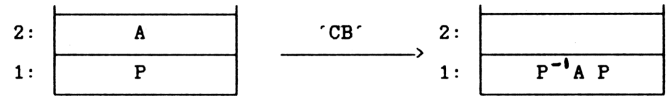
- Recherche de valeurs propres:
  * 'PCAR' donne le polynôme caractéristique d'une matrice carrée.
  * 'VP234' donne les valeurs propres d'une matrice carrée d'ordre 2,3, ou 4.
  * 'DEFL' permet d'approcher les valeurs propres et vecteurs propres dans le cas de matrices de dimension supérieure à 4.
  * 'VECTP' donne la ou les équations du sous-espace propre d'une matrice carrée pour une valeur propre donnée.
- Rang d'une matrice carrée:
  * 'RANG' calcule le rang d'une matrice quelconque (par utilisation de la méthode du pivot de Gauss).
- Résolution symbolique de systèmes linéaires:
  * '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.
- Manipulations de lignes et de colonnes (pour mener pas à pas la méthode du pivot de Gauss):
  * 'GETC' extrait une colonne quelconque d'une matrice.
  * 'GETL' extrait une ligne quelconque d'une matrice.
  * 'PUTC' place une ligne dans une matrice.
  * 'PUTL' place une colonne dans une matrice.
  * 'ECHL' échange deux lignes.
  * 'ECHC' échange deux colonnes.
- 'A=LU' permet de décomposer une matrice carrée en le produit d'une matrice triangulaire inférieure à diagonale unité et d'une matrice triangulaire supérieure.
- 'PUISS' permet de calculer les puissances quelconques d'une matrice carrée.
- 'CRATB' permet de créer un tableau (matrice ou vecteur) dont le terme général répond à une formule donnée.
- 'TR' calcule la trace d'une matrice carrée.
- 'CB' effectue le changement de matrice d'une application linéaire (connaissant la matrice initiale et la matrice de changement de base).

Répertoire 'MATR'. Programmes 'CB' et 'TR'.

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.

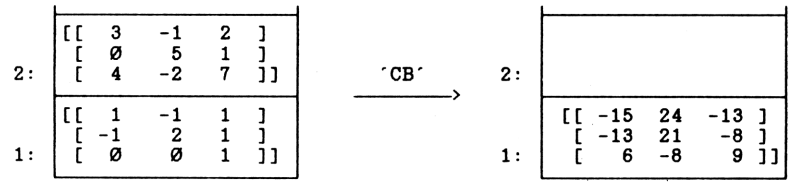
Le schéma fonctionnel est:



'CB':

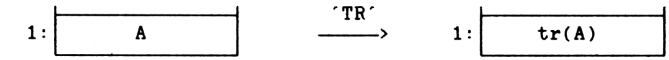
```
« →  a  p
   « a  p  /  p  *
   »
»
```

Exemple:



CALCUL DE LA TRACE D'UNE MATRICE CARREE

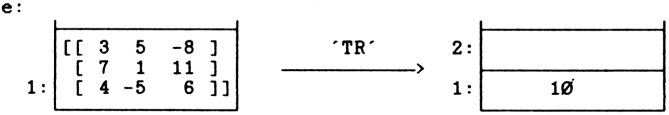
'TR' calcule la trace tr(A) (c'est à dire la somme des coefficients diagonaux) d'une matrice carrée A, suivant le schéma fonctionnel:



'TR':

```
« DUP  SIZE  2  GET  Ø  →  a  n  t
   « Ø  1  n  FOR  i
        'a(i,i)'  EVAL  +
      NEXT
   »
»
```

Exemple:

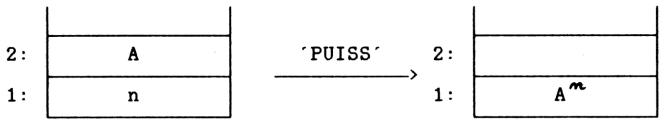


Répertoire 'MATR'. Programme 'PUISS'.

PUISSANCES D'UNE MATRICE CARREE.

'PUISS' Calcule les puissances A^m (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.

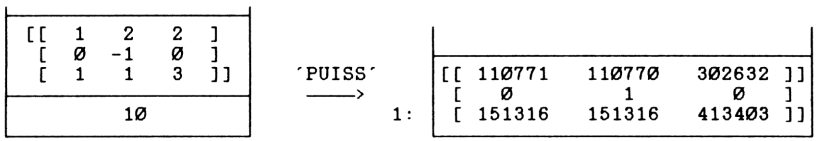
Le schéma fonctionnel est:



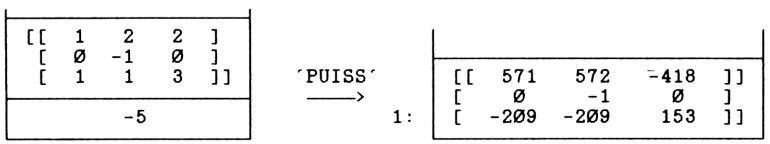
'PUISS':

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

Exemple 1: (en 2.5 secondes)



Exemple 2: (en 2 secondes)



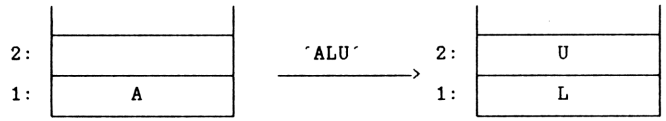
Répertoire 'MATR'. Programme 'ALU'.

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 les variables 'L' et 'U' qui se trouvent éventuellement dans le répertoire.

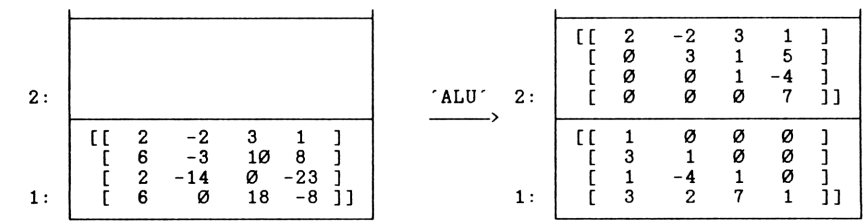
Le schéma fonctionnel est:



'ALU':

```
«  ARRY→  1  GET  DUP  IDN  'L'  STO  {}  'U'  STO
   → d
    «  1  d  FOR  i
          d  →ARRY  U  +  'U'  STO
       NEXT
       1  d  1  -  FOR  col
          'U'  col  GET  col  GET
          →  piv
          «  col  1  +  d  FOR  j
                'L'  j  col  2  →LIST  'U'  j  GET
                col  GET  piv  /  PUT  'U'  j
                'U(j)-U(col)*L(j,col)'  EVAL  PUT
             NEXT
          »
       NEXT
       1  d  FOR  i
          'U'  i  GET  ARRY→  DROP
       NEXT
       d  d  2  →LIST  →ARRY  L  { L U }  PURGE
    »
»
```

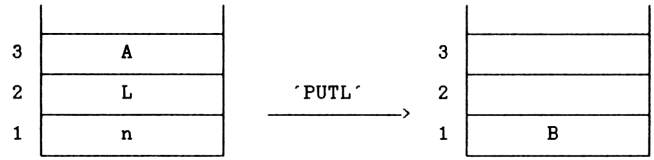
Exemple: (en 6 secondes)



Répertoire ´**MATR**´. Programme ´**PUTL**´

PLACER UNE LIGNE DONNEE DANS UNE MATRICE DONNEE.

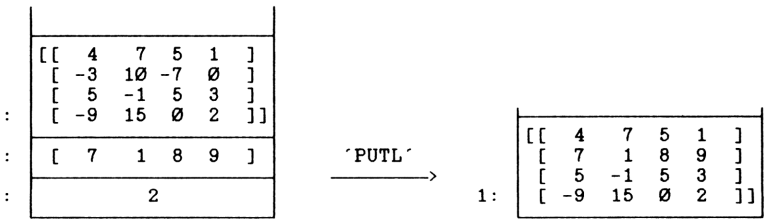
´PUTL´ place la ligne L dans la matrice A, au niveau de la ligne de numéro n, transformant ainsi A en une matrice B, suivant le schéma fonctionnel suivant:



´PUTL´:

```
«   →  a   ligne   k
     «  a   k   1   2   →LIST
        1   a   SIZE   2   GET   FOR   i
            ´ligne(i)´   EVAL   PUTI
        NEXT
        DROP
     »
»
```

Exemple:



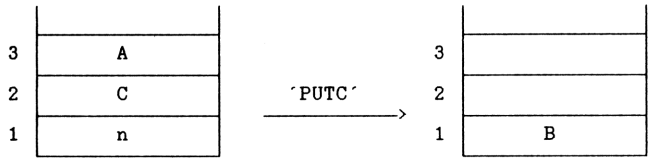
Répertoire 'MATR'. Programme 'PUTC'

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.

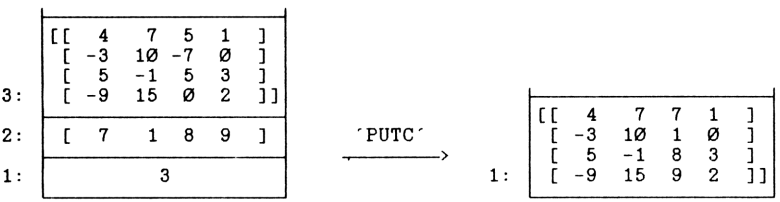
schéma fonctionnel:



'PUTC':

```
«  SWAP  DUP  SIZE  SIZE
   IF  2  ==  THEN  ARRY→  1  GET  →ARRY  END
   SWAP  ROT  TRN  3  ROLLD  PUTL  TRN
»
```

Exemple:

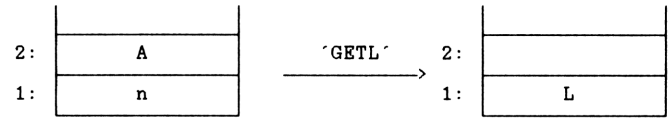


Répertoire ´**MATR**´. Programme ´**GETL**´

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.

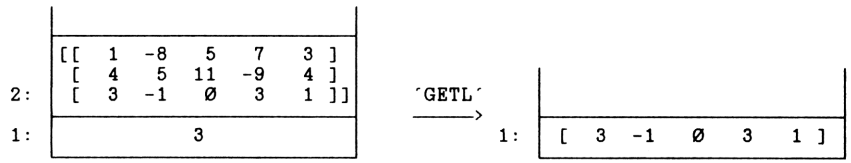
schéma fonctionnel:



´GETL´:

```
«  →  k
   «  TRN  DUP  SIZE  2  2  SUB  Ø  CON
      k  1  PUT  *
   »
»
```

Exemple:

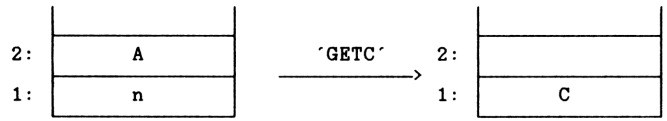


Répertoire ´**MATR**´. Programme ´**GETC**´

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.

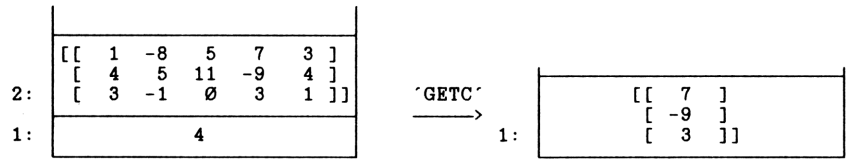
schéma fonctionnel:



´GETC´:

```
«   →  k
     «  DUP   SIZE   1   1   PUT   Ø   CON   TRN
        k   1   2   →LIST   1   PUT   *
     »
»
```

Exemple:

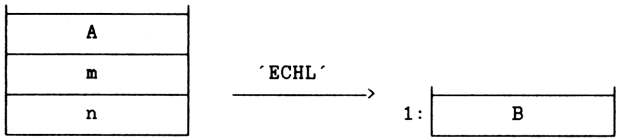


Répertoire 'MATR'. Programmes 'ECHL' et 'ECHC'

ECHANGE DE DEUX LIGNES D'UNE MATRICE.

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

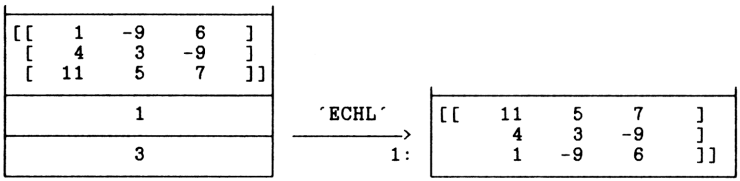
schéma fonctionnel:



'ECHL':

```
«   →  m  n
    «  DUP  SIZE  1  GET  IDN  m  m  2  →LIST  Ø  PUT
       n  n  2  →LIST  Ø  PUT  m  n  2  →LIST  1  PUT
       n  m  2  →LIST  1  PUT  SWAP  *
    »
»
```

Exemple:



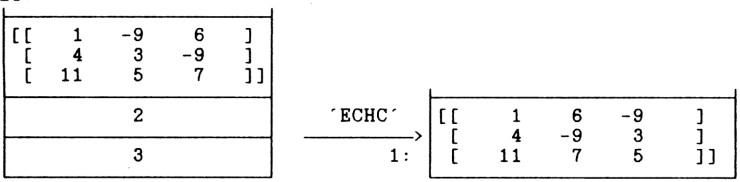
ECHANGE DE DEUX COLONNES D'UNE MATRICE.

'ECHC' échange les colonnes de numéros n et m de la matrice A, transformant ainsi A en une matrice B. L schéma fonctionnel est analogue à celui de 'ECHL'. D'ailleurs 'ECHC' appelle 'ECHL'.

'ECHC':

```
«   →  m  n
    «  TRN  m  n  ECHL  TRN
    »
»
```

Exemple:



Répertoire 'MATR'. Programme 'CRTAB'

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):

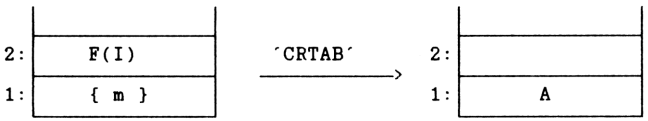
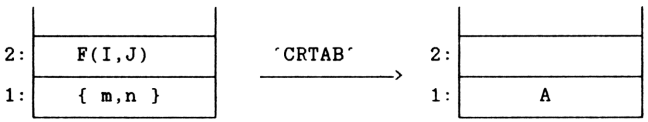


schéma fonctionnel (création d'une matrice):



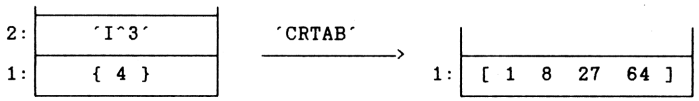
'CRTAB':

```
« → f dim
   « dim Ø CON dim GETI DROP
     DO
        DUP LIST→ IF 2 == THEN 'J' STO END
        'I' STO f EVAL PUTI
     UNTIL DUP LIST→ IF 2 == THEN * END
           1 == END
     DROP { I J } PURGE
   »
»
```

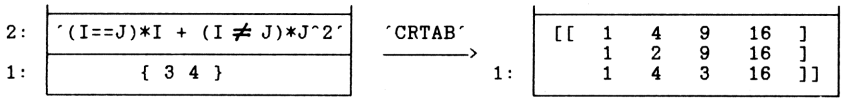
Répertoire 'MATR'. Programme 'CRTAB' (suite)

EXEMPLES D'UTILISATION DU PROGRAMME 'CRATB'.

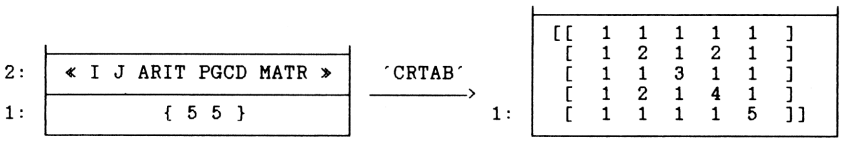
Exemple 1: Création d'un vecteur de longueur 4 de terme général A(I)=I^3. (temps de calcul inférieur à deux secondes).



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≠J.
(temps de calcul 5 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'.
(Le temps de calcul est environ de 10 secondes)



Répertoire 'MATR'. Programme 'PCAR'

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:

$$P(x) = a_m x^m + a_{m-1} x^{m-1} + \ldots\ldots + a_k x^k + \ldots\ldots + a_1 x + a_0 .$$

En particulier:

$a_m = 1$, $a_{m-1} = -tr(A)$ (tr(A)=trace de A), $a_0 = (-1)^m \det(A)$.

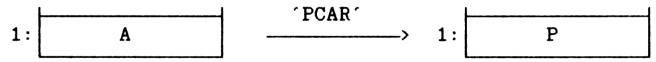
Les racines du polynôme caractéristique P de A sont aussi les valeurs propres de A.

Le polynôme P est obtenu sous la forme du vecteur:

$$[\ a_m \quad a_{m-1} \quad \ldots \quad a_1 \quad a_0\ ].$$

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

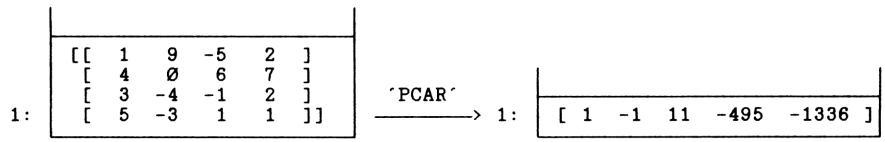
schéma fonctionnel:



'PCAR':

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

Exemple: (temps de calcul 6 secondes)



Répertoire 'MATR'. Programme 'VP234'

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=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:

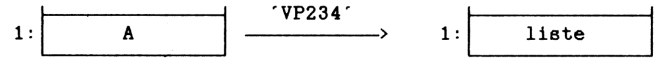
- calcul du polynôme caractéristique (programme 'PCAR').
- passage dans le répertoire 'POLY'.
- calcul des racines du polynôme caractéristique (utilisation des programmes 'DEG2', 'DEG3', 'DEG4' du répertoire 'POLY').
- retour dans le répertoire 'MATR'.

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.
5 secondes pour une matrice carrée d'ordre 3.
12 secondes pour une matrice carrée d'ordre 4.

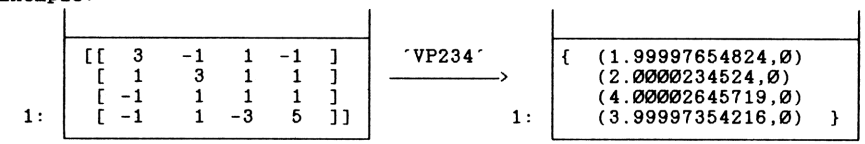
schéma fonctionnel:



'VP234':

```
«   DUP   SIZE   2   GET   RCLF   STD
    →  n   x
    «  IF  "2 3 4"   n   →STR   POS
       THEN   PCAR   POLY   "DEG"   n   →STR
              +   STR→   EVAL   MATR
       END
       n   →LIST   x   STOF
    »
»
```

Exemple:



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

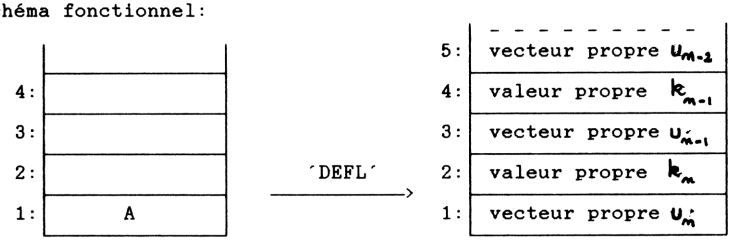
2 est une valeur propre double.
4 est une valeur propre double.

Répertoire 'MATR'. Programme 'DEFL'

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 $k_1, k_2, \ldots, k_m$ telles que $|k_1| < |k_2| < \ldots < |k_m|$.

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 donnée 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. Vous trouverez certainement des résultats différents sur l'exemple. Il ne s'agit pas d'une erreur. En effet le programme 'DEFL' utilise le générateur de nombres aléatoires 'RAND' de la HP28S.

Exemple:
avec la matrice A égale à:

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

On trouve successivement (en mode 8 FIX):

```
En 17 secondes: k =-24.9999989 et
     u = [ -2.Ø116359E-9   Ø.4Ø824829   Ø.4Ø824829    -.81649658  ].
En 31 secondes: k =1Ø.ØØØØØØØ1 et
     u = [ -Ø.5ØØØØØØØ     Ø.5ØØØØØØØ   Ø.5ØØØØØØØ    -Ø.5ØØØØØØØ ].
En 29 secondes: k =-3.ØØØØØØØØ et
     u = [ 3.97471899E-9  -Ø.7Ø71Ø678  -3.583Ø6366E-9  Ø.7Ø71Ø678 ].
En 2Ø secondes: k = 1.ØØØØØØØØ et
     u = [ Ø.63245553     -.316222777   -.316222777     Ø.63245553 ].
```

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

Répertoire ´MATR´. Programme ´DEFL´ (Suite).

TEXTE DU PROGRAMME ´DEFL´

´DEFL´:

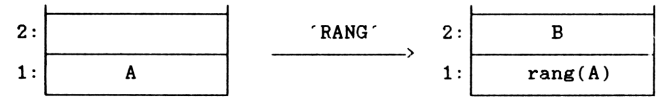
```
«   IF   DUP   TYPE   THEN   .ØØØØØØØ1   END
    SWAP   DUP   SIZE   2   2   SUB   DUP   Ø   CON
    DUP   ROT   1   GET

    →   eps   a   old   new   p

    «   «   →   mat
            «   old   {1}
                1   p   START   RAND   PUTI   NEXT   DROP
                DO
                   ´old´   STO   ´mat*old´   EVAL   DUP   ABS   /
                   IF   DUP   old   DOT   Ø   <   THEN   NEG   END
                   DUP   ´new´   STO
                UNTIL   ´ABS(new-old)<eps´   EVAL   END
                ´mat*new´   EVAL   DOT   new   ABS   /   new
            »
        »

        →   vpmax

        «   1   p   FOR   k
                a   vpmax   EVAL
                IF   k   p   <   THEN
                   HALT
                   DUP2   a   TRN   vpmax   EVAL   ROT   DUP2   DOT   /
                   ARRY→   DROP   p   1   2   →LIST   →ARRY   SWAP
                   ARRY→   DROP   1   p   2   →LIST   →ARRY   *
                   3   ROLLD   +   2   /   *   a   -   NEG   ´a´   STO
                END
            NEXT
        »
    »
»
```

Répertoire 'MATR'. Programme 'RANG'

CALCUL DU RANG D'UNE MATRICE.

'RANG' calcule le rang d'une matrice carrée 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:




```
'RANG':
  «  DUP  TYPE  Ø  ==  DUP  IF  THEN  SWAP  DROP  END  SWAP
     DUP  SIZE  LIST→  DROP  1  1  Ø  Ø  .ØØØØØ1
     →  flag  a  nl  nc  i  j  rg  lp  eps
     «  «  Ø  i  nl  FOR  ii  'a(ii,j)'  EVAL  ABS  DUP2
           IF  <  THEN  ii  'lp'  STO  SWAP  END
           DROP  NEXT
        »
        «  nl  IDN  i  i  2  →LIST  Ø  PUT
           lp  lp  2  →LIST  Ø  PUT  i  lp  2  →LIST  1  PUT
           lp  i  2  →LIST  1  PUT  a  *  'a'  STO
        »
        «  'a(i,j)'  EVAL  →  piv
           «  nl  IDN
              IF  flag  THEN  1  ELSE  i  1  +  END
              nl  FOR  ii
                 IF  ii  i  ≠
                    THEN  ii  i  2  →LIST  '-a(ii,j)/piv'
                          EVAL  PUT  END
              NEXT  a  *  'a'  STO
           »
        »
        «  1  nl  FOR  i  1  nc  FOR  j
             IF  'a(i,j)'  EVAL  ABS  eps  <
                THEN  i  j  2  →LIST  Ø  PUT  END
           NEXT  NEXT
        »
        →  cp  ech  pivote  annul
        «  WHILE  i  nl  ≤  j  nc  ≤  AND
           REPEAT  cp  EVAL
              IF  eps  >  THEN
                 rg  1  +  'rg'  STO
                 IF  i  lp  <  THEN  ech  EVAL  END
                 IF  i  nl  <  flag  OR
                    THEN  pivote  EVAL  END
                 i  1  +  'i'  STO
              END
              j  1  +  'j'  STO
           END
           a  annul  EVAL  rg
        »
     »
  »
```

Répertoire 'MATR'. Programme 'RANG' (Suite)

EXEMPLES D'UTILISATION DU PROGRAMME 'RANG'.

Exemple 1: (en environ 1Ø secondes)

```
1:  [[   2   -5   3   1   ]          'RANG'  2:  [[ 5   -9    6     2  ]
    [   3   -7   3  -1   ]          ------->       Ø  -1.6   -.6  -2.2 ]
    [   5   -9   6   2   ]                         Ø    Ø  -2.25 -2.25]
    [   4   -6   3   1   ]]                        Ø    Ø    Ø     1  ]]

                                               1:  [          4          ]
```

Exemple 2: (en environ 19 secondes)

La matrice A égale à

```
[[  18   -12   1Ø   11   -9   -19   1Ø  ]
 [   4     5    9    5    2     1    3  ]
 [   1     8    5    2    4     6    1  ]
 [  1Ø     5    3    4    1     2    5  ]
 [  -6     Ø    6    1    1    -1   -2  ]]
```

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

```
[[    18     -12     1Ø      11     -9     -19     1Ø  ]
 [     Ø   11.67  -2.56   -2.11      6   12.56   -.56 ]
 [     Ø       Ø   8.46    3.94    .Ø6  -3.Ø3   1.14 ]
 [     Ø       Ø      Ø       Ø      Ø      Ø      Ø  ]
 [     Ø       Ø      Ø       Ø      Ø      Ø      Ø  ]
 [     Ø       Ø      Ø       Ø      Ø      Ø      Ø  ]]
```

N.B: Lors de la procédure de calcul du rang, tout coefficient dont la valeur absolue est inférieure à .ØØØØØ1 est considéré comme nul (ceci afin de lutter contre les erreurs d'arrondi). On peut modifier cette valeur, qui apparaît à la deuxième ligne du programme.

Variante:

Si, avant d'appeler 'RANG', on place un réel quelconque 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      Ø      Ø     5.39  -2.88  -3.45   8.43 ]
 [     Ø   11.67      Ø    -.92   6.Ø2  11.64   -.21 ]
 [     Ø       Ø   8.46    3.94    .Ø6  -3.Ø3   1.14 ]
 [     Ø       Ø      Ø       Ø      Ø      Ø      Ø  ]
 [     Ø       Ø      Ø       Ø      Ø      Ø      Ø  ]
 [     Ø       Ø      Ø       Ø      Ø      Ø      Ø  ]]
```

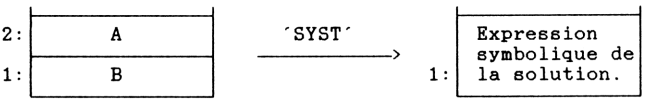
Répertoire ´**MATR**´. Programme ´**SYST**´

RESOLUTION SYMBOLIQUE D´UN SYSTEME D´EQUATIONS LINEAIRES.

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

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

Si on note A la matrice du sytème 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 sytème: 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.

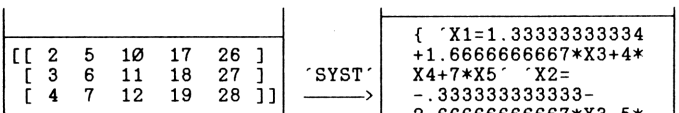
  et/ou de la forme:
  ´X2=3*X4-2*X5´ (par exemple) si la résolution du système conduit à cette expression de l´inconnue X2 en fonction des inconnues X4 et X5 (ces deux-ci étant alors indéterminées).

(Les inconnues du sytème sont notées X1, X2, X3, ...)

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

Exemple : (en 16 secondes)
Résolution du système

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



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

$X_1 = 4/3 + 5X_3/3 + 4X_4 + 7X_5$

$X_2 = -1/3 - 8X_3/3 - 5X_4 - 8X_5$.

Répertoire 'MATR'. Programme 'SYST' (Suite).

TEXTE DU PROGRAMME 'SYST'

'SYST':

```
«  RCLF  →  v  w

   «  TRN  ARRY→

      →  dim

      «  v  ARRY→  DROP  dim  DUP  1  GET  1  +  1
         SWAP  PUT  DUP  'dim'  STO  →ARRY  TRN  Ø
         RANG  dim  1  GET  Ø
         «  "{ X"  SWAP  →STR  +  "}"  +  STR→  1  GET  »

         →  a  rg  nc  j  var

         «  STD  {}
            1  rg  FOR  i
               DO
                  j  1  +  'j'  STO
               UNTIL  'a(i,j)'  EVAL  END
               IF  j  nc  ==
                 THEN  DROP  "Pas de solution"  ABORT  END
               j  var  EVAL
               'a(i,j)'  EVAL
               →  piv

               «  '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
         »

      »  w  STOF
   »
»
```

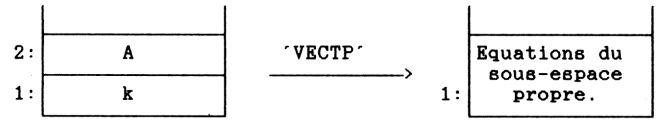
Répertoire 'MATR'. Programme 'VECTP'

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 celà, le programme 'SYST' est appelé afin de résoudre symboliquement le système:

(A-kI)X=Ø, 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,....

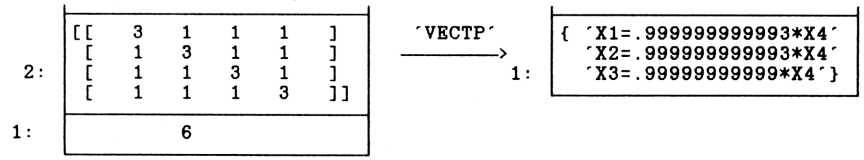
schéma fonctionnel:



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=Ø' 'X2=Ø' 'X3=Ø' ... }

```
'VECTP':
    «  →  vp
       «   DUP  SIZE   1   GET
           →  n
           «  n   IDN   vp   *   -   n   1   →LIST   Ø   CON   SYST
           »
       »
    »
```

Exemple: (en 17 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 Ø Ø 1], [-1 Ø 1 Ø], [-1 1 Ø Ø] (obtenus en donnant la valeur 1 à l'une des variables X2,X3,X4, et la valeur Ø aux deux autres).

ANALYSE

Sous ce titre un peu vague, et dans le répertoire ´ANALY´, on trouvera un certain nombre de programmes traitant de problèmes classiques comme:

* 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 ´ANALY´ contient:

´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 ceratin nombre de points.

´LAGRA´: Calcul du polynôme interpolateur de Lagrange "passant" par une famille de points donnée.

´TRACE´: En liaison avec les programmes ´APMC´ et ´LAGRA´, tracé de la famille de points utilisée, puis la courbe obtenue.

´SXY´ : Résolution numérique approchée, par la méthode de Newton, d´un système de deux équations à deux inconnues $F(X,Y)=Ø$ $G(X,Y)=Ø$

´SXYZ´ : Résolution numérique approchée, par la méthode de Newton, d´un système de 3 équations à 3 inconnues $F(X,Y,Z)=Ø$ $G(X,Y,Z)=Ø$ $H(X,Y,Z)=Ø$

´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.

Répertoire ´**ANALY**´ programme ´**APMC**´

APPROXIMATION AU SENS DES MOINDRES CARRES DISCRETS.

Soit (X1,Y1), (X2,Y2),... (Xi,Yi),..., (Xn,Yn) n points du plan (d´abscisses distinctes deux à deux).

Soit une famille de p fonctions F1, F2, ..., Fi,..., Fp, linéairement indépendantes, avec p<n.

Alors il existe une unique application F, combinaison linéaire des Fk, qui réalise la meilleure approximation des points (Xi,Yi) au sens des moindres carrés discrets, c´est-à-dire qui minimise la différence:

$$\sum_{i=1}^{i=n} (Yi - F(Xi))^2 .$$

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 (Xi,Yi), eux mêmes écrits sous forme de nombres complexes.
  exemple: ´DATA´ <— [(-2,3) (-1,Ø) (Ø,1) (1,5) (2,3)]

* placer dans une variable nommée ´FONC´ la liste des paramètres permettant d´identifier les fonctions Fk.

  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´ Ø 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 'ANALY' programme 'APMC' (suite)

TEXTE DU PROGRAMME 'APMC' ET EXEMPLE D'UTILISATION.

```
'APMC':
    «   DATA  C→R  DUP  SIZE  1  GET  DUP  FONC  4  GET  SWAP  2  →LIST
        Ø   CON   FONC   LIST→  DROP   3   DUPN   1   -   *   +   Ø
        →   abs   ord   col   a   fonc   kmin   stepk   lig   kmax   sol
        «   a   { 1 1 }   kmin   kmax   FOR   i
              i   'K'   STO
              1   col   FOR   j
                  'abs'   j   GET   'X'   STO   fonc   EVAL   PUTI
              NEXT
            stepk   STEP
            DROP   'a'   STO   a   DUP   DUP   TRN   *
            →  x   y
            «   x   y   /   x   y   3   PICK   RSD   y   /   +
            »
            ord   *   'sol'   STO   { X }   PURGE   kmin   'K'   STO
            Ø   1   lig   FOR   i
                'sol'   i   GET   fonc   EVAL   *   +   K   stepk   +   'K'   STO
            NEXT
            STEQ   { K }   PURGE
        »
    »
```

Exemple:

On place le vecteur [(-3,-1) (-2,1) (-1,2) (1,2) (2,1) (3,Ø)] dans la variable 'DATA'.

On cherche l'application réalisant la meilleure approximation (au sens des moindres carrés) du nuage des points (-3,-1), ..., (3,Ø), et qui s'écrive sous la forme:

F(X)= a + b*cos(X) + c*cos(2*X) + d*cos(3*X) + e*cos(4*X).

On met donc la liste { 'COS(K*X)' Ø 1 5 } dans la variable 'FONC'.

On appelle alors 'APMC'. L'éxécution du programme dure 15 secondes.
'APMC' met dans la variable 'EQ' l'expression algébrique:
(en mode 3 FIX):
'1.Ø3Ø+Ø.4Ø1*COS(X)+Ø.349*COS(2*X)+Ø.14Ø*COS(3*X)-1.588*COS(4*X)'

L'évaluation de 'EQ' donne les résultats suivants: (en mode 3 FIX).

X	-3	-2	-1	1	2	3
F(X)	-Ø.5ØØ	1.ØØØ	2.ØØØ	2.ØØØ	1.ØØØ	-Ø.5ØØ

(en -2,-1,1,2 les valeurs obtenues sont exactes à 1E-11 près).

Remarque: les familles de fonctions les plus utilisées sont:
Fk(X)=cos(kX), Fk(X)=sin(kX), Fk(X)=X^K, Fk(X)=exp(kX).

Répertoire ´**ANALY**´ programme ´**LAGRA**´

CALCUL DU POLYNOME INTERPOLATEUR DE LAGRANGE.

Soit (X1,Y1), (X2,Y2),... (Xi,Yi),..., (Xn,Yn) 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(Xi)=Yi. On dit que P est le polynôme interpolateur de Lagrange correspondant au nuage des points (Xi,Yi).

´LAGRA´ 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 [(X1,Y1) (X2,Y2) ... (Xn,Yn)] dont les éléments sont les nombres complexes représentant les points (Xi,Yi).

´LAGRA´:

```
«  DATA  C→R  DUP  SIZE  1  GET  DUP  DUP  2  →LIST  Ø  CON  Ø
   →  abs  ord  n  a  x
   «  ord  a  { 1 1 }
      1  n  FOR  i
         ´abs´  i  GET  ´x´  STO
         Ø  n  1  -  FOR  j
            x  j  ^  PUTI
         NEXT
      NEXT  DROP
      →  p  q
      «  p  q  /  p  q  3  PICK  RSD  q  /  +
      »
      ´abs´  STO
      Ø  1  n  FOR  i
         ´abs´  i  GET  ´X´  i  1  -  ^  *  +
      NEXT
      STEQ
   »
»
```

Exemple:

Après avoir mis [(-3,3) (-2,1) (-1,2) (1,3) (2,-1) (3,2)] dans ´DATA´ et avoir éxécuté ´LAGRA´, on trouve dans ´EQ´ (en 11 secondes) :

´4 + 1.Ø3333333333*X - 1.66666666667*X^2 - .583333333333*X^3 +.166666666667*X^4+.Ø5*X^5´.

Effectivement, ce polynôme vérifie:
P(-3)=3, P(-2)=.99999999999, P(-1)=2, P(1)=3, P(2)=-1.ØØØØØØØØØØ1, P(3)=2.

Répertoire 'ANALY' programme 'TRACE'

TRACE DE POINTS ET D'UNE COURBE LES APPROCHANT.

La variable 'DATA' est supposée contenir un vecteur de nombres complexes (Xi,Yi). 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'.

Le programme 'TRACE' est particulièrement destiné à être appelé après les programmes 'LAGRA' (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 (Xi,Yi) et la courbe qui en réalise une approximation.

A la fin du tracé, il est possible de numériser des points à l'aide du réticule (instruction DGTIZ).
On quitte le programme par un appui sur 'ON'. On trouve alors au niveau 1 de la pile une chaîne de caractères représentant l'image obtenue (instruction LCD).

'TRACE':

```
«   CLLCD   DATA   ARRY→   1   GET
    →  i
    «  1   i   START
           PIXEL
       NEXT
       DRAW   DGTIZ   LCD→
    »
»
```

Répertoire ´**ANALY**´ programme ´**SXY**´

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)=Ø
G(X,Y)=Ø

Le principe est le suivant:

On définit une suite (Xn,Yn) par la donnée d´un point initial (XØ,YØ) et par la relation:

$$\begin{bmatrix} X(n+1) \\ Y(n+1) \end{bmatrix} = \begin{bmatrix} X(n) \\ Y(n) \end{bmatrix} - \begin{bmatrix} J(\ X(n),Y(n)\ ) \end{bmatrix}^{-1} \begin{bmatrix} F(\ X(n),Y(n)\ ) \\ G(\ X(n),Y(n)\ ) \end{bmatrix}$$

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).
"[XØ, YØ]" est un **vecteur** représentant le point de départ des itérations.

3:	F(X,Y)
2:	G(X,Y)
1:	[XØ, YØ]

On appelle ensuite le programme ´SXY´. Au bout d´un certain temps (nécessaire au calcul des dérivées partielles) le point [X1, Y1] est mis sur la pile.
Le programme ´SXY´ est alors suspendu. On fait CONT pour voir apparaître [X2, Y2], puis le programme est suspendu, etc ...

A tout moment où le programme est suspendu, on peut avant de faire CONT, placer un nouveau point [XØ, YØ] au niveau 1 de la pile (à la place du point [Xn,Yn] qu´on vient d´obtenir), afin de lancer une nouvelle recherche.
De même, lors d´une suspension du programme (ou avant l´appel initial de ´SXY´), et avant de faire CONT, on peut placer au niveau 1 de la pile un entier p > 1 (Les autres objets de la pile remontant d´un niveau). Le programme ´SXY´ passe alors directement de [Xn, Yn] à [X(n+p), Y(n+p)].

Répertoire 'ANALY' programme 'SXY' (suite)

TEXTE DU PROGRAMME 'SXY' ET EXEMPLE D'UTILISATION.

'SXY':

```
«  IF  DUP  TYPE  THEN  1  END
   →  f  g  v  n
   «  { X Y }  PURGE
      f  'X'  ∂  f  'Y'  ∂  g  'X'  ∂  g  'Y'  ∂
      →  dfx  dfy  dgx  dgy
      «  DO
            1  n  START
               v  ARRY→  DROP  'Y'  STO  'X'  STO  v
               dfx  EVAL  dfy  EVAL  dgx  EVAL  dgy  EVAL
               {2 2}  →ARRY  INV  f  EVAL  g  EVAL  2  →ARRY
               *  -  'v'  STO
            NEXT
            v  HALT  IF  DEPTH  NOT  THEN  { X Y }  PURGE  ABORT  END
            IF  DUP  TYPE  THEN  1  END
            'n'  STO  'v'  STO
         UNTIL  Ø  END
      »
   »
»
```

Exemple:

On veut résoudre le système $X^2+Y^2=1$, $2X+Y=1$. Si on choisit le point [1,Ø] comme point de départ, on crée donc la pile:

3	'X*X+Y*Y-1'
2	'2*X+Y-1'
1	[1 Ø]

On appelle ensuite 'SXY'.
Au bout de 5 secondes, le point [1,-1] est placé sur la pile et 'SXY' est suspendu.
Chaque CONT amène un nouveau point (le calcul de chaque point durant 1 seconde). On trouve ainsi succesivement: (mode 6 FIX)
[Ø.833333 -Ø.666667],
[Ø.801282 -Ø.602564],
et [Ø.800002 -Ø.600004].

La suite (Xn,Yn) converge vers [Ø.8, -Ø.6], qui est effectivement solution du problème. Ici la convergence est rapide, mais elle peut l'être moins: il suffit alors de placer, lors d'une suspension de 'SXY', un entier p>1 au niveau 1 de la pile, pour passer directement de [X(n),Y(n)] à [X(n+p), Y(n+p)]. Sauf nouvelle modification le calcul suivant se fera à nouveau pas à pas (calcul de [X(n+p+1), Y(n+p+1)]).

Le mieux pour sortir du programme 'SXYZ' est de vider la pile (DROP ou CLEAR) avant de faire CONT. Ainsi les variables temporaires X,Y,Z sont elles purgées du répertoire.

Répertoire ´**ANALY**´ programme ´**SXYZ**´

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 : F(X,Y,Z)=Ø
G(X,Y,Z)=Ø
H(X,Y,Z)=Ø

Le principe est le suivant:

On définit une suite (Xn,Yn,Zn) par la donnée d´un point initial (XØ,YØ,ZØ) et par la relation:

$$\begin{bmatrix} X(n+1) \\ Y(n+1) \\ Z(n+1) \end{bmatrix} = \begin{bmatrix} X(n) \\ Y(n) \\ Z(n) \end{bmatrix} - \begin{bmatrix} J(\ X(n),Y(n),Z(n)\ ) \end{bmatrix}^{-1} \begin{bmatrix} F(X(n),Y(n),Z(n)) \\ G(X(n),Y(n),Z(n)) \\ H(X(n),Y(n),Z(n)) \end{bmatrix}$$

Où J est la matrice Jacobienne $J(X,Y,Z)) = \begin{bmatrix} F'_x\ (X,Y,Z) & F'_y\ (X,Y,Z) & F'_z\ (X,Y,Z) \\ G'_x\ (X,Y,Z) & G'_y\ (X,Y,Z) & G'_z\ (X,Y,Z) \\ H'_x\ (X,Y,Z) & H'_y\ (X,Y,Z) & H'_z\ (X,Y,Z) \end{bmatrix}$

Sous certaines conditions, la suite (Xn,Yn,Zn) converge vers une solution du système.

Au début, la pile doit avoir l´aspect suivant:
"F(X,Y,Z)", "G(X,Y,Z)", "H(X,Y,Z)" sont les expressions des applications F, G, H (dans ces expressions l´usage des majuscules X ,Y, et Z est obligatoire).
"[XØ, YØ, ZØ]" est un **vecteur** représentant le point de départ des itérations.

4:	F(X,Y,Z)
3:	G(X,Y,Z)
2:	H(X,Y,Z)
1:	[XØ, YØ, ZØ]

On appelle ensuite le programme ´SXYZ´. Au bout d´un certain temps (nécessaire au calcul des dérivées partielles) le point [X1, Y1, Z1] est mis sur la pile.
Le programme ´SXYZ´ est alors suspendu. On fait CONT pour voir apparaître [X2, Y2, Z2], puis le programme est suspendu, etc ...

A tout moment où le programme est suspendu, on peut avant de faire CONT, placer un nouveau point [XØ, YØ, ZØ] 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.
De même, lors d´une suspension du programme (ou avant l´appel initial de ´SXYZ´), et avant de faire CONT, on peut placer au niveau 1 de la pile un entier p > 1 (Les autres objets de la pile remontant d´un niveau). Le programme ´SXYZ´ passe alors de [Xn,Yn,Zn] à [X(n+p),Y(n+p),Z(n+p)].

Le mieux pour sortir du programme ´SXYZ´ est de vider la pile (DROP ou CLEAR) avant de faire CONT. Ainsi les variables temporaires X,Y,Z sont elles purgées du répertoire.

Répertoire 'ANALY' programme 'SXYZ' (suite)

TEXTE DU PROGRAMME 'SXYZ' ET EXEMPLE D'UTILISATION.

'SXYZ':

```
«  IF  DUP  TYPE  THEN  1  END
   →  f  g  h  v  n
   «  { X Y Z }  PURGE
      f  'X'  ∂  f  'Y'  ∂  f  'Z'  ∂
      g  'X'  ∂  g  'Y'  ∂  g  'Z'  ∂
      h  'X'  ∂  h  'Y'  ∂  h  'Z'  ∂
      →  dfx  dfy  dfz  dgx  dgy  dgz  dhx  dhy  dhz
      «  DO
            1  n  START
               v  ARRY→  DROP  'Z'  STO  'Y'  STO  'X'  STO  v
               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  *  -  'v'  STO
            NEXT
            v  HALT
            IF  DEPTH  NOT  THEN  { X Y Z }  PURGE  ABORT  END
            IF  DUP  TYPE  THEN  1  END
            'n'  STO  'v'  STO
         UNTIL  Ø  END
      »
   »
»
```

Exemple: soit à résoudre le système : $X^2+Y^2+Z^2=1$, $X+3X+2Z=5$, $X^3+YZ=-1$.

On crée tout d'abord la pile (si on veut partir du point [3 2 Ø]):

```
4:  'X*X+Y*Y+Z*Z-6'
3:  'X+3*Y+2*Z-5'
2:  'X^3+Y*Z+1'
1:  [ 3 2 Ø ]
```

On appelle 'SXYZ': au bout de 12 secondes, on a au niveau 1 : [2.Ø4Ø98 1.68852 -1.Ø5328] (en mode 5 FIX).
On obtient alors successivement:
[1.42661 1.84591 -.98218]
[1.11Ø88 1.96698 -1.ØØ591]
(le calcul d'un nouveau point dure 3 secondes).
Le programme étant suspendu sur ce dernier point, on met 1Ø sous ce nouveau point, avant de relancer par CONT.

On obtient alors en une quinzaine de secondes: [1 2 -1]. Le programme a ainsi calculé 1Ø nouveaux points avant d'afficher le dixième. La suite (Xn,Yn,Zn) converge vers le point (1,2,-1), qui est effectivement solution du prblème.

Il suffit alors de faire CLEAR puis CONT pour sortir du programme.

Répertoire ´**ANALY**´ programme ´**RK4**´

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(xØ)=yØ, où (xØ,yØ) est un point donné.
(une telle solution existe de manière unique, sous certaines hypothèses de régularité de F).

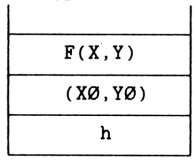
On définit la suite Xn = XØ + n*h, où h est le "pas" de la méthode,
et la suite Yn, définie par la donnée du terme initial YØ, et par le système:

a = h * F(X(n-1),Y(n-1)).
b = h * F(X(n-1) + h/2 , Y(n-1) + a/2).
c = h * F(X(n-1) + h/2 , Y(n-1) + b/2).
d = h * F(X(n-1) + h , Y(n-1) + c).

et Y(n) = Y(n-1) + (a + 2*b + 2*c + d) / 6.

Les Y(n) sont alors des valeurs approchées de F(X(n))=F(XØ+n*h).

Avant d´appeler ´RK4´, la pile doit avoir l´aspect suivant:



où:
"F(X,Y)"= expresssion de F (majuscules X et Y obligatoires).
"(XØ,YØ)"= point de départ (sous forme d´un nombre complexe).
"h"= pas de la méthode (nombre réel).

On lance ´RK4´. Au bout d´un certain temps (nécessaire au calcul des dérivées partielles) le point (X1,Y1) est mis sur la pile. Le programme ´RK4´ est alors suspendu. Un appui sur CONT provoque le calcul du nouveau point (X2,Y2) et ainsi de suite.... Les différents points restent sur la pile, le nouveau venant se placer au niveau 1, remontant les autres d´un niveau.
A chaque fois que le programme est suspendu, on peut modifier le pas "h". On fait par exemple Ø.1 ´h´ STO (minuscule obligatoire).
De même, pendant une suspension du programme, (et aussi avant l´appel initial de ´RK4´), on peut placer au niveau 1 de la pile un entier p>1. Quand on fait ensuite CONT, le programme ´RK4´ calcule p nouveaux points, au lieu d´un seul, et les place sur la pile (avant d´être à nouveau suspendu).
On sort de ´RK4´ par l´instruction KILL du menu CONTRL.

Répertoire 'ANALY' programme 'RK4' (suite)

TEXTE DU PROGRAMME 'RK4' ET EXEMPLE D'UTILISATION.

```
'RK4':
    «   IF  2  PICK  TYPE  THEN  1  END
        →  f  v  h  n
        «  v  C→R  'Y'  STO  'X'  STO  Ø  Ø  Ø  Ø
           →  k1  k2  k3  k4
           «  v
              DO
                 1  n  START
                    f  EVAL  h  *  'k1'  STO  X  h  2  /  +
                    'X'  STO  'IM(v)+k1/2'  EVAL  'Y'  STO  f
                    EVAL  h  *  'k2'  STO  'IM(v)+k2/2'  EVAL
                    'Y'  STO  f  EVAL  h  *  'k3'  STO  X  h
                    2  /  +  'X'  STO  'IM(v)+k3'  EVAL  'Y'
                    STO  f  EVAL  h  *  'k4'  STO  X
                    'IM(v)+(k1+2*k2+2*k3+k4)/6'  EVAL  R→C  DUP
                    'v'  STO
                 NEXT
                 HALT
                 IF  DUP  TYPE  THEN  1  END
                 'n'  STO  DUP  'v'  STO
              UNTIL  Ø  END
        »
    »
»
```

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)=TAN(X^2).

Pour aller plus vite on veut connaître directement les valeurs de f(Xn) pour Xn= n*Ø.Ø5 et Ø ≤ n ≤ 1Ø. On crée donc la pile suivante:

```
4: '2*X*(1+Y*Y)'
3:   ( Ø, Ø )
2:     Ø.Ø5
1:      1Ø
```

Le résultat (11 points différents) est obtenu en 6 secondes.

On trouve par exemple:

f(.25) ≈ 6.25815Ø1685E-2, la valeur exacte étant:

tan(.25^2) ≈ 6.25815Ø75663E-2.

de même, on trouve:

f(.5) ≈ .255342Ø32756, la valeur exacte étant:

tan(.5^2) ≈ .255341921221.

Remarque: Le programme 'TRACE' est utile pour visualiser les résultats obtenus par 'RK4'.

Répertoire 'ANALY' programme 'FOUR'

SOMMES PARTIELLES D'UNE SERIE DE FOURIER.

Soit f une fonction périodique de période T > Ø. 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:

$$\sum_{k=Ø}^{+\infty} (ak*\cos(k*w*x) + bk*\sin(k*w*x)) \quad \text{où } w = \frac{2*\pi}{T} .$$

avec, pour tout k ≥ Ø,

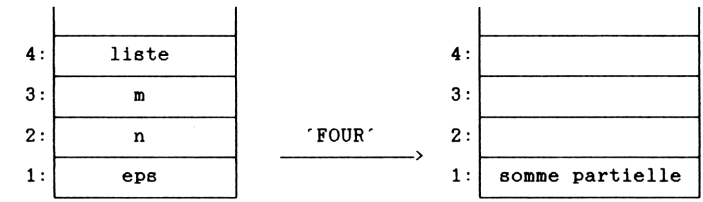
$$ak = \frac{2}{T}\int_a^b f(x)*\cos(k*w*x)dx \quad \text{et} \quad bk = \frac{2}{T}\int_a^b f(x)*\sin(k*w*x)dx$$

(Exception: $aØ = \frac{1}{T}\int_a^b f(x)dx$)

'FOUR' permet de calculer toute somme partielle de cette série, c'est-à-dire toute expression:

$$\sum_{k=m}^{n} (ak*\cos(k*w*x) + bk*\sin(k*w*x)) \quad .$$

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 ak et bk.

eps = erreur relative permise dans les calculs d'intégrales.

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:

$$W = \frac{2\pi}{T} \quad (W="pulsation").$$

Répertoire 'ANALY' programme 'FOUR' (suite)

TEXTE DU PROGRAMME 'FOUR'.

'FOUR':

```
« → m n eps
  « LIST→ DROP DUP2 - NEG
    → f a b t
    « RAD 35 CF 2 π * t / 'W' STO
      « 'X' a b 3 →LIST eps ∫ 2 PICK ABS
        IF > THEN DROP Ø ELSE t / END
      »
      → int
      « Ø m n FOR k
          IF k Ø == THEN f int EVAL +
          ELSE
            k 'W' * 'X' *
            → h
            « h EVAL COS f * int EVAL 2 *
              h COS * + h EVAL SIN f *
              int EVAL 2 * h SIN * +
            »
          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.

Celà se produit en particulier pour les fonctions paires (les "bk" sont nuls) ou impaires (las "ak" sont nuls).

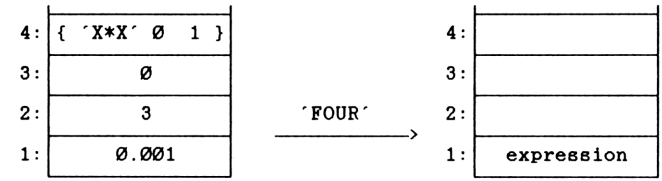
Dans de tels cas, l'expression de la somme partielle de la série de Fourier est débarassée de ses termes négligeables.

Répertoire 'ANALY' programme 'FOUR' (suite)

TEXTE DU 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 [Ø, 1]. On veut par exemple calculer les coefficients ak et bk pour $Ø \leq k \leq 3$. Les calculs d'intégrales seront faits avec une erreur relative de .ØØ1. On obtient en 3Ø secondes:



Avec (en MODE 3 FIX):
expression = 'Ø.333+Ø.1Ø1*COS(W*X)-Ø.318*SIN(W*X)+Ø.Ø25*COS(2*W*X) -Ø.159*SIN(2*W*X)+Ø.Ø11*COS(3*W*X)-Ø.1Ø6*SIN(3*W*X) '

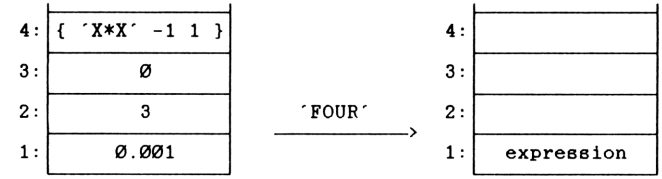
Un appui sur EVAL permet alors de remplacer 'W' par sa valeur, c'est-à-dire $2*\pi/T = 2*\pi \approx 6.283$.

Sur cet exemple, on peut tout aussi bien s'interesser uniquement aux coefficients a5 et b5. il suffit de placer 5 et 5 aux niveaux 3 et 2.

On obtient alors, en 19 secondes, au niveau 1 de la pile, l'expression: 'Ø.ØØ4*COS(5*W*X)-Ø.Ø64*SIN(5*W*X).

Exemple2:

On s'interesse à la même fonction X ⟶ X^2, mais on veut la développer sur [Ø,1] en série de cosinus. Il suffit de la considérer comme paire et définie sur [-1,1] (donc périodique de période 2). En 3Ø secondes:



Avec (en MODE 3 FIX):
expression = 'Ø.333-Ø.4Ø5*COS(W*X)+Ø.1Ø1*COS(2*W*X)-Ø.Ø45**COS(3*W*X)'

Un appui sur EVAL permet alors de remplacer 'W' par sa valeur, c'est-à-dire $2*\pi/T = \pi \approx 3.142$.

Exemple3:

On peut de même développer X^2 sur [Ø,1] en série de sinus. Il suffit de prolonger X^2 sur [-1,Ø] de façon impaire et de considérer que la période est 2. On place { 'X*X*SIGN(X)' -1 1 } au niveau 4 (les autres niveaux restant les mêmes que dans l'exemple 2). On obtient
(en mode 3 FIX):
' Ø.379*SIN(W*X) - Ø.318*SIN(2*W*X) + Ø.2Ø3**SIN(3*W*X) '
Une pression sur EVAL permet alors de remplacer W par sa valeur,
c'est-à-dire: $2\pi/T = \pi$.

DEVELOPPEMENTS LIMITES

La HP28 permet de calculer (sous réserve de dérivabilité suffisante) le développement limité à un ordre quelconque n, en Ø, d'une fonction f.
Ce développement est obtenu par la formule de Taylor:

$$f(x) = f(Ø) + \frac{f'(Ø)}{1!}x + \frac{f''(Ø)}{2!}x^2 + \ldots + \frac{f^{(n)}(Ø)}{n!}x^n + o(x^n).$$

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 HP28 calcule les dérivées successives de f, sous forme symbolique, avant de les évaluer en Ø. 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 3Øs 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 21 secondes.

On peut multiplier les exemples, de manière plus convaincante encore. Le calcul du développement de ATAN(ASIN(X)) (arctangente de arcsinus de x), en Ø 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 45ØØ octets de mémoire libre). Sur la même machine, le même développement est obtenu en 22 secondes avec les programmes du répertoire 'DL'.... et on trouve:

'X-.166666666666*X^3+.1Ø8333333333*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 $f(x)= a + bx + cx^2 + \ldots + d\,x^n + o(x^n)$ 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 spécifiques 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 appelé par pratiquement tous les programmes du répertoire 'DL'.
'**ΣDL**' : somme de deux développements limités.
'**πDL**' : produit de deux développements limités.
'**CPDL**' : composé de deux développements limités.
'**INVDL**': inverse d'un développement limité.
'**QDL**' : quotient de deux développements limités.
'**DLN**' : élévation à une 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 X^n 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**';
'**ASN**', '**ACS**', '**ATG**', '**ASH**', '**ACH**', '**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 —> A^X, X —> LN(1+X), X —> (1+X)^A;
X —> SIN(X), X —> COS(X), X —> TAN(X);
X —> SINH(X), X —> COSH(X), X —> TANH;
X —> ASIN(X), X —> ACOS(X), X —> ATAN(X);
X —> ASINH(X), X —> ACOS(X)H, 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' interprèteront cette longueur comme donnant l'ordre du développement limité . Par exemple, si vous mettez [1 3 -4] sur la pile, celà signifie que vous y mettez le développement limité 1 + 3*X -4*X^2 + o(X^2). Si vous voulez calculer le développement limité de SIN(1 + 3*X -4*X^2) à l'ordre 6, il vous faudra mettre [1 3 -4 0 0 0 0] sur la pile (qui représente le développement limité 1 + 3*X -4*X^2 + o(X^6)).

Réciproquement, lorque 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).

Répertoire 'DL' Programme 'DIM'

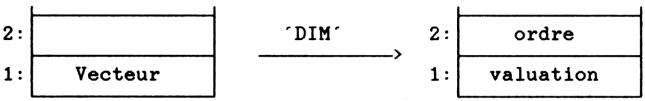
ORDRE ET VALUATION D'UN DEVELOPPEMENT LIMITE.

Si f(x) = A0 + A1*X + A2*X² + ... + An*Xⁿ + o(Xⁿ) est un développement limité, son ordre est n et sa valuation est p (si p est le plus petit indice tel que Ap ≠ 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..

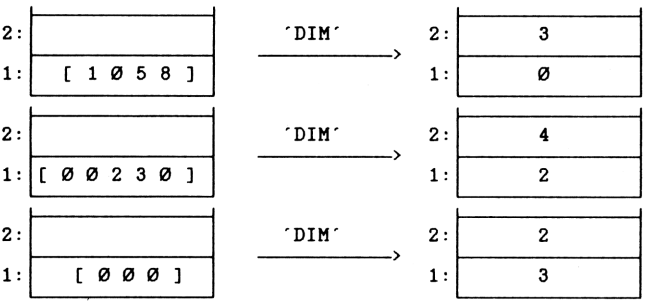
La valuation p est un entier tel que 0 ≤ p ≤ n. Exception: dans le cas où le vecteur au niveau 1 ne contient que des coefficients nuls, on obtient p=n+1.

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':

```
« DUP SIZE 1 GET → a n
  « n 1 -
    IF a RNRM THEN 1 → i
    « WHILE 'a(i)' EVAL NOT REPEAT i 1 + 'i' STO END
      i 1 -
    »
    ELSE n END
  »
»
```

Exemples:



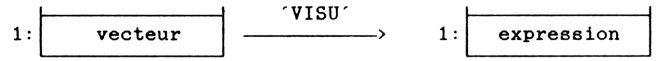
Répertoire 'DL' Programme 'VISU'

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. celà permet de distinguer les développements limités obtenus à partir, par exemple de [1 -2 5] et de [1 -2 5 0 0], qui sans celà donneraient la même expression.

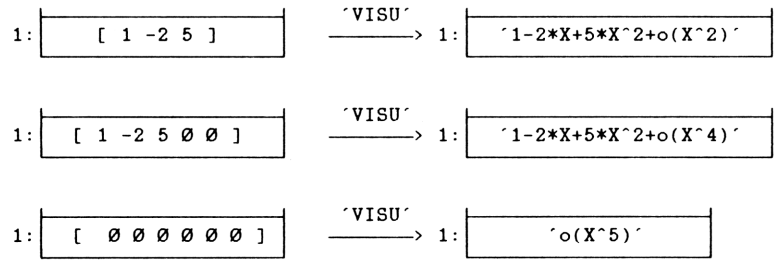
Le schéma fonctionnel est le suivant:



'VISU':

```
«   DUP   SIZE   1   GET
    →  p   n
    «  0   1   n   FOR   i
           'p'   i   GET   'X'   i   1   -   ^   *   +
       NEXT
       'o(X^Y)'   4   n   1   -   1   →LIST   OBSUB   +
    »
»
```

Exemples: (les temps de calculs sont légèrement supérieurs à 1 seconde)



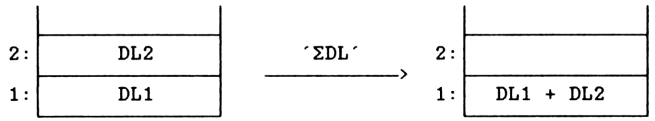
Répertoire 'DL' Programme 'ΣDL'

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).

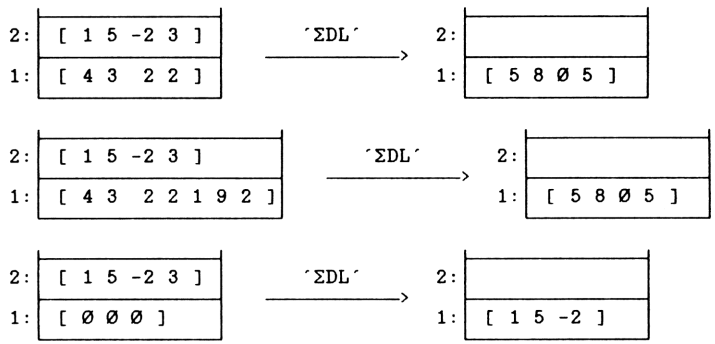
Le schéma fonctionnel est:



'ΣDL':

```
«   DUP2   SIZE   1   GET   SWAP   SIZE
    1   GET   DUP2   MIN   1   →LIST
    →   dim
    «   IF   <   THEN   SWAP   END
        dim   RDM   +
    »
»
```

Exemples: (en moins d'une seconde)



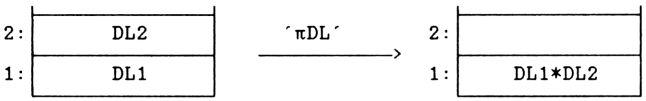
Répertoire 'DL' Programme 'πDL'

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:



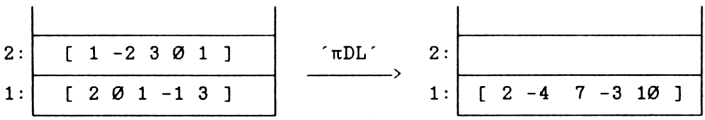
N.B: le programme 'πDL' appelle 'DIM' et le programme 'PRODP' du répertoire 'POLY'.

'πDL':

```
«  →  f  g
   «  f  DIM  g  DIM  4  ROLL  +  3  ROLLD  +  MIN
      1  +  1  →LIST
      →  dim
      «  f  g  POLY  PRODP  DL  dim  RDM
   »
»
```

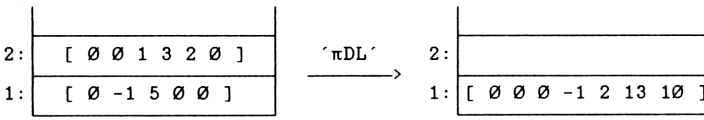
Exemples:

* en 4 secondes:



Car (1 -2*X +3*X^2 +X^4 +o(X^4)) * (2 +X^2 -X^3 +3*X^4 +o(X^4))
= 2 -4*X +7*X^2 -3*X^3 +1ØX^4 +o(X^4).

* en 5 secondes:



Car (X^2 +3*X^3 +2*X^4 +o(X^5)) * (-X +5*X^2 +o(X^4))
= -X^3 +2*X^4 +13*X^5 +1Ø*X^6 +o(X^6)

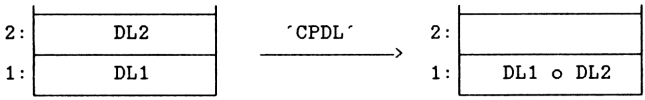
Répertoire 'DL' Programme 'CPDL'

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, 'πDL' donne le résultat à l'ordre le plus élevé possible (tous les coefficients obtenus sont exacts).

Le schéma fonctionnel est:

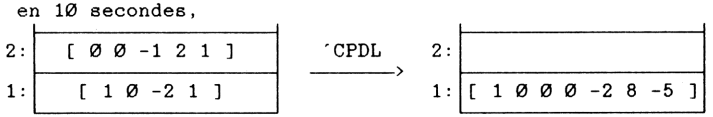


N.B: le programme 'πDL' appelle 'DIM' et le programme 'PRODP' du répertoire 'POLY'.

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':
  « → f g
    « IF 'f(1)' EVAL THEN f g "Erreur"
      ELSE
        'g(1)' EVAL 1 →ARRY g { 1 } Ø PUT
        'g' STO f DIM g DIM
        → sf vf sg vg
        « 'vf*(vg-1)+gf+1' EVAL 'vf*(sg+1)' EVAL
          MIN 1 →LIST [ 1 ]
          → dim r
          « dim RDM
            IF sg THEN
              POLY 2 sg 1 + FOR i
                f r PRODP dim RDM DUP
                'r' STO g i GET * +
              NEXT
              DL
            END
          »
        »
      END
    »
  »
```

Exemple: en 1Ø secondes,



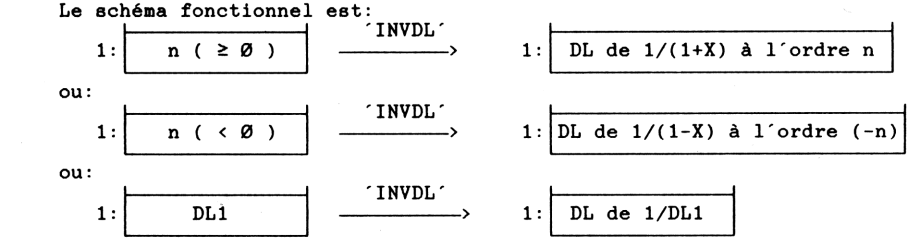
Car si f(X) = -X^2 +2*X^3 +X^4 +o(X^4)
et si g(X) = 1 -2*X^2 +X^3 +o(X^3), alors:
gof(X) = 1 -2*X^4 +8*X^5 -5*X^6 +o(X^6).

Répertoire 'DL' Programme 'INVDL'

INVERSE D'UN DEVELOPPEMENT LIMITE ET DEVELOPPEMENT DE 1/(1+X).

'INVDL' 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).

Le schéma fonctionnel est:

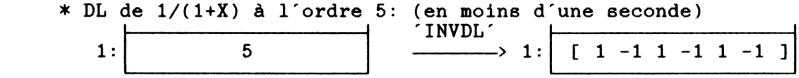


'INVDL':

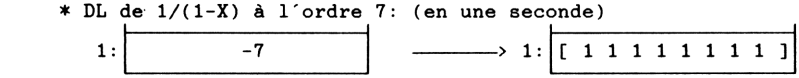
```
«
    « → s  n
        «  Ø  n  FOR  i  's^i'  EVAL  NEXT  n  1  +  →ARRY
        »
    »  →  d
    «  DUP
       IF  DUP  TYPE  THEN
          1  GET  DUP  INV  3  ROLLD  /  { 1 }  Ø  PUT  DUP
          DIM  /  FLOOR  -1  SWAP  d  EVAL  CPDL  *
       ELSE
          SIGN  NEG  SWAP  ABS  d  EVAL
       END
    »
»
```

Exemple:

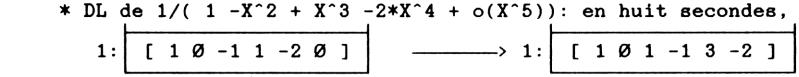
* DL de 1/(1+X) à l'ordre 5: (en moins d'une seconde)



* DL de 1/(1-X) à l'ordre 7: (en une seconde)



* DL de 1/(1 -X^2 + X^3 -2*X^4 + o(X^5)): en huit secondes,



en effet ici:

1/(1 -X^2 + X^3 -2*X^4 + o(X^5)) = 1 +X^2 -X^3 +3*X^4 -2*X^5 +o(X^5)

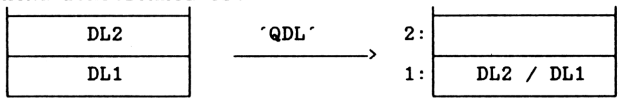
Répertoire 'DL' Programme 'QDL'

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:



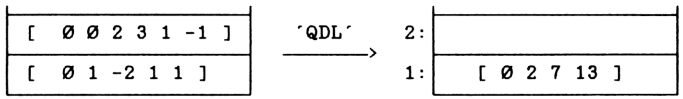
N.B: le programme 'QDL' appelle 'DIM' et le programme 'DIVPC' du répertoire 'POLY'.

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':

```
«   DUP2  DIM  ROT  DIM  →  m  q  n  p
    «  IF  p  q  <  THEN  "Erreur"  ABORT  END
       n  m  MIN  q  -  1  +  1  →LIST
       «  ARRY→  1  GET  →  s
          «  s  s  q  -  1  +  FOR  i  i  ROLL  DROP  -1  STEP
             s  q  -  →ARRY
          »
       »  →  f  g  r  t
       «  f  IF  q  THEN  t  EVAL  END  r  RDM
          g  IF  q  THEN  t  EVAL  END  r  RDM
          r  1  GET
          IF  DUP  1  ==  THEN  DROP  1  GET  /
          ELSE  POLY  DIVPC  DL  DROP  r  RDM  END
       »
    »
»
```

Exemple: en six secondes:



En effet,

si f(X) = 2*X^2 + 3*X^3 +X^4 -X^5 +o(X^5)

et si g(X) = X -2*X^2 +X^3 +X^4 +o(X^4), alors:

$$\frac{f(X)}{g(X)} = 2X +7X^2 +13X^3 + o(X^3).$$

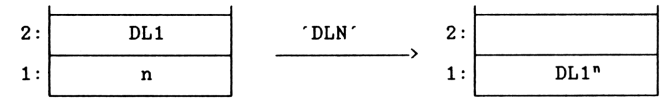
Répertoire 'DL' Programme 'DLN'

ELEVATION D'UN DEVELOPPEMENT LIMITE A UNE PUISSANCE ENTIERE.

'DLN' permet d'éléver un développement limté DL1 (représenté sous forme d'un vecteur) à une puissance entière n (n entier positif ou nul). Le résultat est un vecteur représentant $DL1^n$.

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:

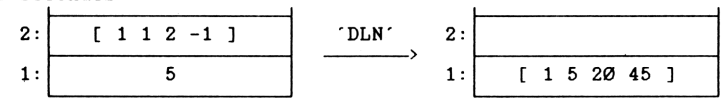


N.B: 'DLN' appelle 'DIM' et le programme 'PRODP' du répertoire 'POLY'.

```
'DLN':
    «   SWAP   DUP   DIM
        →  n   f   m   p
        «  'p*(n-1)+m+1'   EVAL   1   →LIST
           →  dim
           «  POLY   [ 1 ]   dim   RDM
              WHILE   n   Ø   >   REPEAT
                 IF   n   2   MOD   THEN   f   PRODP   dim   RDM   END
                 n   2   /   FLOOR   'n'   STO
                 IF   n   Ø   >   THEN
                    f   DUP   PRODP   dim   RDM   'f'   STO
                 END
              END
              DL
           »
        »
    »
```

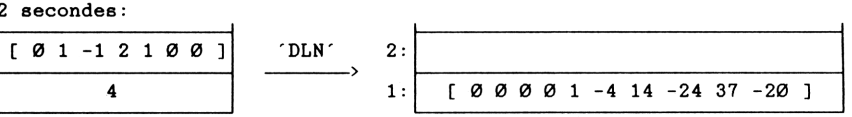
Exemples:

* en 1Ø secondes:



car (1 +X +2*X^2 -X^3 +o(X^3))^5 = 1 +5*X +2Ø*X^2 +45*X^3 +o(X^3).

* en 22 secondes:

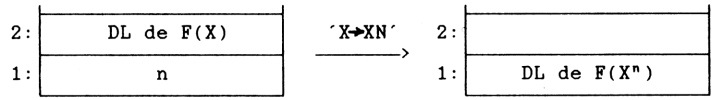


car (X -X^2+2*X^3+X^4+o(X^6))^4
= X^4 -4*X^5 +14*X^6 -24*X^7 +37*X^8 -2Ø*X^9 +o(X^9).

TRANSFORMATION X —> X^N DANS UN DEVELOPPEMENT LIMITE.

'X→XN' permet de transformer le développement limité de F(X) en le développement limité de F(X^n), où n est un entier naturel.

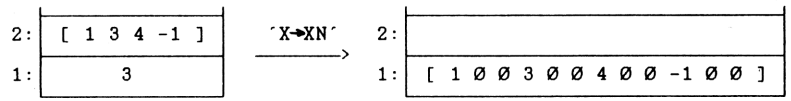
Le schéma fonctionnel est le suivant:



'X→XN':

```
«  →  n
   «  ARRY→  { 1 }  SWAP  +
      →  dim
      «  dim  →ARRY  dim  1  n  PUT  RDM  TRN
         ARRY→  LIST→  DROP  *  →ARRY
      »
   »
»
```

Exemple: (en une seconde)

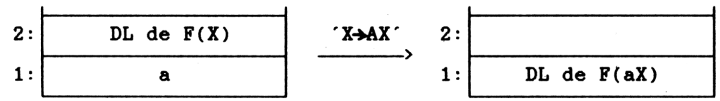


Car si F(X)= 1 +3*X +4*X^2 -X^3 +o(X^3), alors
F(X^3)= 1 +3*X^3 +4*X^6 -X^9 +o(X^11).

TRANSFORMATION X —> A*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.

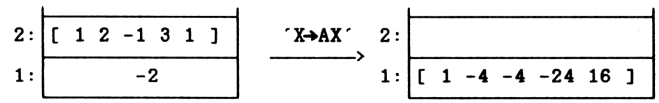
Le schéma fonctionnel est le suivant:



'X→AX':

```
«   →   d   a
     «  d   { 1 }
        1   d   SIZE   1   GET   FOR   i
            'd(i)*a^(i-1)'   EVAL   PUTI
        NEXT
        DROP
      »
»
```

Exemple: (en une à deux secondes)



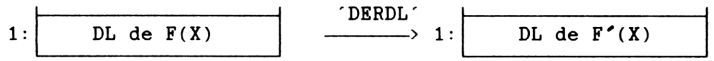
car si F(X) = 1 +2*X -X^2 +3*X^3 +X^4 +o(X^4),
F(-2X)= 1 -4*X -4*X^2 -24*X^3 +16*X^4 +o(X^4).

Répertoire 'DL' Programmes 'DERDL'

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 dérivablité de F sont nécessaires).

Le schéma fonctionnel est:

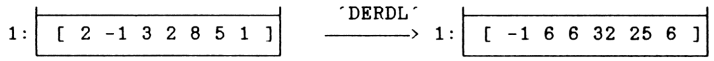


L'ordre du développement limité obtenu est évidemment inférieur d'une unité à celui du développement initial.

'DERDL':

```
«   DUP   SIZE   1   GET
    →  d   n
    «  IF   n   1   ==   THEN
          [ Ø ]
       ELSE
          2   n   FOR   i
             'd(i)*(i-1)'   EVAL
          NEXT
          n   1   -   →ARRY
       END
    »
»
```

Exemple: (en une seconde)



Le développement limité :

2 -X +3*X^2 +2*X^3 +8*X^4 +5*X^5 +X^6 +o(X^6)

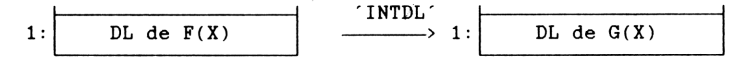
est donc transformé en:

-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 Ø, par intégration terme à terme de chaque coefficient.

Le schéma fonctionnel est:

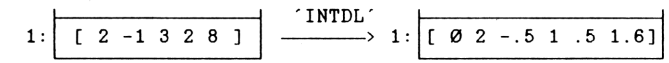


L'ordre du développement limité obtenu est évidemment supérieur d'une unité à celui du développement initial. Son terme constant (c'est à dire le premier coefficient du vecteur obtenu) est nul.

'INTDL':

```
«   DUP   SIZE   1   GET
    →  d   n
    «  Ø   1   n   FOR   i
           'd(i)/i'   EVAL
       NEXT
       n   1   +   →ARRY

    »
»
```

Exemple: (en une seconde)



Le développement limité :

2 -X +3*X^2 +2*X^3 +8*X^4 +o(X^4)

est donc transformé en:

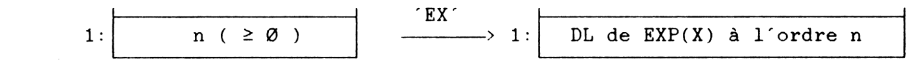
2*X -.5*X^2 +X^3 +.5*X^4 +1.6*X^5 +o(X^5).

Répertoire 'DL' programme 'EX'

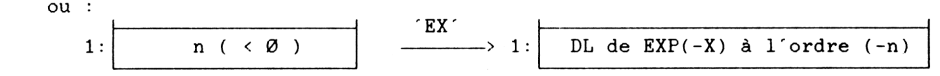
DEVELOPPEMENT LIMITE DE EXP(X) ET EXPONENTIELLE D'UN D.L.

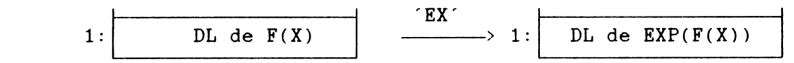
'EX' permet tout à la fois de calculer le développement limité de EXP(X) ou de EXP(-X) à un ordre n quelconque, ou celui de EXP(F(X)) ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



ou :





Dans ce dernier cas le D.L. de EXP(F(X)) est obtenu au même ordre que celui de F(X).

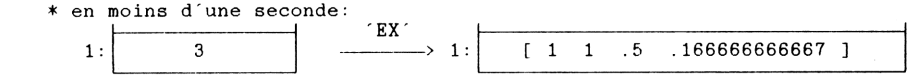
N.B: 'EX' appelle les programmes 'DIM' et 'CPDL'

'EX':

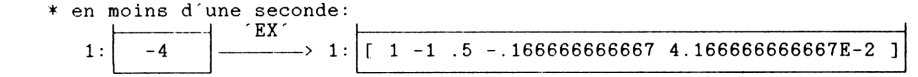
```
«    «  →   ε   n
        «   Ø   n   FOR   i    'ε^i/FACT(i)'   EVAL   NEXT
            n   1   +   →ARRY
        »
     » → d
     «  DUP   IF   DUP   TYPE   THEN
           1   GET   EXP   SWAP   { 1 }   Ø   PUT   DUP
           DIM   /   FLOOR   1   SWAP   d   EVAL   CPDL   *
        ELSE
           SIGN   SWAP   ABS   d   EVAL
        END
     »
»
```

Exemples:

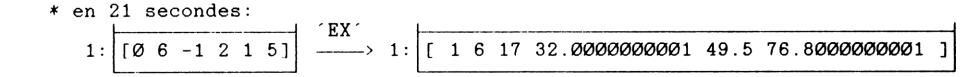
* en moins d'une seconde:



* en moins d'une seconde:



* en 21 secondes:

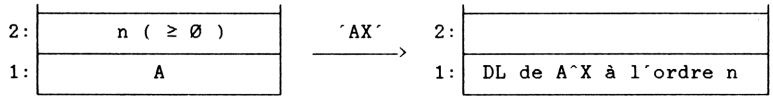


Répertoire 'DL' programme **'AX'**

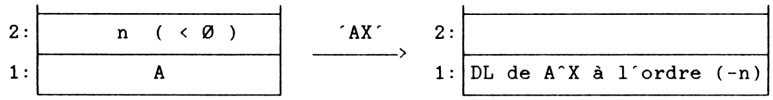
DEVELOPPEMENT LIMITE DE A^X ET D.L. DE A^F(X).

'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 > Ø.

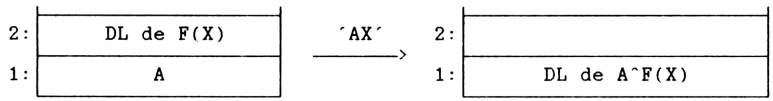
Le schéma fonctionnel est:



ou:



ou:



Dans ce dernier cas le D.L. de A^(F(X)) est obtenu au même ordre que celui de F(X).

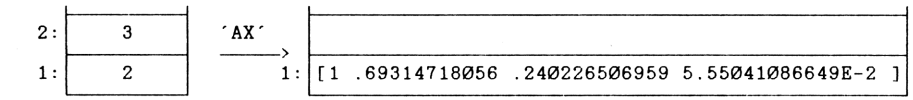
N.B: 'AX' appelle les programmes 'EX' et 'X→AX'.

'AX':

```
«   →  a
    «  IF   DUP   TYPE   THEN   a   LN   *   EX
       ELSE   EX   a   LN   X→AX   END
    »
»
```

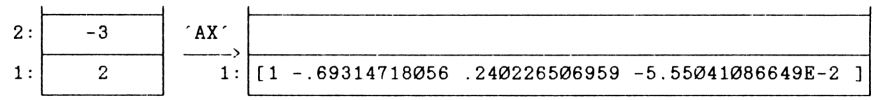
Exemples:

* en une à deux secondes:



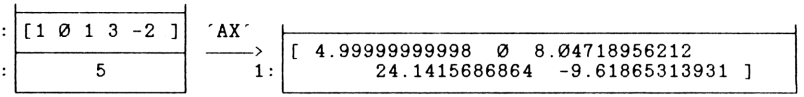
Car:
2^X = 1 +.69314718056*X .240226506959*X^2 +5.55041086649E-2*X^3 +o(X^3).

* en une à deux secondes:



(on obtient ainsi le D.L. de 2^(-X) à l'ordre 3).

* en 7 secondes:



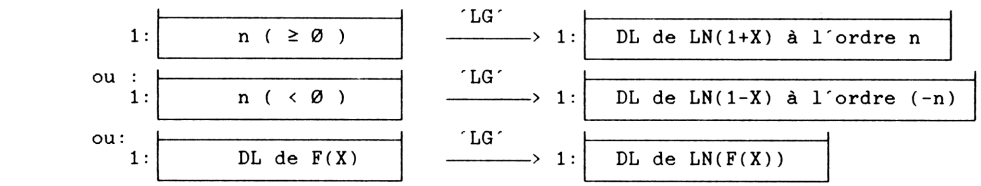
On obtient ainsi le D.L. de 5^(1 +X^2 +3*X^3 -2*X^4 +o(X^4)) à l'ordre 4.

Répertoire 'DL' programme 'LG'

DEVELOPPEMENT LIMITE DE LN(X) ET LOG. NEPERIEN D'UN D.L.

'LG' permet tout à la fois de calculer le développement limité de LN(1+X) ou de LN(1-X) à un ordre n quelconque, ou celui de 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 LN(F(X)) est obtenu au même ordre que celui de F(X). De plus on doit avoir F(Ø) > Ø.

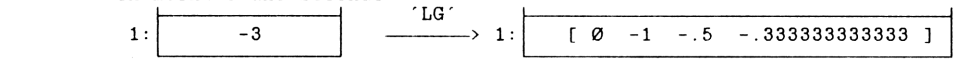
N.B: 'LG' appelle les programmes 'DIM' et 'CPDL'

'LG':

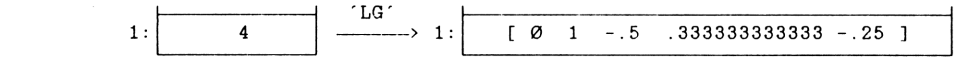
```
«    « → s n
       « Ø IF n THEN 1 n FOR i 'sˆi/i' EVAL NEXT END
         n 1 + →ARRY NEG
       »
     » → d
       « DUP IF DUP TYPE THEN
           1 GET DUP LN 3 ROLLD / { 1 } Ø PUT DUP
           DIM / FLOOR -1 SWAP d EVAL CPDL { 1 } ROT PUT
         ELSE
           SIGN NEG SWAP ABS d EVAL
         END
       »
»
```

Exemples:

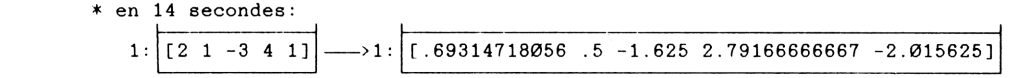
* en moins d'une seconde:



* en moins d'une seconde:



* en 14 secondes:



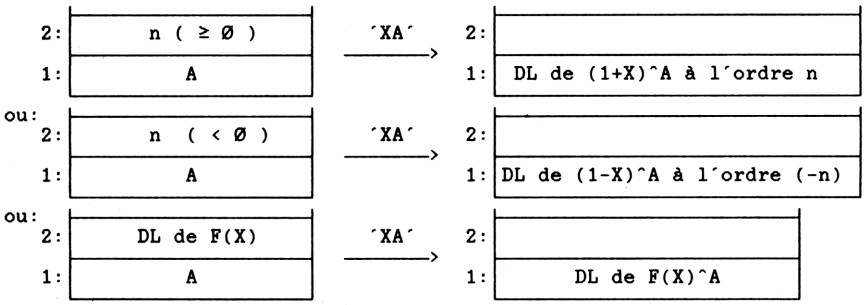
Car LN(2 +X -3*Xˆ2 +4*Xˆ3 +Xˆ4 +o(Xˆ4))=
.69314718Ø56 +.5*X -1.625*Xˆ2 +2.79166666667*Xˆ3 -2.Ø15625*Xˆ4 +o(Xˆ4).

Répertoire 'DL' programme 'XA'

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(Ø) doit être non nul.

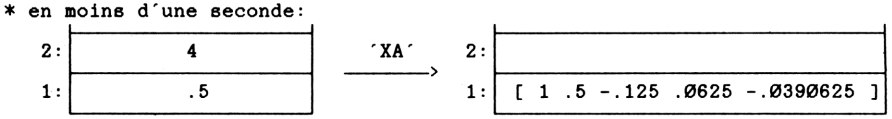
N.B: 'XA' appelle les programmes 'DIM' et 'CPDL'.

'XA':

```
«  →  a
   «
      «  →  s  n
         «  1
            IF  n  THEN  1  n  FOR  i  DUP  's*(a-i+1)/i'
               EVAL  *  NEXT  END
            n  1  +  →ARRY
         »
      »  →  d
      «  DUP
         IF  DUP  TYPE  THEN
            1  GET  DUP  a  ^  3  ROLLD  /  { 1 }  Ø  PUT
            DUP  DIM  /  FLOOR  1  SWAP  d  EVAL  CPDL  *
         ELSE
            SIGN  SWAP  ABS  d  EVAL
         END
      »
   »
»
```

Exemple:

* en moins d'une seconde:



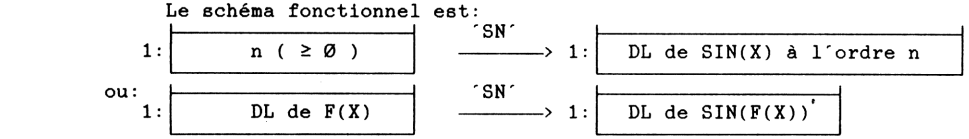
car (1+X)^(1/2) = 1 +(1/2)*X -(1/8)*X^2 +(1/16)*X^3 -(5/128)*X^4 +o(X^4)

Répertoire 'DL' programme 'SN'

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 ≥ Ø), ou celui de SIN(F(X)) ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



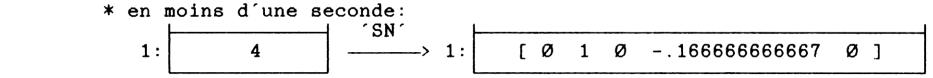
Dans ce dernier cas le D.L. de SIN(F(X)) est obtenu au même ordre que celui de F(X).

N.B: 'SN' appelle les programmes 'DIM', 'CPDL', 'CS', 'SN', 'ΣDL'.

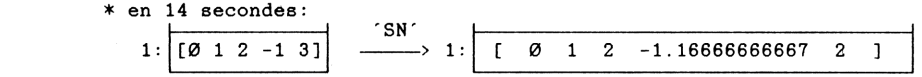
```
'SN':
    «   «  →  n
            «  Ø   n   FOR  i
                  Ø   i   1   +   FACT   INV
                  IF   i   4   MOD   THEN   NEG   END
               2   STEP
               IF   n   2   MOD   NOT   THEN   DROP   END
               n   1   +   →ARRY
            »
        »
        →  d
        «  IF   DUP   TYPE   THEN
              { 1 }   GETI   SWAP   DROP
           →  fØ
           «  IF   fØ   THEN
                 { 1 }   Ø   PUT   DUP   SN   fØ   COS   *
                 SWAP   CS   fØ   SIN   *   ΣDL
              ELSE
                 DUP   DIM   /   FLOOR   d   EVAL   CPDL
              END
           »
        ELSE
           d   EVAL
        END
    »
»
```

Exemples:

* en moins d'une seconde:



* en 14 secondes:



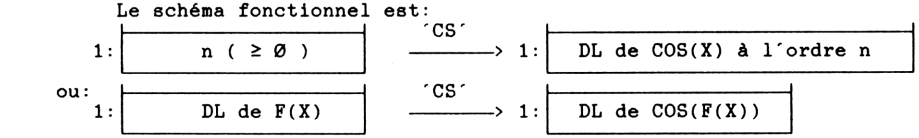
Car SIN(X +2*X^2 -X^3 +3*X^4 +o(X^4))
= X +2*X^2 -1.16666666667*X^3 +2*X^4 +o(X^4).

Répertoire 'DL' programme 'CS'

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 ≥ Ø), 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).

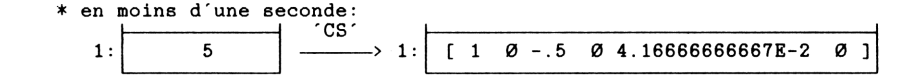
N.B: 'CS' appelle les programmes 'DIM', 'CPDL', 'SN', 'CS', et 'ΣDL'.

'CS':

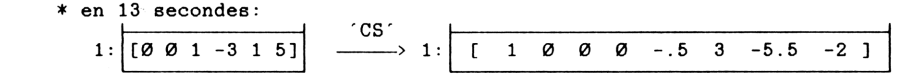
```
«  «  →  n
      «  Ø   n   FOR  i
            i   FACT   INV
            IF   i   4   MOD   THEN   NEG   END
            Ø
         2   STEP
         IF   n   2   MOD   NOT   THEN   DROP   END
         n   1   +   →ARRY
      »
   »
   →  d
   «  IF   DUP   TYPE   THEN
         { 1 }   GETI   SWAP   DROP
         →  fØ
         «  IF   fØ   THEN
               { 1 }   Ø   PUT   DUP   CS   fØ   COS   *
               SWAP   SN   fØ   SIN   *   NEG   ΣDL
            ELSE
               DUP   DIM   /   FLOOR   1   +   d   EVAL   CPDL
            END
         »
      ELSE
         d   EVAL
      END
   »
»
```

Exemples:

* en moins d'une seconde:



* en 13 secondes:



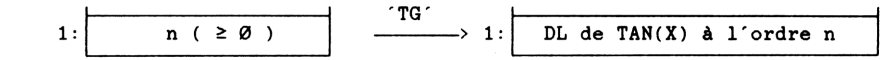

```
Car COS( X^2 -3*X^3 +X^4 +5*X^5 +o(X^5) )
     =   1 -.5*X^4 +3*X^5 -5.5*X^6 -2*X^7 +o(X^7).
```

Répertoire 'DL' programme 'TG'

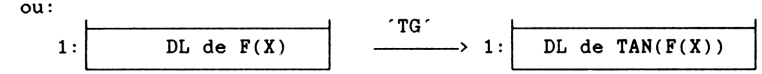
DEVELOPPEMENT LIMITE DE TAN(X) ET TANGENTE D'UN D.L.

'TG' permet tout à la fois de calculer le développement limité de TAN(X) à un ordre n quelconque (n ≥ Ø), ou celui de TAN(F(X)) ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



ou:



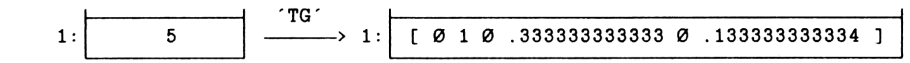
Dans ce dernier cas le D.L. de TAN(F(X)) est obtenu au même ordre que celui de F(X).
N.B: 'TG' appelle les programmes 'CS', 'SN', 'QDL'.

'TG':

```
« DUP SN SWAP CS QDL »
```

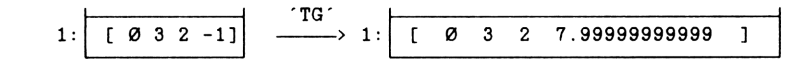
Exemples:

* en 8 secondes:



car TAN(X) = X + (1/3)*X^3 +(2/15)*X^5 + o(X^5).

* en 24 secondes:



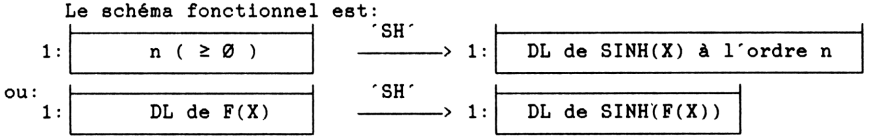
car TAN(3*X +2*X^2 -X^3 +o(X^3)) = 3*X +2*X^2 +8*X^3 +o(X^3) .

Répertoire 'DL' programme 'SH'

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 ≥ Ø), 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).

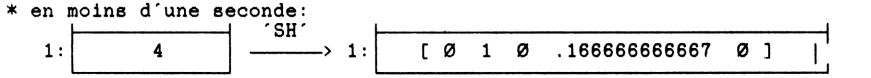
N.B: 'SH' appelle les programmes 'DIM', 'CPDL', 'CH', 'SH', 'ΣDL'.

'SH':

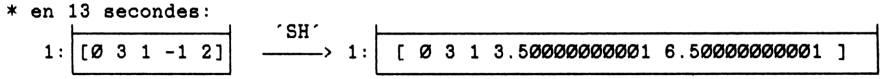
```
«  «  →  n
       «  Ø  n  FOR  i
             Ø  i  1  +  FACT  INV
          2  STEP
          IF  n  2  MOD  NOT  THEN  DROP  END
          n  1  +  →ARRY
       »
    »
    →  d
    «  IF  DUP  TYPE  THEN
          { 1 }  GETI  SWAP  DROP
          →  fØ
          «  IF  fØ  THEN
                { 1 }  Ø  PUT  DUP  SH  fØ  COSH  *
                SWAP  CH  fØ  SINH  *  ΣDL
             ELSE
                DUP  DIM  /  FLOOR  d  EVAL  CPDL
             END
          »
       ELSE
          d  EVAL
       END
    »
»
```

Exemples:

* en moins d'une seconde:



* en 13 secondes:



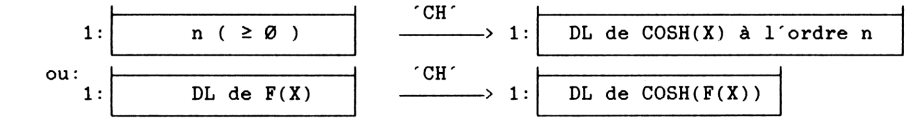
Car SINH(3*X +X^2 -X^3 +2*X^4 +o(X^4))
= 3*X +X^2 +3.5*X^3 +6.5*X^4 +o(X^4).

Répertoire 'DL' programme 'CH'

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 ≥ Ø), 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).

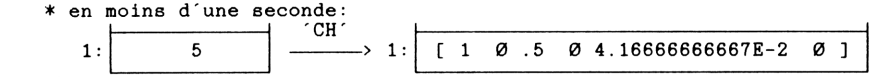
N.B: 'CH' appelle les programmes 'DIM', 'CPDL', 'SH', 'CH', et 'ΣDL'.

'CH':

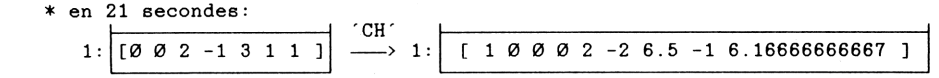
```
«  «  →  n
      «  Ø  n  FOR  i
            i  FACT  INV  Ø
         2  STEP
         IF  n  2  MOD  NOT  THEN  DROP  END
         n  1  +  →ARRY
      »
   »
   →  d
   «  IF  DUP  TYPE  THEN
         { 1 }  GETI  SWAP  DROP
         →  fØ
         «  IF  fØ  THEN
               { 1 }  Ø  PUT  DUP  CH  fØ  COSH  *
               SWAP  SH  fØ  SINH  *  ΣDL
            ELSE
               DUP  DIM  /  FLOOR  1  +  d  EVAL  CPDL
            END
         »
      ELSE
         d  EVAL
      END
   »
»
```

Exemples:

* en moins d'une seconde:



* en 21 secondes:



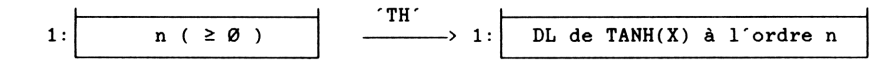
Car COSH(2*X^2 -X^3 +3*X^3 +X^4 +X^5 +o(X^5))
= 1 +2*X^4 -2*X^5 +6.5*X^6 -X^7 +6.16666666667*X^8 + o(X^8).

Répertoire 'DL' programme 'TH'

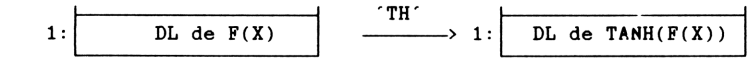
DEVELOPPEMENT LIMITE DE TANH(X) ET TANGENTE HYPERBOLIQUE D'UN D.L.

'TH' permet tout à la fois de calculer le développement limité de TANH(X) à un ordre n quelconque (n ≥ Ø), ou celui de TANH(F(X)) ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



ou:



Dans ce dernier cas le D.L. de TANH(F(X)) est obtenu au même ordre que celui de F(X).

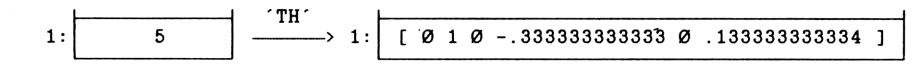
N.B: 'TH' appelle les programmes 'CH', 'SH', 'QDL'.

'TH':

« DUP SH SWAP CH QDL »

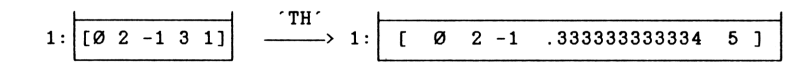
Exemples:

* en 8 secondes:



car TAN(X) = X - (1/3)*X^3 +(2/15)*X^5 + o(X^5).

* en 36 secondes:



car 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) .

Répertoire 'DL' programme 'ASN'

DEVELOPPEMENT LIMITE DE ASIN(X) ET ARC-SINUS D'UN D.L.

'ASN' permet tout à la fois de calculer le développement limité de ASIN(X) à un ordre n quelconque (n ≥ Ø), ou celui de ASIN(F(X)) ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:

```
1: [ n ( ≥ Ø ) ]  'ASN' ----> 1: [ DL de ASIN(X) à l'ordre n ]
```

ou:

```
1: [ DL de F(X) ]  'ASN' ----> 1: [ DL de ASIN(F(X)) ]
```

Dans ce dernier cas le D.L. de ASIN(F(X)) est obtenu au même ordre que celui de F(X). Le coefficient F(Ø) doit être nul.

N.B: 'ASN' appelle les programmes 'DIM', 'CPDL'.

'ASN':

```
«  «  →  n
      «  Ø  n  FOR  i
            Ø  'FACT(i)/2^i/FACT(i/2)^2/(i+1)'  EVAL
         2  STEP
         IF  n  2  MOD  NOT  THEN  DROP  END
         n  1  +  →ARRY
      »
   »
   →  d
   «  IF  DUP  TYPE  THEN
         { 1 }  GETI  SWAP  DROP
         IF  THEN  "Erreur"  ABORT  END
         DUP  DIM  /  FLOOR  d  EVAL  CPDL
      ELSE
         d  EVAL
      END
   »
»
```

Exemples:

* en une seconde:

```
1: [ 6 ]  'ASN' ----> 1: [ Ø  1  Ø  .166666666667  Ø  .Ø75  Ø ]
```

* en 14 secondes:

```
1: [Ø 1 -3 2 5]  'ASN' ----> 1: [ Ø  1  -3  2.16666666667  3.5 ]
```

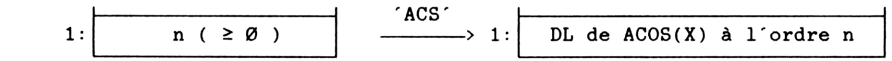
Car ASIN(X -3*X^2 +2*X^3 +5*X^4 +o(X^4))
= X -3*X^2 +2.16666666667*X^3 +3.5*X^4 +o(X^4).

Répertoire 'DL' programme 'ACS'

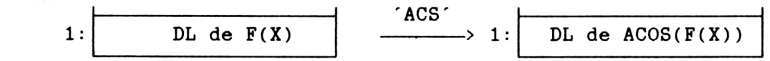
DEVELOPPEMENT LIMITE DE ACOS(X) ET ARC-COSINUS D'UN D.L.

'ACS' permet tout à la fois de calculer le développement limité de ACOS(X) à un ordre n quelconque (n ≥ Ø), ou celui de ACOS(F(X)) ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



ou:



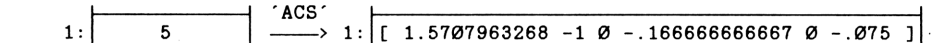
Dans ce dernier cas le D.L. de ACOS(F(X)) est obtenu au même ordre que celui de F(X). Le coefficient F(Ø) doit être nul.

N.B: 'ACS' appelle le programme 'ASN'.

'ACS':

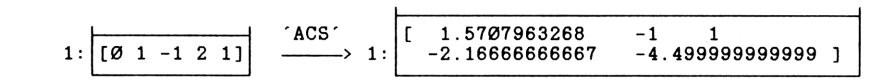
```
« ASN NEG { 1 } π →NUM 2 / PUT »
```

* en une seconde:



Car ACOS(X)= π/2 -X -(1/6)*X^3 -(3/4Ø)*X^5 +o(X^5).

* en 14 secondes:



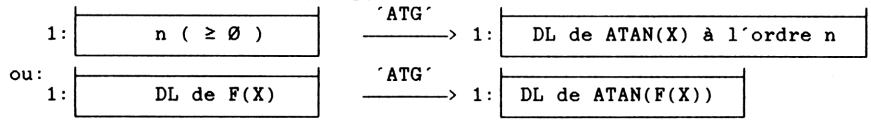
Car ACOS(X -X^2 +2*X^3 +X^4 +o(X^4))
= π/2 -X +X^2 -(13/6)*X^3 -(9/2)*X^4 +o(X^4).

Répertoire 'DL' programme 'ATG'

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 ≥ Ø), 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(Ø) doit être nul.

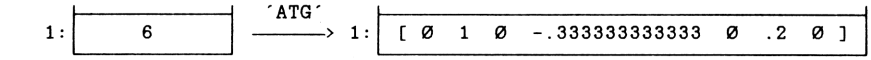
N.B: 'ATG' appelle les programmes 'DIM', 'CPDL'.

'ATG':

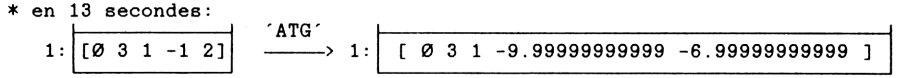
```
«    «   →  n
        «   Ø   n   FOR  i
                Ø   i   1   +   INV
                IF   i   4   MOD   THEN   NEG   END
            2   STEP
            IF   n   2   MOD   NOT   THEN   DROP   END
            n   1   +   →ARRY
        »
    »
    →  d
    «  IF   DUP   TYPE   THEN
          { 1 }   GETI   SWAP   DROP
          IF   THEN   "Erreur"   ABORT   END
          DUP   DIM   /   FLOOR   d   EVAL   CPDL
       ELSE
          d   EVAL
       END
    »
»
```

Exemples:

* en une seconde:



* en 13 secondes:



Car ATAN(3*X +X^2 -X^3 +2*X^4 +o(X^4))
= 3*X +X^2 -1Ø*X^3 -7*X^4 +o(X^4).

Répertoire 'DL' programme 'ASH'

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 ASINH(X) à un ordre n quelconque (n ≥ Ø), ou celui de ASINH(F(X)) ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:

```
   1: | n ( ≥ Ø ) |  ---'ASH'--->  1: | DL de ASINH(X) à l'ordre n |
```

ou:

```
   1: | DL de F(X) |  ---'ASH'--->  1: | DL de ASINH(F(X)) |
```

Dans ce dernier cas le D.L. de ASINH(F(X)) est obtenu au même ordre que celui de F(X). Le coefficient F(Ø) doit être nul.

N.B: 'ASH' appelle les programmes 'DIM', 'CPDL'.

'ASH':

```
« « → n
    « Ø n FOR i
        Ø 'FACT(i)/2^i/FACT(i/2)^2/(i+1)' EVAL
        IF i 4 MOD THEN NEG END
      2 STEP
      IF n 2 MOD NOT THEN DROP END
      n 1 + →ARRY
    »
  »
  → d
  « IF DUP TYPE THEN
       { 1 } GETI SWAP DROP
       IF THEN "Erreur" ABORT END
       DUP DIM / FLOOR d EVAL CPDL
    ELSE
       d EVAL
    END
  »
»
```

Exemples:

* en une seconde:

```
   1: |  6  |  ---'ASH'--->  1: | [ Ø 1 Ø -.166666666667 Ø .Ø75 Ø ] |
```

* en 14 secondes:

```
   1: | [Ø 1 -3 2 5] |  ---'ASH'--->  1: | [ Ø 1 -3 1.83333333333 6.5 ] |
```

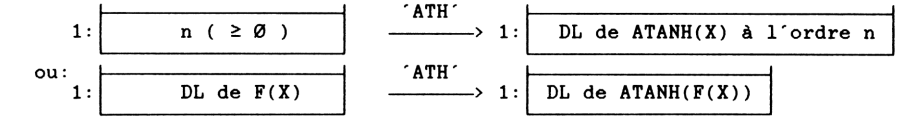
Car ASINH(X -3*X^2 +2*X^3 +5*X^4 +o(X^4))
= X -3*X^2 +(11/6)*X^3 +(13/2)*X^4 +o(X^4).

Répertoire 'DL' programme 'ATH'

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 ≥ Ø), ou celui de ATANH(F(X)) ou F est elle même donnée par son développement limité.

Le schéma fonctionnel est:



ou:

Dans ce dernier cas le D.L. de ATANH(F(X)) est obtenu au même ordre que celui de F(X). Le coefficient F(Ø) doit être nul.

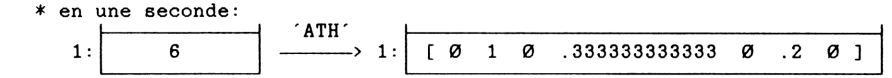
N.B: 'ATH' appelle les programmes 'DIM', 'CPDL'.

'ATH':

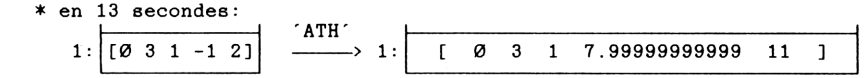
```
« « → n
     « Ø n FOR i
          Ø i 1 + INV
        2 STEP
        IF n 2 MOD NOT THEN DROP END
        n 1 + →ARRY
     »
  »
  → d
  « IF DUP TYPE THEN
       { 1 } GETI SWAP DROP
       IF THEN "Erreur" ABORT END
       DUP DIM / FLOOR d EVAL CPDL
     ELSE
       d EVAL
     END
  »
»
```

Exemples:

* en une seconde:



* en 13 secondes:



Car ATANH(3*X +X^2 -X^3 +2*X^4 +o(X^4))
= 3*X +X^2 +8*X^3 +11*X^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 connait les éléments essentiels, de calculer des distances ou des écarts angulaires, enfin de permettre le passage d'un système de coordonnées à un autre dans l'espace.

Le répertoire 'GEOM' contient les programmes suivants:

'DRTE': équation d'une droite dont on connait:
* deux points
ou * un point et un vecteur directeur.

'PLAN': équation d'un plan dont on connait:
* 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 connait:
* le centre et le rayon
ou * deux points diamétralement opposés.

'DIST': calcul de la distance entre:
* un point et une droite (dans le plan ou dans l'espace)
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

'COORD': passage d'un système de coordonnées à un autre en dimension 3 entre les systèmes suivants:
* coordonnées cartésiennes.
* coordonnées cylindriques (semi-polaires)
* coordonnées sphériques.

Répertoire 'GEOM' programme 'DRTE'

EQUATION D'UNE DROITE DU PLAN.

'DRTE' permet d'obtenir, sous forme symbolique, l'équation d'une droite D du plan dont on connait:

* 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.

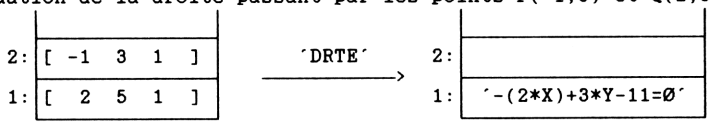
Un point de coordonnées x,y est représenté par le vecteur [x, y, 1]
Un vecteur de composantes x,y est représentée par [x, y, Ø].
C'est donc la troisième composante qui détermine s'il s'agit d'un point ou d'un vecteur.

L'équation est obtenue sous la forme symbolique: 'A*X+B*Y+C=Ø'.

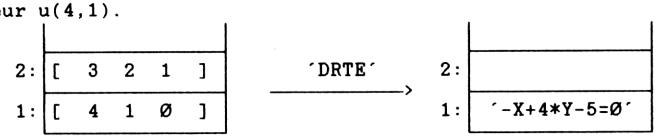
'DTRE':

```
«  CROSS  ARRY→  DROP  ROT  'X'  *
   ROT  'Y'  *  +  SWAP  +  Ø  =
»
```

Exemple1:
(en une seconde)
Equation de la droite passant par les points P(-1,3) et Q(2,5).



Exemple2:
(en une seconde)
Equation de la droite passant par le point P(3,2) et de vecteur directeur u(4,1).



Dans l'exemple ci-dessus, on peut inverser l'ordre dans lequel sont donnés P et u.

Répertoire 'GEOM' programme 'PLAN'

EQUATION D'UN PLAN.

'PLAN' permet d'obtenir, sous forme symbolique, l'équation d'un plan (π) dont on connait:
* 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 Ø].
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=Ø'.

'PLAN':

```
«   →  a   b   c
    «  a   ARRY→   DROP   b   ARRY→   DROP   c   ARRY→   DROP
       { 3 4 }   →ARRY    STOΣ
       [ 1 Ø Ø Ø ]   Σ+    RCLΣ   DET   'X'   *   Σ-    DROP
       [ Ø 1 Ø Ø ]   Σ+    RCLΣ   DET   'Y'   *   +   Σ-    DROP
       [ Ø Ø 1 Ø ]   Σ+    RCLΣ   DET   'Z'   *   +   Σ-    DROP
       [ Ø Ø Ø 1 ]   Σ+    RCLΣ   DET   +   Ø   =   CLΣ
    »
»
```

Exemple1:
Equation du plan passant par les points P(-1,2,3) Q(4,1,5) R(2,-3,Ø):
(en sept secondes)

3:	[-1 2 3 1]			3:	
2:	[4 1 5 1]		'PLAN' →	2:	
1:	[2 -3 Ø 1]			1:	'-13*X-21*Y+22*Z-37=Ø'

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 sept secondes)

3:	[-2 1 3 Ø]			3:	
2:	[-1 1 -3 1]		'PLAN' →	2:	
1:	[-1 4 -2 Ø]			1:	'-(14*X)-7*Y-7*Z-28=Ø'

Répertoire 'GEOM' programme 'CERC'

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 connait:
* 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] (si on est 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=Ø' dans le cas d'un cercle.
'X^2+Y^2+Z^2+A*X+B*Y+C*Z+D=Ø' dans le cas d'une sphère.

```
'CERC':
    «  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
          'X^2+Y^2'  IF  d  3  ==  THEN  'Z^2'  +  END
          'a'  1  GET  2  *  NEG  'X'  *  +  'a'  2  GET  2
          *  NEG  'Y'  *  +
          IF  d  3  ==  THEN
             'a'  3  GET  2  *  NEG  'Z'  *  +
          END
          a  DUP  DOT  b  SQ  -  +  Ø  =
       »
    »
```

Exemple1: (en moins de deux secondes)
Equation du cercle de centre Ω(2,3) et de rayon 5.

2:	[2 3]	'CERC' →	2:	
1:	5		1:	'X^2+Y^2-4*X-6*Y-12=Ø'

Exemple2: (temps de calcul = deux 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=Ø'

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 d'équation '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.

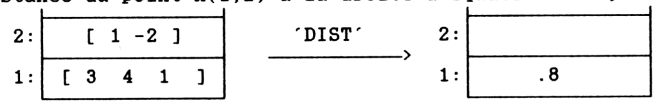
La distance cherchée est obtenue au niveau 1.

```
'DIST':
    «   DUP2  TYPE  SWAP  TYPE
        →  t1  t2
        «  IF  t1  3  ==  t2  3  ==  AND  THEN
              →  p  e
              «  e  p  SIZE  RDM  DUP  p  DOT  e  DUP  SIZE
                 GET  +  ABS  SWAP  ABS  /
              »
           ELSE
              IF  t2  3  ==  t1  5  ==  AND  THEN
              →  p  d
              «  d  LIST→ DROP  DUP  ROT  p  -  CROSS  ABS  SWAP  ABS  /
              »
              ELSE
                 →  d1  d2
                 «  'd1'  2  GET  'd2'  2  GET  DUP2  ARRY→
                    DROP  4  ROLL  ARRY→  DROP  'd1'  1  GET
                    'd2'  1  GET  -  ARRY→  DROP  { 3 3 }  →ARRY
                    DET  ABS  3  ROLLD  CROSS  ABS  /
                 »
              END
           END
        »
    »
```

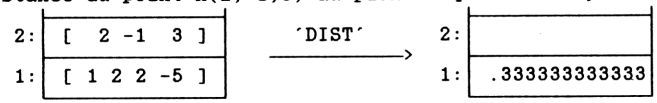
Répertoire 'GEOM' programme 'DIST' (suite)

PROGRAMME 'DIST' EXEMPLES D'UTILISATION.

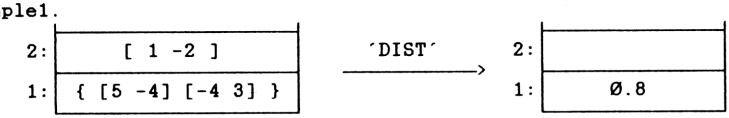
Exemple1: (en une seconde)
Distance du point M(1,2) à la droite d'équation 3x+4y+1=Ø.



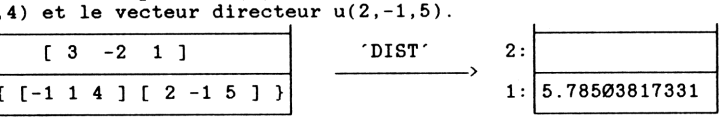
Exemple2: (en moins de deux secondes)
Distance du point M(2,-1,3) au plan d'équation x+2y+2z-5=Ø.



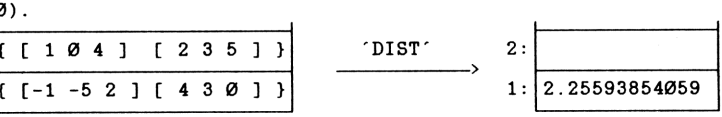
Exemple3: (en une seconde)
Distance du point M(1,-2) à la droite définie par le point A(5,-4) et le vecteur directeur u=(-4,3). Ce sont les mêmes données que dans l'exemple1.



Exemple4: (en 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).



Exemple5: (en deux secondes)
Distance de la droite D définie par le point A(1,Ø,-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,Ø).



Répertoire 'GEOM' programme 'ANGL'

CALCUL D'ECART ANGULAIRE:

'ANGL' calcule l'écart angulaire θ (Ø ≤ θ ≤ π/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=Ø (resp. tout plan d'équation ax+by+cz+d=Ø).

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 et en degrés (avec le format HMS). Les deux résultats sont donnés avec quatre décimales et le mode interne du calculateur n'est pas modifié.

Il suffit de presser une touche quelconque pour sortir du programme 'ANGL'.

'ANGL':

```
«  RCLF
   → a b x
   « a b DOT ABS a ABS b ABS * / RAD
     ACOS DUP 4 FIX "En radians" 1 DISP 2 DISP
     "En degres (HMS)" 3 DISP R→D →HMS 4 DISP
     x STOF
     DO UNTIL KEY END
     DROP CLMF
   »
»
```

Exemple:

Ecart angulaire entre les vecteurs u(1,-4,2) et v(3,1,5).
(en une seconde)

```
                                          4: En radians
2: [ 1 -4 2 ]      'ANGL'                 3: 1.2324
                  ------------>           2: En degres (HMS)
1: [ 3  1 5 ]                             1: 70.3642
```

Répertoire 'GEOM' programme 'COORD'

CHANGEMENTS DE COORDONNEES EN DIMENSION 3.

'COORD' permet d'effectuer le passage d'un système de coordonnées à un autre parmi les systèmes suivants:

* coordonnées cartésiennes: (x,y,z).
* coordonnées cylindriques (ou semi-polaires): (ro , θ, z).
* coordonnées sphériques:(r, fi, θ) où fi représente la "longitude" et θ la "colatitude".

les formules permettant de calculer les coordonnées cartésiennes en fonction des autres sont:

x = ro*cos(θ) y = ro*sin(θ) z = z	et
x = r*sin(θ)*cos(fi) y = r*sin(θ)*sin(fi) z = r*cos(θ)	

Réciproquement le programme 'COORD' permet de calculer pour tout point M de l'espace,

* le triplet de ses coordonnées polaires (ro, θ, z), avec ro ≥ Ø et $-\pi < \theta \leq \pi$.
* le triplet de ses coordonnées sphériques (r, fi, θ), avec r ≥ Ø, $Ø \leq \theta \leq \pi$, et $-\pi < fi \leq \pi$.

Le fonctionnement de 'COORD' est le suivant:

On place au niveau 1 de la pile le vecteur [a b c] de coordonnées que l'on veut traduire.
On appelle le programme 'COORD'.
Immédiatement, la chaîne de caractères "P→C C→P S→C C→S P→S S→P" (longue de 23 caractères) est placée au niveau 4 de la pile (donc juste au dessus des 6 touches du menu "curseur".

L'appui sur une des touches de ce menu "curseur" effectue alors la traduction correspondante entre systèmes de cooordonnées.

Par exemple:
La touche "INS", qui se trouve en dessous de "P→C", permet d'effectuer la traduction "polaires —> cartésiennes".
La touche ▼ , qui se trouve en dessous de "C→S", permet d'effectuer la traduction "cartésiennes —> sphériques".

Attention: les arguments et les résultats, lorsqu'ils sont des angles (coordonnées polaires ou sphériques) doivent être interprétés en fonction du mode interne du calculateur (mode "radians" ou non).

Répertoire 'GEOM' programme 'COORD'

TEXTE DU PROGRAMME 'COORD' ET EXEMPLE D'UTILISATION.

```
'COORD':
    «    "P→C C→P S→C C→S P→S S→P"   4   DISP
         DO   UNTIL   KEY   END
         →  v   t
         «  IF  v   SIZE   { 3 }   ==   THEN   v   ARRY→  DROP
            →  a   b   c
            «  IF  t   "INS"   ==   THEN
                     a   b   R→C   P→R   C→R   c   3   →ARRY
               END
               IF  t   "DEL"   ==   THEN
                  a   b   R→C   R→P   C→R   c   3   →ARRY
               END
               IF  t   "UP"   ==   THEN
                  b   SIN   c   COS   *   b   SIN   c   SIN   *
                  b   COS   3   →ARRY   a   *
               END
               IF  t   "DOWN"   ==   THEN
                  a   b   R→C   R→P   C→R   c   ROT   R→C   R→P   C→R
                  ROT   3   →ARRY
               END
               IF  t   "LEFT"   ==   THEN
                  c   a   R→C   R→P   C→R   b   3   →ARRY
               END
               IF  t   "RIGHT"   ==   THEN
                  a   b   R→C   P→R   C→R   c   ROT   3   →ARRY
               END
            »
            END
         »
         CLMF
    »
```

Exemples: en mode 3 FIX et en mode 'degrés'.

```
                    'C→S'
[ 1 1 1 ]      ————————————>    [ 1.732  54.736  45 ]

                    'S→C'
[ 1 60 30 ]    ————————————>    [ 0.750 0.433 0.500 ]

                    'P→C'
[ 1 45 10 ]    ————————————>    [ 0.707 0.707 10 ]

                    'C→P'
[ 1 1 1 ]      ————————————>    [ 1.414 45 1 ]

                    'P→S'
[ 1 30 45 ]    ————————————>    [ 45.011 1.273 30 ]

                    'S→P'
[ 1 45 45 ]    ————————————>    [ 0.707 45 0.707 ]
```

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 HP28S 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 possiblité qu'a le HP28 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.

'GRAD' : Calcul du gradient d'une fonction de plusieurs variables.

'DELTA' : Calcul du laplacien d'une fonction de plusieurs variables.

'DIFF' : Calcul de la différentielle d'une fonction de plusieurs variables.

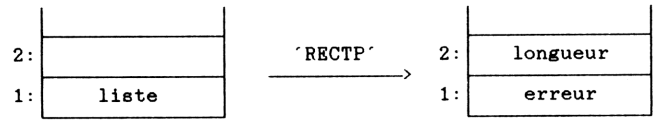
Répertoire 'GDIF' Programme 'RECTP'

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 ≤ X ≤ B.
* par une représentation paramétrique X=X(T), Y=Y(T), où A ≤ T ≤ B.
* par une équation en polaires RO=RO(T), où A ≤ T ≤ B.

Le schéma fonctionnel est:



où "liste" est une liste contenant les éléments nécessaires au calcul,
où "longueur" est une valeur approchée du résultat,
où "erreur" est un majorant de l'erreur absolue commise.

L'organisation de "liste" est la suivante:
{ F(X) A B EPS } dans le premier cas,
{ X(T) Y(T) A B EPS } dans le deuxième cas,
{ RO(T) A B EPS } dans le dernier cas,
Avec:
A: valeur inférieure du paramètre (X ou T).
B: valeur supérieure du paramètre (X ou T).
EPS: précision relative souhaitée sur le résultat.
F(X): expression symbolique de la variable 'X'.
X(T), Y(T), RO(T): expressions symboliques de la variable 'T'.

```
'RECTP':
    «   LIST→   { X T }   PURGE
        →  a   b   eps   n
        «  IF   n   5   ==   THEN
                'T'  ∂   2   ^   SWAP   'T'  ∂   2   ^   +   ƒ   'T'
           ELSE
              DUP   'T'  ∂
              IF   DUP   Ø   SAME   THEN
                   DROP   'X'  ∂   2   ^   1   +   ƒ   'X'
              ELSE
                   2   ^   SWAP   2   ^   +   ƒ   'T'
              END
           END
           a   b   3   →LIST   eps   ∫
        »
    »
```

Répertoire 'GDIF'

Programme 'RECTP'
(suite)

PROGRAMME 'RECTP' : EXEMPLES D'UTILISATION.

Exemple 1: Longueur de l'arc de chaînette Y=CH(X), et Ø ≤ X ≤ 1.
(avec une erreur relative inférieure à 1E-5)

2:				2:	1.17520119384
1:	{ 'COSH(X)' Ø 1 .ØØØØ1 }	'RECTP' ⟶		1:	1.17510685202E-5

Le résultat est obtenu en environ 7 secondes. On montre que le résultat exact est: SH(1)= (e - 1/e)/2 ≈ 1.17520119364.
L'erreur réellement commise ici est donc ≈ 2E-1Ø.

Exemple 2: Longueur de l'arc de cycloïde X=COS(T), Y=T-SIN(T), entre T=Ø et T=2π.
(avec une erreur relative inférieure à 1E-3)

2:					8.ØØØØØØ88Ø95
1:	{ 'COS(T)' 'T-SIN(T)' Ø '2*π' .ØØ1 }	'RECTP' ⟶		1:	8.ØØ78314391E-3

Le résultat est obtenu en environ 9 secondes. On montre que le résultat exact est 8. L'erreur réellement commise ici est donc ≈ 9E-7.
Remarque: sur l'exemple ci-dessus, il n'est pas nécessaire d'armer ou de désarmer l'indicateur binaire 35 (mode d'évaluation des constantes symboliques).

Exemple 3: Longueur de la cardioïde RO=1+COS(T) (avec une précision relative inférieure à 1E-4).
Pour obtenir la longueur totale, T doit varier de Ø à 2π.

2:					8.ØØØØØØ97338
1:	{ '1+COS(T)' Ø '2*π' .ØØØ1 }	'RECTP' ⟶		1:	7.9810519651E-4

Le résultat est obtenu en environ 9 secondes. On montre que le résultat exact est 8. L'erreur réellement commise ici est donc ≈ 1E-6.

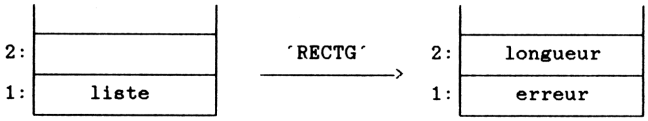
Répertoire 'GDIF' Programme 'RECTG'

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 ≤ T ≤ B.

Le schéma fonctionnel est:



où "liste" est une liste contenant les éléments nécessaires au calcul,
où "longueur" est une valeur approchée du résultat,
où "erreur" est un majorant de l'erreur absolue commise.

L'organisation de "liste" est la suivante:

{ X(T) Y(T) Z(T) A B EPS } dans le premier cas,
{ RO(T) Z(T) A B EPS } dans le deuxième cas.

Avec:

A: valeur inférieure du paramètre T.
B: valeur supérieure du paramètre T.
EPS: précision relative souhaitée sur le résultat.
X(T),Y(T),Z(T),RO(T): expressions symboliques de la variable 'T'.

'RECTG':

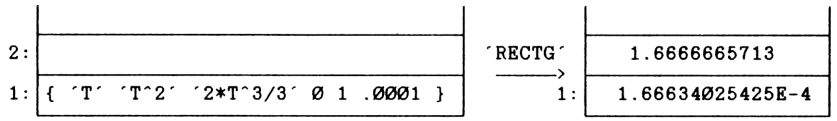
```
«   LIST→  'T'  PURGE
    →  a  b  eps  n
    «  'T'  ∂  2  ^  SWAP
       IF  n  6  ==  THEN
           'T'  ∂  2  ^  +  SWAP  'T'  ∂  2  ^
       ELSE
           DUP  'T'  ∂  2  ^  SWAP  2  ^  +
       END
       +  ∫  'T'  a  b  3  →LIST  eps  ∫
    »
»
```

Répertoire 'GDIF'

Programme '**RECTG**'
(suite)

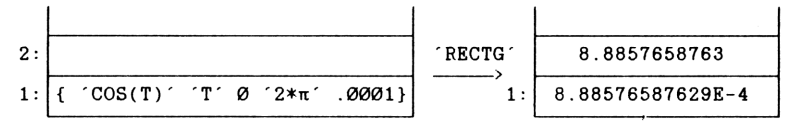
PROGRAMME 'RECTG' : EXEMPLES D'UTILISATION.

Exemple 1: Longueur de l'arc défini par la représentation paramétrique
Y=X, Z=(2/3)X^3, où Ø ≤ X ≤ 1, et avec une précision relative inférieure à 1E-4.
Ici il faut prendre X pour paramètre T.



Le résultat est obtenu en 1Ø secondes.
On montre que le résultat exact est 5/3 ≈ 1.66666666667.
L'erreur réellement commise ici est donc ≈ -1E-7.

Exemple 2: Longueur de l'arc d'hélice circulaire défini en coordonnées cylindriques par: RO=COS(T), Z=T, où Ø ≤ T ≤ 2π, avec une précision relative inférieure à 1E-4.



Le résultat est obtenu en 4 secondes.
On montre que le résultat exact est 2√2π ≈ 8.8857658763.
L'erreur réellement commise ici est donc nulle (en machine).

Répertoire 'GDIF' Programme 'CURV'

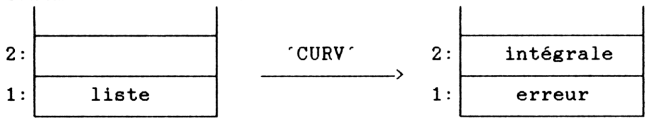
CALCUL D'INTEGRALES CURVILIGNES.

'CURV' calcule l'integrale 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 ≤ T ≤ 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 ≤ T ≤ B.

Le schéma fonctionnel est:



où "liste" est une liste contenant les éléments nécessaires au calcul,
où "intégrale" est une valeur approchée du résultat,
où "erreur" est un majorant de l'erreur absolue commise.

L'organisation de "liste" est la suivante:
{ P(X,Y) Q(X,Y) X(T) Y(T) Z(T) A B EPS } dans le premier cas,
{ P(X,Y,Z) Q(X,Y,Z) R(X,Y,Z) X(T) Y(T) Z(T) A B EPS } dans le 2me cas.

Avec:
A: valeur inférieure du paramètre T.
B: valeur supérieure du paramètre T.
EPS: précision relative souhaitée sur le résultat.
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'.

```
'CURV':
    «   LIST→  'T'  PURGE
        →  a  b  eps  n
        «  IF  n  9  ==  THEN  'Z'  STO  END
           'Y'  STO  'X'  STO
           IF  n  9  ==  THEN
               'Z'  'T'  ∂  *
             ELSE
               Ø
           END
           3  ROLLD  'Y'  'T'  ∂  *  SWAP
           'X'  'T'  ∂  *  +  +
           'T'  a  b  3  →LIST  eps  ∫
           { X Y Z }  PURGE
        »
    »
```

Répertoire ´GDIF´ Programme ´CURV´
(suite)

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-SIN(T), Y=1-COS(T), Ø ≤ T ≤ 2*π.
(avec une erreur relative inférieure à 1E-4).

On place la liste
{ ´2-Y´ ´X´ ´T-SIN(T)´ ´1-COS(T)´ Ø ´2*π´ .ØØØ1 }
au niveau 1 de la pile et on appelle ´CURV´.

Au bout de 14 secondes, on obtient:

```
2:  -6.28321695462
1:  1.25331364126E-3
```

On montre que le résultat exact est:
$-2\pi \approx -6.28318530718$.
L´erreur réellement commise ici est donc égale à ≈ -3E-5.

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=COS(T), Y=SIN(T), Z=T, Ø≤T≤2π.

On place la liste
{ ´Y-Z´ ´Z-X´ ´X-Y´ ´COS(T)´ ´SIN(T)´ ´T´ Ø ´2*π´ .ØØØ1 }
au niveau 1 de la pile et on appelle ´CURV´.

Au bout de 18 secondes, on obtient:

```
2:  -12.5663887599
1:  1.81629778555E-3
```

On montre que le résultat exact est:
$-4\pi \approx -12.5663706144$.
L´erreur réellement commise ici est donc égale à ≈ -2E-5.
(le temps de calcul est de 3Ø secondes avec eps= .ØØØØØ1 et on obtient -12.566370626)

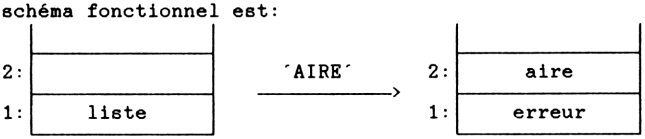
Répertoire 'GDIF' Programme 'AIRE'

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 ≤ T ≤ B.
2) en coordonnées polaires RO=RO(T) avec A≤ T ≤ B.

Le schéma fonctionnel est:



où "liste" est une liste contenant les éléments nécessaires au calcul,
où "aire" est une valeur approchée du résultat,
où "erreur" est un majorant de l'erreur absolue commise.

L'organisation de "liste" est la suivante:
{ X(T) Y(T) A B EPS } dans le premier cas,
{ RO(T) A B EPS } dans le deuxième cas.

Avec:

A: valeur inférieure du paramètre T.
B: valeur supérieure du paramètre T.
EPS: précision relative souhaitée sur le résultat.
X(T), Y(T), RO(T): expressions symboliques de la variable 'T'.

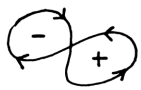
Remarques:

Dans le premier cas, l'intégrale calculée est $\int_{\Gamma} XdY$, où Γ est la courbe frontière parcourue dans le sens T=A->T=B.

Dans le second cas, l'intégrale calculée est $(1/2)\int_{A}^{B} RO^2(T)\,dT$.

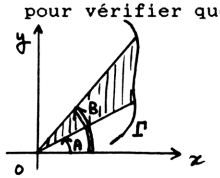
Le sens A->B doit correspondre à une orientation de Γ dans le sens trigonométrique (lors du parcours de la courbe frontière, les points du domaine intérieur doivent être sur la gauche). dans le cas contaire, on obtient une aire comptée négativement.

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 droites d'angles polaires A et B.



Répertoire 'GDIF'

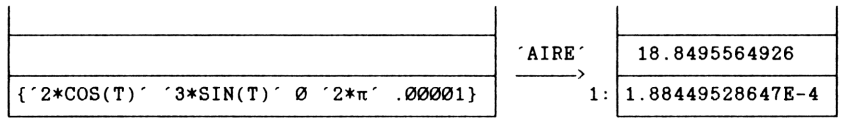
Programme 'AIRE'
(suite)

TEXTE DU PROGRAMME 'AIRE' - EXEMPLES D'UTILISATION.

```
'AIRE'
    «   LIST→   'T'   PURGE
        →  a   b   eps   n
        «  IF   n   4   ==   THEN
                 2   ^   2   /
             ELSE
                'T'  ∂   *
             END
             'T'   a   b   3   →LIST   eps   ∫
        »
    »
```

Exemple 1:

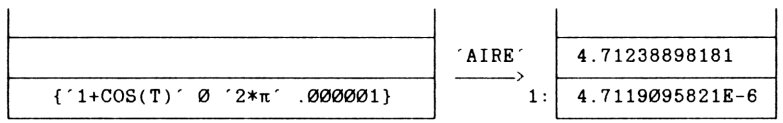
On veut déterminer l'aire intérieure à l'ellipse d'équation:
$X^2/4 + Y^2/9 = 1$
Une représentation paramétrique de cette ellipse est donnée par:
$X=2\cos(T)$, $Y=3\sin(T)$, avec $0 \le T \le 2\pi$.
(on veut un résultat entaché d'une erreur relative inférieure à 1E-5)
L'utilisation du programme 'AIRE' donne ici:



Le résultat est obtenu en 13 secondes.
On montre que le résultat exact est $6\pi \approx 18.8495559215$.
L'erreur réellement commise ici est $\approx$ 6E-7.

Exemple 2:

On veut déterminer l'aire intérieure à la cardioïde RO=1+COS(T).
Celle-ci est entièrement parcourue quand T parcourt le segment $[0,2\pi]$.
(on veut un résultat entaché d'une erreur relative inférieure à 1E-6)
L'utilisation du programme 'AIRE' donne ici:



Le résultat est obtenu en 21 secondes.
On montre que le résultat exact est $3\pi/2 \approx 4.71238898038$.
L'erreur réellement commise ici est $\approx$ 1E-9.

Répertoire 'GDIF' Programme 'CRBRE'

CENTRE ET RAYON DE COURBURE D'UNE COURBE PLANE.

'CRBRE' calcule les coordonnées du centre de courbure, et le rayon de courbure, en un point quelconque d'une courbe plane définie:

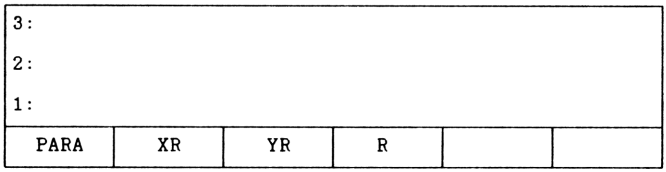
1) par une équation Y=F(X).
2) par une représentation paramétrique X=X(T), Y=Y(T).
3) par une représentation en polaires RO=RO(T).

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), un menu personnalisé apparaît, l'écran devenant:



(PARA , XR , YR et R apparaissent en vidéo inverse)

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 'INS' (qui se trouve en dessous de PARA).
Une pression sur 'DEL' (en dessous de XR) donne l'abscisse XR du centre de courbure en ce point.
Une pression sur ▲ (en dessous de YR) donne l'ordonnée YR du centre de courbure en ce point.
Une pression sur ▼ (qui est en dessous de R) donne le rayon de courbure R en ce point.

On peut répéter ces opérations en un nombre de points quelconque. Pendant ce temps le programme 'CRBRE' reste suspendu. Pour en terminer avec ce programme, il suffit de faire CONT. On revient ainsi au menu du répertoire.

Les calculs de R, XR, et 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.

Répertoire 'GDIF' Programme 'CRBRE' (suite)

TEXTE DU PROGRAMME 'CRBRE'

'CRBRE':

```
«  { T }  PURGE
   IF  DUP  TYPE  5  ==  THEN
       LIST→  DROP
   ELSE
       IF  DUP  'T'  ∂  Ø  SAME  THEN
           'X'  SWAP  'T'  'X'  STO
       ELSE
           DUP  'COS(T)'  *  SWAP  'SIN(T)'  *
       END
   END
   →  x  y
   «  x  'T'  ∂  DUP  'T'  ∂  y  'T' ∂ DUP  'T'  ∂
      →  x1  x2  y1  y2
      «  x1  y2  *  y1  x2  *  -  x1  2  ^  y1  2  ^  +
         →  d  n
         «  n  d  /  y1  *  NEG  x  +  n  d  /  x1
            *  y  +  n  1.5  ^  d  /
            →  xr  yr  r
            «  «  'T'  STO  »  'PARA'  STO
               «  xr  EVAL  »  'XR'  STO
               «  yr  EVAL  »  'YR'  STO
               «  r  EVAL  »  'R'  STO
               { PARA XR YR R }  MENU
               HALT
               { X T PARA XR YR R }  PURGE
               VARS  MENU
            »
         »
      »
   »
»
```

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 quatre secondes, le menu personnalisé apparaît.

Si on donne au paramètre (ici X) la valeur Ø, on trouve:
XR=Ø, YR=2, R=1.

Si on donne au paramètre la valeur 1, on trouve (en mode 5 fix):
XR=-Ø.81343, YR=3.Ø8616, R=2.3811Ø.

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 quatre secondes, le menu personnalisé apparaît.

Si on donne au paramètre (ici T) la valeur π, on trouve:
XR=3.14159265359 (=π), YR=-2, R=-4.

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 cycloïde 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 17 secondes, le menu personnalisé apparaît.

Si on donne au paramètre (ici T) la valeur Ø, on trouve (mode 5 FIX):
XR=.66667, YR=Ø, R=1.33333.

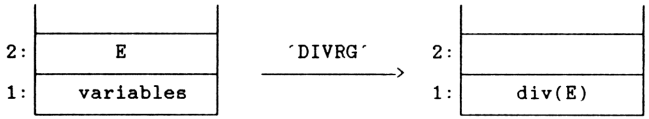
Si on donne au paramètre la valeur π/2, on trouve (en mode 5 FIX):
XR=.66667, YR=.33333, R=.94281.

Remarque: ici la valeur théorique de R est, au point de paramètre T:
R=4/3*COS(T/2).

Répertoire 'GDIF' Programme 'DIVRG'

DIVERGENCE D'UN CHAMP DE VECTEURS.

'DIVRG' calcule la divergence 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 }. Div(E) est alors égal à:

$$\frac{\partial P}{\partial X} + \frac{\partial Q}{\partial Y} + \frac{\partial R}{\partial Z} .$$

N.B: le programme 'DIVRG' est suspendu (avant calcul des dérivées partielles) sur un menu de saisie personnalisé des variables figurant dans la liste "variables".

Ceci permet de calculer div(E) en un point précis, ou bien d'obtenir son expression symbolique (le résultat étant obtenu en appuyant sur CONT).

```
'DIVRG':
    «   DUP   PURGE
        →  f   var
        «  { STO }   var  +   MENU   HALT   Ø
           1   f   SIZE   FOR   i
              'f'   i   GET   'var'   i   GET   ∂   +
           NEXT
           var   PURGE    VARS   MENU
        »
    »
```

PROGRAMME 'DIVRG' : EXEMPLES D'UTILISATION.

Exemple 1:

On veut 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) et Q(X,Y)= EXP(XY)).
On crée la pile ci-dessous et on appelle 'DIVRG', qui est aussitôt suspendu, avec affichage d'un menu de saisie des variables X et Y.

```
2: {'COS(Y/X)' 'EXP(X*Y)'}
1: { X Y }
```

On poursuit alors par CONT, sans donner de valeur à X et à Y. Le résultat est obtenu en 4 secondes.

On touve au niveau 1 de la pile:

```
1: '-(SIN(Y/X)*(-(Y/X^2)))+X*EXP(X*Y)'
```

Ce qui signifie que:

$$div(E)= \frac{\partial P}{\partial X} + \frac{\partial Q}{\partial Y} = \frac{Y}{X^2} SIN(Y / X) + X EXP(XY).$$

Exemple 2:

On veut calculer la divergence du champ
E(X,Y,Z) = (LN(X+Y), Y*Z², X/(YZ)), au point (1,2,3).
(Ici P = LN(X+Y), Q = Y*Z² et R = X/(YZ)).
On crée la pile ci-dessous:

```
2: { 'LN(X+Y)' 'Y*Z^2' 'X/Y/Z' }
1: { X Y Z }
```

et on appelle 'DIVRG'. Celui-ci est alors suspendu avec présentation d'un menu de saisie des variables X, Y , Z.
On donne à X, Y, Z, les valeurs respectives 1, 2, 3.
On poursuit par CONT. le résultat est obtenu au bout de 4 secondes.

On trouve

```
1: 9.27777777777
```

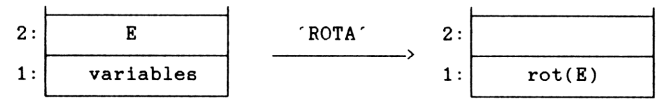
ce qui signifie qu'au point (1,2,3):

$$Div(E) = \frac{\partial P}{\partial X}(1,2,3) + \frac{\partial Q}{\partial Y}(1,2,3) + \frac{\partial R}{\partial Z}(1,2,3) = 167 / 18 .$$

Répertoire 'GDIF' Programme 'ROTA'

ROTATIONNEL D'UN CHAMP DE VECTEURS.

'ROTA' calcule le rotationnel 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.

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 du menu de saisie personnalisé des variables. On peut ainsi obtenir l'expression symbolique du rotationnel, ou sa valeur en un point, suivant que l'on poursuit immédiatement par CONT ou qu'auparavant on donne aux variables des valeurs numériques.

'ROTA':

```
«   IF   DUP   SIZE   3   ==   THEN   DUP   PURGE   Ø
    →  f   var   j
    «  1   3   FOR   i
          IF   i   3   ==   THEN   1   ELSE   i   1   +   END
          'j'   STO
          'f'   j   GET   'var'   i   GET   ∂
          'f'   i   GET   'var'   j   GET   ∂   -
       NEXT
       3   ROLLD   3   →LIST   'f'   STO
       { STO }   var   +   MENU   HALT
       f   LIST→   DROP
       1   3   START   ROT   EVAL   NEXT
       IFERR   3   →ARRY   THEN   →LIST   END
       var   PURGE   VARS   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 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, avec présentation d'un menu personnalisé de saisie des variables X,Y,Z.

1) si on poursuit par CONT 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: rot(E) = (-Z, -X, -Y).

2) si on poursuit par CONT en donnant à X la valeur 1, et en ne donnant aucune valeur à Y et Z, 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 poursuit par CONT en donnant à X,Y,Z les valeurs respectives 1,2 et 3, le résultat obtenu est:

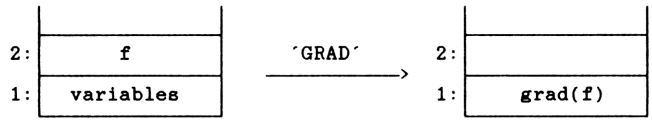
```
1: [ -3  -1  -2 ]
```

On obtient ainsi la valeur du rotationnel de E au point (1,2,3).

Répertoire 'GDIF' Programme 'GRAD'

GRADIENT D'UNE FONCTION DE PLUSIEURS VARIABLES.

'GRAD' 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: Dès l'appel de 'GRAD', celui-ci est suspendu, avec présentation d'un menu personnalisé de saisie des variables. Si on poursuit immédiatement par CONT, on obtient l'expression symbolique de grad(f). Si auparavant on donne aux variables une valeur numérique, on obtient la valeur de grad(f) en un point bien précis.

'GRAD':

```
«   DUP   PURGE
    →   f   var
    «   { STO }   var   +   MENU   HALT
        1   var   SIZE   FOR   i
            f   'var'   i   GET   ∂
        NEXT
        IFERR   var   SIZE   →ARRY   THEN
          →LIST
        END
        var   PURGE   VARS   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 'GRAD'. Celui-ci est alors suspendu, avec présentation d'un menu personnalisé de saisie des variables X,Y.

1) si on poursuit par CONT sans donner de valeur à X,Y,Z, le résultat est (en 5 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 poursuit par CONT en donnant à X,Y les valeurs respectives 1,2 le résultat obtenu est (en MODE 5 FIX) :

```
1: [ 13.4377Ø 3.64392 ]
```

On obtient ainsi la valeur du gradient de f au point (1,2).

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 'GRAD'. Celui-ci est alors suspendu, avec présentation d'un menu personnalisé de saisie des variables X,Y et Z.

1) si on poursuit par CONT sans donner de valeur à X,Y,Z, le résultat est (en 6 secondes):

```
1: { Y 'X+INV(Z)' '2*Z-Y/Z^2' }
```

c'est l'expression de grad(f) en un point quelconque (X,Y,Z).

2) si on poursuit par CONT en donnant à X,Y les valeurs respectives 1,2 le résultat obtenu est :

```
1: { 2 '1+INV(Z)' '2*Z-2/Z^2' }
```

On obtient ainsi l'expression du gradient de f en tout point de coordonnées (1,2,Z).

Répertoire 'GDIF' Programme 'DELTA'

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: Dès l'appel de 'DELTA', celui-ci est suspendu, avec présentation d'un menu personnalisé de saisie des variables. Si on poursuit immédiatement par CONT, on obtient l'expression symbolique de Δ(f). Si auparavant on donne aux variables une valeur numérique, on obtient la valeur de Δ(f) en un point bien précis.

'DELTA':

```
«   DUP   PURGE
    →  f   var
       «  Ø   1   var   SIZE   FOR   i
             f   1   2   START
                'var'   i   GET   ∂
             NEXT
             +
          NEXT
          → d
          «  { STO }   var   +   MENU   HALT   d   »   EVAL
          var   PURGE   VARS   MENU
       »
»
```

PROGRAMME 'DELTA' : EXEMPLES D'UTILISATION.

Exemple 1 :

On veut calculer le laplacien de f(X,Y)= X^2*SIN(Y).
On crée la pile ci-dessous:

```
2: | 'X^2*SIN(Y)' |
1: | { X Y }      |
```

On appelle ensuite le programme 'DELTA'. Celui-ci est alors suspendu, au bout de quatre secondes, avec présentation d'un menu personnalisé de saisie des variables X,Y.

1) si on poursuit par CONT sans donner de valeur à X,Y, le résultat est :

```
1: | '2*SIN(Y)+X^2*(-SIN(Y))' |
```

(expression du laplacien de f en un point quelconque (X,Y)).

2) si on poursuit par CONT en donnant à X,Y les valeurs respectives 1,2 le résultat obtenu est :

```
1: | .9Ø92974268240 |
```

On obtient ainsi la valeur du laplacien de f au point (1,2).

Exemple 2 :

On veut calculer le laplacien de f(X,Y,Z)= 'XY + Z² + Y/Z'.
On crée la pile ci-dessous:

```
2: | 'X*Y+Z^2+Y/Z' |
1: | { X Y Z }     |
```

On appelle ensuite le programme 'DELTA'. Celui-ci est alors suspendu, au bout de six secondes, avec présentation d'un menu personnalisé de saisie des variables X,Y et Z.

1) si on poursuit par CONT sans donner de valeur à X,Y,Z, le résultat est (en 2 secondes):

```
1: | '2+Y*(2*Z)/Z^2^2' |
```

car l'expression de $\Delta(f)$ en un point quelconque (X,Y,Z) est: 2 + 2*Y/Z^3. (pour le voir utiliser COLCT et EXPAN).

2) si on poursuit par CONT en donnant à Z la valeur 1, on trouve:

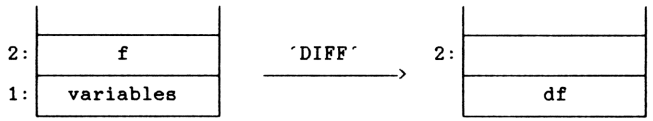
```
1: | '2+Y*2' |
```

ce qui l'expression du laplacien de f en tout point de coordonnées (X, Y, 1).

Répertoire 'GDIF' Programme 'DIFF'

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:

$$df = \frac{\partial f}{\partial X}\,dX + \frac{\partial f}{\partial X}\,dY + \frac{\partial f}{\partial X}\,dZ.$$

N.B: Le programme 'DIFF' appelle le programme 'GRAD'. 'DIFF' est suspendu à un moment donné avec présentation d'un menu personnalisé de saisie des variables. Si on poursuit immédiatement par CONT, on obtient l'expression symbolique de df. Si auparavant on donne aux variables une valeur numérique, on obtient la valeur de df en un point bien précis.

'DIFF':

```
«   →  var
    «  var   GRAD
       →  dp
       «  Ø   1   var   SIZE   FOR   i
              'dp'   i   GET   "d"   'var'   i   GET   →STR
              2   OVER   SIZE   1   -   SUB   +   STR→   *   +
          NEXT
       »
    »
»
```

N.B: Si les variables s'appellent X,Y,Z (par exemple), le programme 'DIFF' utilise dans l'expression de df les quantités 'dX', 'dY', 'dZ'. Pour que 'DIFF' fonctionne correctement, il faut qu'aucune de ces quantités ne soit le nom d'une variable du répertoire.

Répertoire 'GDIF' Programme 'DIFF' (suite)

PROGRAMME 'DIFF' : EXEMPLES D'UTILISATION.

Exemple 1 :

On veut calculer la différentielle de f(X,Y)= X^2*SIN(Y).
On crée la pile ci-dessous:

```
2: 'X^2*SIN(Y)'
1: { X Y }
```

On appelle ensuite le programme 'DIFF'. Celui-ci est alors suspendu, avec présentation d'un menu personnalisé de saisie des variables X,Y.

1) si on poursuit par CONT 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 poursuit par CONT en donnant à X,Y les valeurs respectives 1,2 le résultat obtenu est (en mode 5 FIX):

```
1: '1.81859*dX-Ø.41615*dY'
```

On obtient ainsi la valeur de df au point (1,2).

Exemple 2 :

On veut calculer la différentielle de
f(RO,TETA,FI)= RO^2*SIN(TETA)*EXP(FI*TETA).
On crée la pile ci-dessous:

```
2: 'RO^2*SIN(TETA)*EXP(FI*TETA)'
1: { RO TETA FI }
```

On appelle ensuite le programme 'DIFF'. Celui-ci est alors suspendu, avec présentation d'un menu personnalisé de saisie des variables RO, TETA, et FI.

Si on poursuit par CONT en donnant à RO, TETA, et FI les valeurs respectives 1,2 et 3 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).

GRAPHISME

La HP28S possède un écran graphique de 32*137 points. Un certain nombre d'instructions se rapportent à la gestion de cet écran; on les trouve regroupées dans le menu PLOT.

Les instructions LCD→ et →LCD, que l'on trouve pourtant dans le menu STRING, sont d'une grande importance dans ce chapitre. Le manuel d'utilisation de la HP28 étant muet sur ces instructions, une note leur est consacrée dans les pages suivantes.

L'instruction de traçage de segment est absente du jeu d'instructions de la HP28. Les différents programmes de tracé de courbes du répertoire 'GRAPH' se contenteront donc de tracés "point par point", quitte à ce que ces tracés soient discontinus.
Il aurait été possible d'insérer dans le répertoire 'GRAPH' un programme de tracé de segment. celà se serait cependant traduit par une certaine lourdeur, et une plus grande lenteur dans les tracés. D'ailleurs, les dimensions modestes de l'écran graphique ne rendaient pas indispensable cette routine.

Le répertoire 'GRAPH' contient les programmes suivants:

'**PAR**' : tracé d'une courbe paramétrée (X=X(T), Y=Y(T)).

'**POL**' : tracé d'une courbe en polaires (Ro = Ro(θ)).

'**POLP**' : tracé d'un courbe en polaires paramétrées (Ro = Ro(T)
θ = θ(T)).

'**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)=Ø.

'**FAMT**' : tracé d'une famille de courbes dépendant d'un paramètre T.

'**FAMI**' : tracé d'une famille quelconque de courbes.

'**ANIM**' : production d'images-écran successives.

Répertoire 'GRAPH'

LES INSTRUCTIONS →LCD ET LCD→ .

L'instruction LCD→ tranforme l'image présente à l'écran en une chaîne de 548 caractères et dépose cette chaîne au niveau 1 de la pile. L'instruction →LCD permet de renvoyer cette chaîne (si elle est présente au niveau 1 de la pile) à l'écran, pour reconstituer l'image initiale.

L'écran peut être décomposé en 4 lignes de caractères. La ligne 1 est la ligne supérieure, la ligne 4 est la ligne inférieure (pour reprendre la syntaxe de l'instruction DISP).
Chaque ligne de caractères est une succession de 137 colonnes de 8 pixels. Chacune de ces colonnes de 8 pixels est représentable par un entier compris entre Ø et 255, et cet entier par un caractère. C'est cette représentation qu'utilisent les instructions LCD→ et →LCD. (on trouve bien 137*4=548).

Ces deux intructions constituent donc un moyen pratique de sauvegarder une image d'écran dans une variable, pour la visualiser tout à loisir.

Il est également possible d'appliquer les instructions logiques OR, XOR, AND, NOT à des chaînes de caractères représentant des images d'écran.

Si A et B sont deux telles chaînes, représentant les écrans EA et EB:
« A B OR » (ou: 'A OR B' EVAL) crée la chaine représentant la superposition des écrans EA et EB.
« A NOT » (ou: 'NOT(A)' EVAL) crée la chaîne représentant l'image en vidéo inversée de l'écran EA.
« A B XOR » (ou : 'A XOR B' EVAL) crée la chaine représentant une image où seuls sont allumés les points qui n'étaient allumés que dans un seul des écrans EA et EB.
« A B AND » (ou : 'A AND B' EVAL) crée la chaine représentant une image où seuls sont allumés les points qui étaient allumés dans les deux écrans EA et EB.

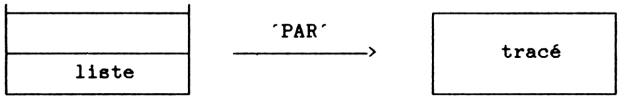
L'inconvénient d'un usage "abusif" de l'instruction LCD→ est évident: à 548 octets la chaîne de caractères produite, la mémoire de la HP28 diminue rapidement.

Répertoire 'GRAPH' programme 'PAR'

TRACE D'UNE COURBE PARAMETREE.

'PAR' permet de tracer point par point 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 { X(T) Y(T) debut fin pas }
où

* X(T), Y(T) sont des expressions algébriques représentant l'abscisse et l'ordonnée 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.

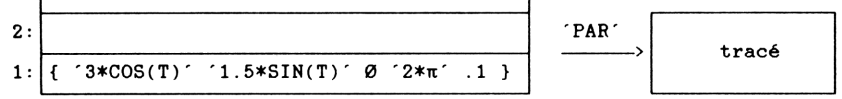
Seuls sont tracés les points correspondant aux valeurs successives de T. En cas d'erreur sur le calcul de X(T) ou de Y(T), on passe au point suivant (le tracé n'est donc pas interrompu).

Si la variable 'PPAR' est absente du répertoire, elle est automatiquement créée avec les valeurs par défaut. les axes de coordonnées sont tracés.

A la fin du tracé, on peut numériser des points à l'aide du réticule (instruction DGTIZ). On sort du programme 'PAR' en appuyant sur la touche 'ON'. On obtient alors au niveau 1 une chaîne de caractères représentant l'image du tracé obtenu (instruction LCD→).

```
'PAR':
    «   LIST→   DROP
        →  x   y   deb   fin   pas
        «  CLLCD   DRAX
           deb  →NUM   fin   →NUM   FOR   i
              i   'T'   STO
              IFERR   x   EVAL   y   EVAL   R→C   PIXEL
                      THEN   CLEAR   END
           pas   STEP
           'T'   PURGE   DGTIZ   LCD→
        »
    »
```

Exemple: tracé de la courbe paramétrée par X(T)=3*COS(T), Y(T)=1.5*SIN(T)
(avec les valeurs par défaut dans PPAR) Ø ≤ T ≤ 2*π, et un pas de Ø.1

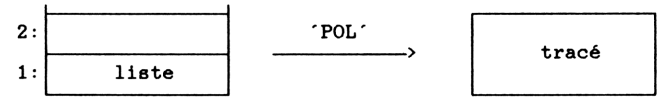


C'est une ellipse. Elle est obtenue en 2Ø secondes.

Répertoire 'GRAPH' programme 'POL'

TRACE D'UNE COURBE EN POLAIRES.

'POL' permet de tracer point par point 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

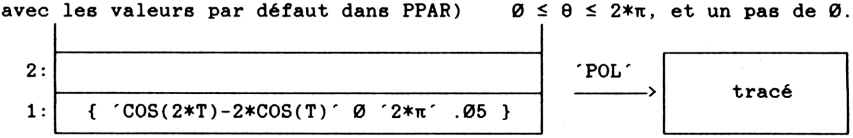
* Ro(T) est l'expression algébrique représentant le rayon-vecteur du point courant, d'angle polaire T (la majuscule T est obligatoire).
* "debut", "fin" représentent respectivement la valeur initiale et la valeur finale prises par l'angle polaire T.
* "pas" est l'incrément de l'angle polaire T. Le temps de tracé est évidemment inversement proportionnel à T.

Seuls sont tracés les points correspondant aux valeurs successives de T. En cas d'erreur sur le calcul de Ro(T), on passe au point suivant (le tracé n'est donc pas interrompu).
Si la variable 'PPAR' est absente du répertoire, elle est automatiquement créée avec les valeurs par défaut. les axes de coordonnées sont tracés.

A la fin du tracé, on peut numériser des points à l'aide du réticule (instruction DGTIZ). On sort du programme 'PAR' en appuyant sur la touche 'ON'. On obtient alors au niveau 1 une chaîne de caractères représentant l'image du tracé obtenu (instruction LCD→).

```
'POL':
    «   LIST→   DROP
        →  r   deb   fin   pas
        «  CLLCD   DRAX
           deb   →NUM   fin   →NUM   FOR   i
              i   'T'   STO
              IFERR   r   EVAL   i   R→C   P→R   PIXEL
                      THEN   CLEAR   END
           pas   STEP
           'T'   PURGE   DGTIZ   LCD→
        »
    »
```

Exemple: tracé de la courbe définie par Ro(θ)= COS(2*θ)-2*COS(θ)
(avec les valeurs par défaut dans PPAR) Ø ≤ θ ≤ 2*π, et un pas de Ø.Ø5



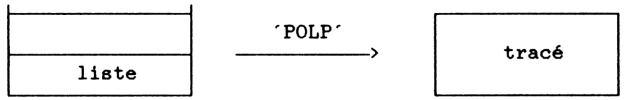
est obtenue en 45 secondes.

Répertoire 'GRAPH' programme 'POLP'

TRACE D'UNE COURBE EN POLAIRES PARAMETREES.

'POLP' permet de tracer point par point une courbe définie par une représentation en polaires paramétrées : Ro= Ro(T), θ=θ(T).

Le schéma fonctionnel est le suivant:



La liste du niveau 1 a le format { Ro(T) θ(T) debut fin pas }
où

* Ro(T) et θ(T) sont les expressions algébriques représentant le rayon-vecteur et l'angle polaire 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.

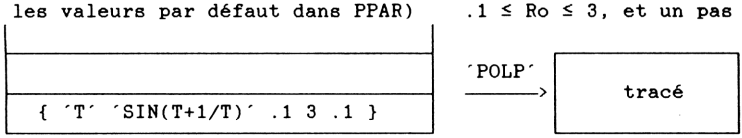
Seuls sont tracés les points correspondant aux valeurs successives de T. En cas d'erreur sur le calcul de Ro(T) ou de θ(T), on passe au point suivant (le tracé n'est donc pas interrompu).

Si la variable 'PPAR' est absente du répertoire, elle est automatiquement créée avec les valeurs par défaut. les axes de coordonnées sont tracés.

A la fin du tracé, on peut numériser des points à l'aide du réticule (instruction DGTIZ). On sort du programme 'PAR' en appuyant sur la touche 'ON'. On obtient alors au niveau 1 une chaîne de caractères représentant l'image du tracé obtenu (instruction LCD→).

```
'POLP':
    «   LIST→  DROP
        →  r   teta   deb   fin   pas
        «  CLLCD   DRAX
           deb  →NUM   fin  →NUM   FOR   i
              i   'T'   STO
              IFERR   r   EVAL   teta   EVAL   R→C   P→R   PIXEL
                      THEN   CLEAR   END
           pas   STEP
           'T'   PURGE   DGTIZ   LCD→
        »
    »
```

Exemple:tracé de la courbe définie par θ=SIN(Ro+1/Ro)
(avec les valeurs par défaut dans PPAR) .1 ≤ Ro ≤ 3, et un pas de Ø.1



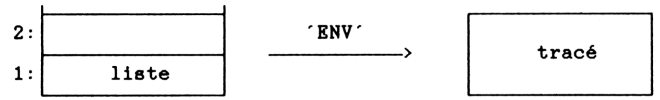
est obtenue en 15 secondes.

Répertoire 'GRAPH' programme 'ENV'

TRACE DE L'ENVELOPPE D'UNE FAMILLE DE DROITES.

'ENV' permet de tracer, point par point, l'enveloppe d'une famille de droites D(T), l'équation de D(T) s'écrivant: A(T)*X + B(T)*Y + C(T) = Ø.

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.

Seuls sont tracés les points correspondant aux valeurs successives de T. En cas d'erreur sur le calcul d'un point, on passe au point suivant (le tracé n'est donc pas interrompu).

Si la variable 'PPAR' est absente du répertoire, elle est automatiquement créée avec les valeurs par défaut. les axes de coordonnées sont tracés.

A la fin du tracé, on peut numériser des points à l'aide du réticule (instruction DGTIZ). On sort du programme 'PAR' en appuyant sur la touche 'ON'. On obtient alors au niveau 1 une chaîne de caractères représentant l'image du tracé obtenu (instruction LCD→).

Le programme 'ENV' appelle le programme 'PAR'.

```
'ENV':
    «   LIST→  DROP
        →  a   b   c   deb   fin   pas
        «  a   'T'  ∂   b   'T'  ∂   c   'T'  ∂
           →  a1   b1   c1
           «  c1   b   *   c   b1   *   -   a   b1   *   a1   b   *   -
              DUP   ROT   ROT   /   SWAP   a1   c   *   a   c1   *   -
              SWAP   /   deb   fin   pas   5   →LIST   PAR
           »
        »
    »
```

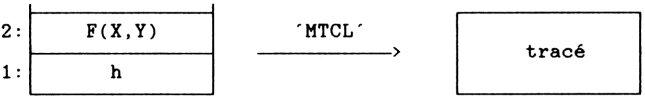
Exemple: Pour tracer l'enveloppe de la famille de droites d'équations: SIN(T)*X+COS(T)*Y-3*SIN(T)*COS(T)=Ø, on place au niveau 1 la liste: { 'SIN(T)' 'COS(T)' '-3*SIN(T)*COS(T)' Ø '2*π' .1 }

Répertoire ´GRAPH´ programme ´MTCL´

METHODE DE MONTE-CARLO
(VISUALISER LES COURBES F(X,Y)=Ø).

On veut visualiser une courbe définie par une équation F(X,Y)=Ø.
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)=Ø, 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).
* h est le "pas" permettant de définir la précision de la méthode.
  le pas h doit être un entier compris entre 1 et 9.
  h=1 signifie que tous les pixels sont passés en revue.
  h=2 signifie qu´une ligne et une colonne sur deux sont concernées.
  etc...

En cours de tracé, il est possible de définir un nouveau "pas" h, en appuyant sur une des touches "1", "2",..,"9" (touche lue "au vol").

```
´MTCL´:
    «   →   f   h
        «   CLLCD   DRAX   PPAR   2   GET   PPAR   1   GET   DUP2
            -   C→R   32   /   SWAP   137   /
            →   pmax   pmin   pasy   pasx
            «   pmin   RE   pmax   RE   FOR   i   i   ´X´   STO
                   pmin   IM   pmax   IM   FOR   j   j   ´Y´   STO
                      f   KVAL
                      IF   Ø   >   THEN   i   j   R→C   PIXEL   END
                      IF   KEY   THEN
                         NUM   48   -   1   MAX   9   MIN   ´h´   STO
                      END
                   pasy   h   *   STEP
                pasx   h   *   STEP
                DGTIZ   LCD→
            »
        »
    »
```

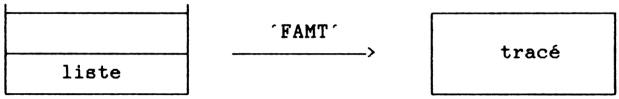
Remarque: à la fin du tracé, on peut numériser des points à l´aide du réticule (instruction DGTIZ). On sort de ´MTCL´ par un appui sur ´ON´. On trouve alors au niveau 1 une chaîne de caractères représentant l´image de l´écran obtenu.

Répertoire 'GRAPH' programme 'FAMT'

TRACE D'UNE FAMILLE DE COURBES DEPENDANT D'UN PARAMETRE.

'FAMT' permet de tracer les différentes courbes représentatives d'une famille de fonctions F dépendant d'un paramètre T.

Le schéma fonctionnel est le suivant:



la liste au niveau 1 ayant le format: { F(X,T) debut fin pas }, où:

* F(X,T) est l'expression algébrique donnant la fonction F: X est la variable et T le paramètre (les majuscules sont obligatoires).
* "debut", "fin" représentent respectivement la valeur initiale et la valeur finale prises par le paramètre T.
* "pas" est l'incrément de le paramètre T.

On appelle alors 'FAMT':

Première possibilité:

Chaque courbe est tracée succesivement à l'écran (avec effacement de l'écran entre deux tracés).
A la sortie du programme, une liste est renvoyée au niveau 1, dont le contenu est formé des chaînes représentant chacun des tracés.

Deuxième possibilité:

Avant d'appeler 'FAMT', on place l'entier 1 au niveau 1 de la pile, la liste des données remontant donc au niveau 2.
Toutes les courbes sont alors tracées successivement, mais sur un même écran. A la sortie, une seule chaîne de caractères (représentant l'image de l'écran obtenu) est renvoyée au niveau 1.

```
'FAMT':
    «   IF   DUP   1   ≠   THEN   Ø   END
        →  flag
        «  LIST→   DROP
           →   deb   fin   pas
           «   STEQ   IF   flag   NOT   THEN   { }   END
               CLLCD
               deb   fin   FOR   i
                  i   'T'   STO   DRAW
                  IF   flag   NOT   THEN   LCD→   +   CLLCD   END
               pas   STEP
               {   EQ   T   }   PURGE
               IF   flag   THEN   LCD→   END
               CLMF
           »
        »
    »
```

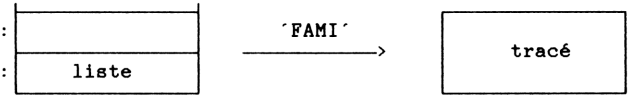
Exemple: avec les paramètres de traçage par défaut, essayer
{ 'SIN(X*T)*T' .5 1.5 .2 }.

Répertoire 'GRAPH' programme 'FAMI'

TRACE D'UNE FAMILLE DE COURBES

'FAMI' permet de tracer les différents éléments d'une famille quelconque de courbes.

Le schéma fonctionnel est le suivant:



La liste présente au niveau 1 doit être constituée d'expressions algébriques (ou de programmes, à condition qu'ils respectent la syntaxe des expressions algébriques: ne pas prendre d'objet sur la pile et y déposer un objet au niveau 1).

Exemple:

```
liste= { 'COS(X)'   « X DUP SIN MIN »  }
```

Première possibilité:

Chaque expression de la liste est tracée successivement à l'écran (avec effacement de l'écran entre deux tracés).

A la sortie du programme, une liste est renvoyée au niveau 1, dont le contenu est formé des chaînes représentant chacun des tracés.

Deuxième possibilité:

Avant d'appeler 'FAMI', on place l'entier 1 au niveau 1 de la pile, la liste des expressions à tracer remontant donc au niveau 2.

Toutes les expressions sont alors tracées successivement, mais sur un même écran. A la sortie, une seule chaîne de caractères (représentant l'image de l'écran obtenu) est déposée au niveau 1.

```
'FAMT':
    «  IF  DUP  1  ≠  THEN  Ø  END
       →  liste  flag
       «  CLLCD
          IF  flag  NOT  THEN  { }  END
          1  liste  SIZE  FOR  i
             'liste'  i  GET  STEQ  DRAW
             IF  flag  NOT  THEN  LCD→  +  CLLCD  END
          NEXT
          'EQ'  PURGE
          IF  flag  THEN  LCD→  END
          CLMF
       »
    »
```

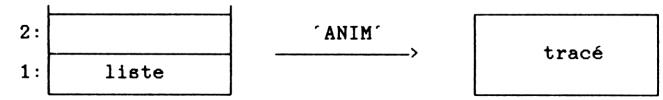
Répertoire 'GRAPH' programme 'ANIM'

PRODUCTION D'IMAGES-ECRAN SUCCESSIVES.

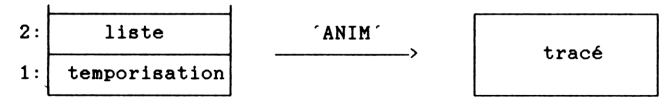
'ANIM' permet de visualiser successivement différentes images-écran .
Ces différentes images doivent être représentées par des chaînes de caractères (syntaxe de l'instruction LCD→). Ces chaînes de caractères doivent être regroupées dans une liste.

Le schéma fonctionnel est le suivant:

Premier cas:



Deuxième cas:



La liste dont il est question ici doit être constituée de chaînes de caractères crées au moyen de l'instruction LCD→.Si la liste en question 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, représentant la durée, exprimée en secondes, entre deux affichages successifs. Le défilement des images à l'écran est alors automatique. On placera Ø au niveau 1 pour obtenir un défilement à la vitesse maximum(7 images par seconde).

```
'ANIM':
    ≪   IF   DUP   TYPE   THEN   -1   END
        →  t
        ≪   { 1 }
            DO
               GETI   →LCD
               IF   t   Ø   ≥   THEN   t   WAIT
               ELSE
                  DO   UNTIL   KEY   END   DROP
               END
            UNTIL   Ø   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 = 870013200006789990000 1.
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 deux entiers longs (calcul du quotient et du reste entiers).
'**PGCDL**' : 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 nombre réel à la forme entier long.
'**L→R**' : passage de la forme entier long à la forme nombre 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:

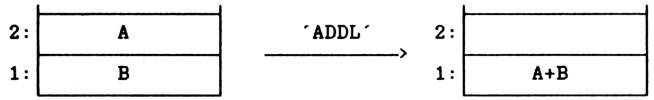
'**FRMT**' : gestion des débordements dans les calculs sur les entiers longs.
'**ELMG**' : dans un entier long, élimination des éventuels coefficients nuls à gauche.

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'.

Répertoire 'LONG' **Programmes 'ADDL' et 'PRODL'**

ADDITION DE DEUX ENTIERS LONGS.

'ADDL' permet d'additionner deux entiers longs A et B, suivant le schéma fonctionnel:

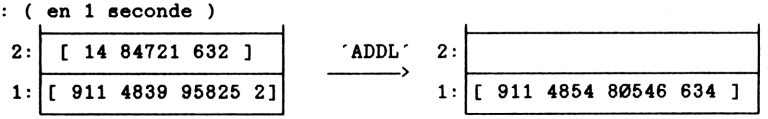


Le résultat est obtenu dans le format 'entier long'. Le programme 'ADDL' passe dans le répertoire 'POLY' ou il appelle le programme 'ADDP'. Il revient dans le répertoire 'LONG', où il appelle le programme 'FRMT' (gestion des débordements).

'ADDL':

« **POLY ADDP LONG FRMT** »

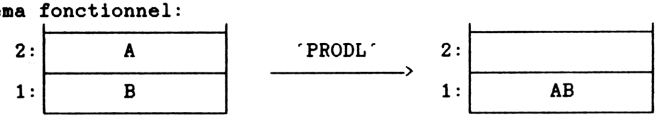
Exemple: (en 1 seconde)



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:

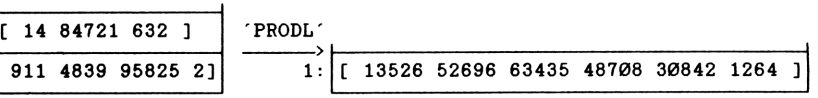


Le résultat est obtenu dans le format 'entier long'. Le programme 'PRODL' passe dans le répertoire 'POLY' ou il appelle le programme 'PRODP'. Il revient dans le répertoire 'LONG', où il appelle le programme 'FRMT' (gestion des débordements).

'PRODL':

« **POLY PRODP LONG FRMT** »

Exemple: (en 2 à 3 secondes)



car 14 84721 00632 * 911 04839 95825 00002 est égal à:
13526 52696 63435 48708 30842 01264.

Répertoire 'LONG' Programme 'EXPOL'

ELEVATION D'UN ENTIER LONG A UNE PUISSANCE ENTIERE.

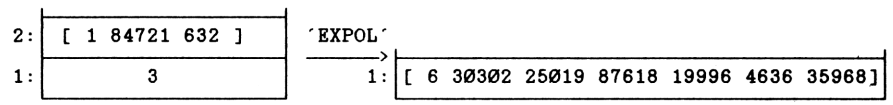
'EXPOL' permet de calculer A ,où A est un entier long, et où n est un entier naturel. Le schéma fonctionnel est:

N.B: 'EXPOL' appelle le programme 'PRODL'.

'EXPOL':

```
« → a n
   « [ 1 ]
     WHILE n Ø >
     REPEAT
        IF n 2 MOD THEN
           a PRODL
        END
        n 2 / FLOOR 'n' STO
        IF n Ø > THEN
           a DUP PRODL 'a' STO
        END
     END
  »
»
```

Exemple: (en 5 secondes)

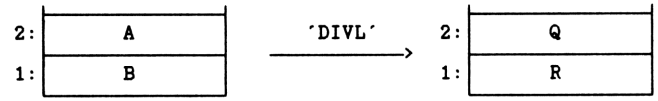


car (1 84721 ØØ632)^3 = 6 3Ø3Ø2 25Ø19 87618 19996 Ø4636 35968.

Répertoire 'LONG' Programme 'DIVL'

DIVISION DE DEUX ENTIERS LONGS.

'DIVL' permet de diviser deux entiers longs A et B, suivant le schéma fonctionnel:

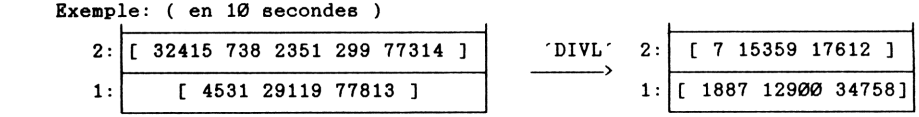


où Q et R sont respectivement le quotient et le reste entiers dans cette division (A = BQ + R avec R < B), tous les deux obtenus dans le format entier long.

N.B: 'DIVL' appelle les programmes 'ELMG', 'FRMT' et 'ADDL'.

```
'DIVL':
    «  ELMG  DUP  SIZE  1  GET  DUP  1  >  ROT  { 1 }  GETI
       SWAP  DROP  ROT  +  4  ROLL  ELMG  DUP  SIZE  1  GET
       SWAP  { 1 }  GETI  SWAP  DROP  [ Ø ]
       →  lb  b  b1  la  a  a1  q
       «  a
          WHILE
            IF  la  lb  >  THEN  IF  a1  b1  <  THEN
                  DUP  {2}  GET  a1  1ØØØØØ  *  +  DUP  'a1' STO
                  {2}  SWAP  PUT  ARRY→  1  GET  ROLL  DROP  la
                  1  -  DUP  'la'  STO  1  →LIST  →ARRY
                END  1
              ELSE  IF  la  lb  ==  THEN  DUP  b
                      IF  DUP2  ==  THEN  DROP2  1
                        ELSE  -  { 1 }
                          WHILE  GETI  DUP  Ø  ==  REPEAT  DROP  END
                          3  ROLLD  DROP2  Ø  >
                        END
                      ELSE  Ø  END
            END
            REPEAT
              IF  a1  b1  1  -  ==  la  lb  ==  AND  THEN
                  la  1  -  'la'  STO  b  -  FRMT  ELMG  [ 1 ]  SWAP
                ELSE
                  a1  DUP  b1  MOD  -  b1  /  DUP  1  →ARRY
                  la  lb  -  1  +  1  →LIST  RDM  3  ROLLD
                  b  *  la  1  →LIST  RDM  -  FRMT  { 1 }
                  GETI  'a1'  STO  DROP
              END
              SWAP  q  ADDL  'q'  STO
            END
            q  SWAP
       »
    »
```

Exemple: (en 10 secondes)

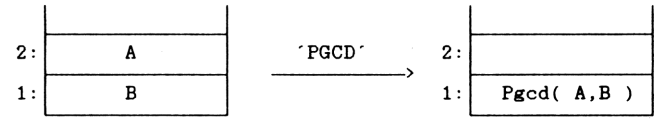


car 32415 00738 02351 00299 77314 est égal à
(4531 29119 77813) * (7 15359 17612) + 1887 12900 34758.

Répertoire 'LONG' Programme 'PGCDL'

PGCD DE DEUX ENTIERS LONGS.

'PGCDL' calcule le pgcd (plus grand commun diviseur) de deux entiers longs A et B. Le schéma fonctionnel est:



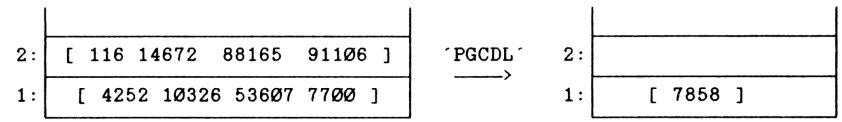
Le Pgcd est obtenu dans le format 'entier long'.

N.B: 'PGCDL' 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).

'PGCDL':

```
«    IF   DUP   [ Ø ]   ==   THEN
          DROP
     ELSE
          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   ROLLD   DIVL   SWAP   DROP   PGCDL
          END
     END
»
```

Exemple: (en 44 secondes)

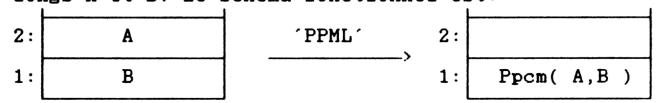


car le Pgcd de 116 14672 88165 91106 et de 4252 10326 53607 07700 est égal à 7858.

Répertoire 'LONG' Programmes 'PPML' et 'FACTL'

PPCM DE DEUX ENTIERS LONGS.

'PPML' calcule le ppcm (plus petit commun multiple) de deux entiers longs A et B. Le schéma fonctionnel est:

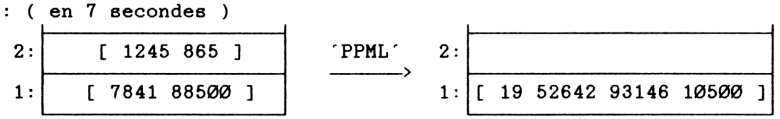


Le Ppcm est obtenu dans le format 'entier long'.
N.B: 'PPMP' appelle les programmes 'PGCDL', 'DIVL' et 'PRODL'.

'PPML':

```
«   DUP2
    →  a  b
    «  PGCDL  b  SWAP  DIVL  DROP  a  PRODL
    »
»
```

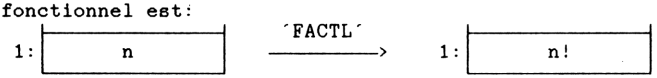
Exemple: (en 7 secondes)



car ppcm (1245 00865, 7841 88500) = 19 52642 93146 10500.

FONCTION FACTORIELLE.

'FACTL' calcule la factorielle n! d'un entier naturel n. Le schéma fonctionnel est:

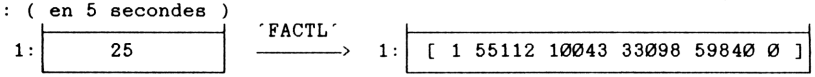


n doit être un entier naturel. Le résultat est obtenu dans le format 'entier long'.
On peut vérifier que la fonction FACT de la HP28 ne fournit tous les chiffres de n! que jusqu'à n=14 (et alors n! = 87178291200). Le rôle de 'FACTL' est d'obtenir tous les chiffres significatifs de n! quand n>14.

'FACTL':

```
«   →  n
    «  IF  n  14  ≤  THEN  n  FACT  1  →ARRY  FRMT
       ELSE  n  n  1  -  FACTL  *  FRMT  END
    »
»
```

Exemple: (en 5 secondes)



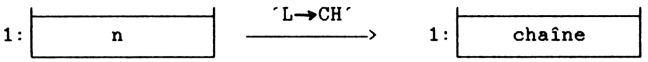
car 25! = 1 55112 10043 33098 59840 00000.

Répertoire 'LONG' Programme 'L→CH'

TRANSFORMATION D'UN ENTIER LONG EN CHAINE DE CARACTERES.

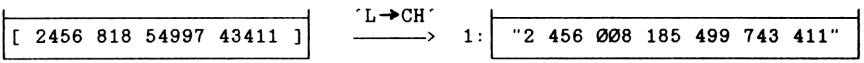
'L→CH' transforme un entier long n en une chaîne de caractères représentant n, avec séparation par un espace tous les trois chiffres.

Le schéma fonctionnel est:




```
'L→CH':
    «   DUP   SIZE   1   GET   ""
        →  n   ch
        «  { 1 }
           WHILE   GETI   DUP   Ø   ==   3   PICK   { 1 }  ≠   AND
              REPEAT
                 DROP
              END
           →STR   'ch'   STO
           WHILE   DUP   { 1 }   ≠
              REPEAT
                 GETI   →STR
                 WHILE   DUP   SIZE   5   <
                    REPEAT
                       "Ø"   SWAP   +
                    END
                 ch   SWAP    +   'ch'   STO
              END
           DROP2  ch  SIZE  DUP  DUP  3  MOD  DUP  4  ROLLD  -  3  /
           IF   DUP   THEN
                +   3   -   FOR   i
                   ch   1   i   SUB   " "   +   ch   i   1
                   +   ch   SIZE   SUB   +   'ch'   STO
                4   STEP
              ELSE
                3   DROPN
           END
           ch
        »
    »
```

Exemple: (moins de 2 secondes)

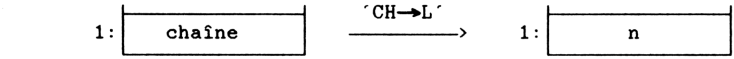


Répertoire 'LONG' Programme 'CH→L'

TRANSFORMATION D'UNE CHAINE DE CARACTERES EN ENTIER LONG.

'CH→L' transforme une chaîne de caractères, exclusivement composée de chiffres ou d'espaces, en l'entier long n correspondant.

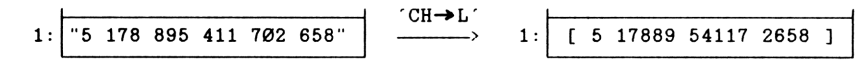
Le schéma fonctionnel est:



Le programme s'arrête sur un message d'erreur si aucun chiffre n'est trouvé, ou si un caractère non autorisé est rencontré.

```
'CH→L':
    «    ""   Ø   Ø   Ø
         →  ch   ch2   t   t2   c
         «  WHILE   ch   SIZE   Ø   >
            REPEAT
               "Ø123456789 "   ch   DUP   SIZE   DUP   SUB   POS
               IF   DUP   Ø   ==   THEN
                     DROP   ""   'ch'   STO   -1   't2'   STO
                  ELSE
                     IF   DUP   11   <   THEN
                           1   -   1Ø   c   ^   *   t2   +   't2'   STO
                           c   1   +   'c'   STO
                           IF   c   5   ==   THEN
                              t   1   +   't'   STO   t2   t   ROLLD
                              Ø   't2'   STO   Ø   'c'   STO
                           END
                        ELSE
                           DROP
                     END
                     ch  DUP  SIZE   1   -   1  SWAP  SUB  'ch'  STO
               END
            END
         IF   t2   -1   ==   THEN
               "Erreur"
            ELSE
               IF   t2   Ø   >   THEN
                  t   1   +   't'   STO   t2   t   ROLLD
               END
               t   1   →LIST   →ARRY
         END
      »
   »
```

Exemple:

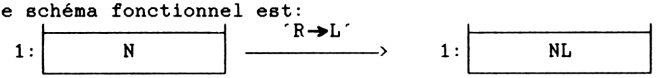


Répertoire 'LONG' Programmes 'R→L' et 'L→R'

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'.

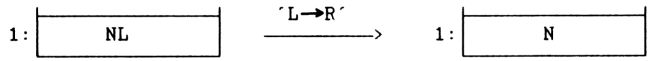
'R→L':

```
«  .5  +  FLOOR  1  →ARRY  FRMT  »
```

Exemples: 1 est tranformé en [1].
71875488654 est transformé en [7 18754 88654].
125487.7 est transformé en [1 25488].
1.49865E3Ø est transformé en [1 49865 Ø Ø Ø Ø Ø].

TRANSFORMATION D'UN ENTIER LONG EN REEL.

'L→R' transforme un entier long NL en son équivalent réel N. Le schéma fonctionnel est:

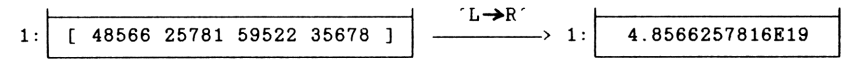


Evidemment, si NL représente un entier supérieur à 1E12, le passage au format réel s'accompagne d'une perte de précision. Néanmoins le programme 'L→R' est utile pour connaître rapidement l'ordre de grandeur d'un entier représenté sous le format 'entier long'.

'L→R':

```
«  DUP  SIZE  1  GET  Ø
   →  n  x
   «  { 1 }
      1  n  START
         GETI  x  1ØØØØØ  *  +  'x'  STO
      NEXT
      DROP2  x
   »
»
```

Exemple: (une seconde)

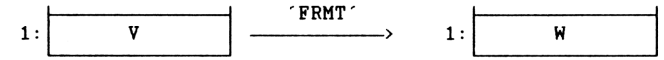


Répertoire 'LONG' Programmes 'FRMT' et 'ELMG'

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':
    «   0   SWAP   ARRY→   1   GET
        →   n
        «   1   n   START
                DUP   100000   MOD   DUP   n   3   +   ROLLD   -   100000   /   +
            NEXT
            WHILE   DUP   0   >
               REPEAT
                  DUP   100000   MOD   DUP   n   1   +   'n'
                  STO   n   2   +   ROLLD   -   100000   /
               END
            DROP   n   →ARRY
        »
    »
```

Exemples:

[106622 110050 129366] est transformé en [1 6623 10051 29366].

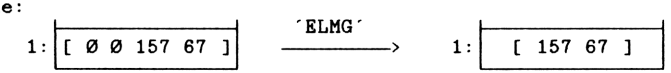
[13902 -35682 -21383] est transformé en [13901 64317 78617].

ELIMINATION DES ZEROS A GAUCHE.

'ELMG' élimine tous les coefficients nuls qui pourraient débuter 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':
    «   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 ´R→Q´ du répertoire ´R.C´ (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.
´HNAP´	: 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.
´NORM´	: fonction de répartition de la loi normale.
´N.C.R´	: fonction de répartition de la loi normale centrée réduite.

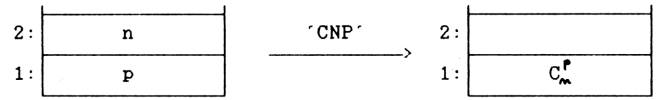
Répertoire 'PROBA' programmes 'CNP'
'ANP'

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 $\frac{n!}{p!(n-p)!}$.

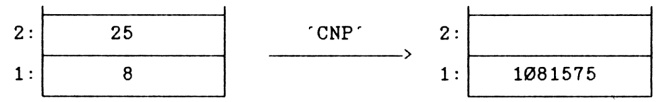
'CNP' ne fait que reprendre COMB, mais est à la fois plus accessible et plus explicite. D'autre part le résultat de 'CNP' est Ø dans le cas où p n'est pas compris entre Ø et n.

Le schéma fonctionnel est:



'CNP': « IFERR COMB THEN Ø END »

Exemple:

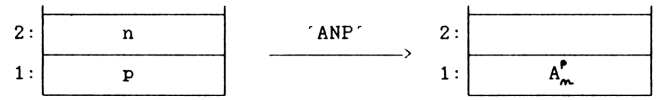


ARRANGEMENTS.

'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 $\frac{n!}{(n-p)!}$.

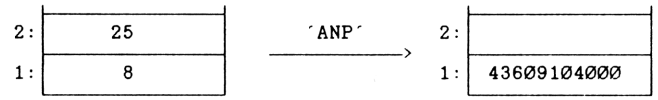
'ANP' ne fait que reprendre PERM, mais est à la fois plus accessible et plus explicite. D'autre part le résultat de 'ANP' est Ø dans le cas où p n'est pas compris entre Ø et n.

Le schéma fonctionnel est:



'ANP': « IFERR PERM THEN Ø END »

Exemple:

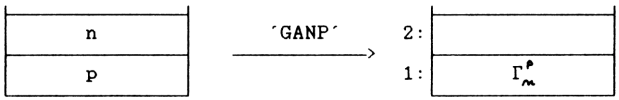


Répertoire 'PROBA' programmes 'GANP'
'BINO'

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_n^p
et vaut $\Gamma_n^p = C_{n+p-1}^{p-1}$. 'GANP' appelle le programme 'CNP'.

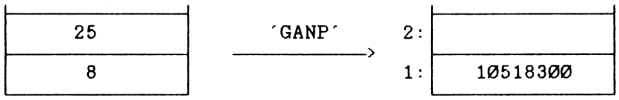
Le schéma fonctionnel est:



'GANP':

```
« DUP ROT + 1 - SWAP CNP »
```

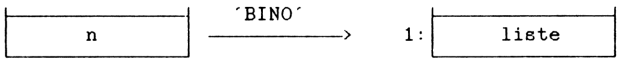
Exemple:



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_n^k avec $0 \le k \le n$. (la liste est ordonnée suivant les k croissants).

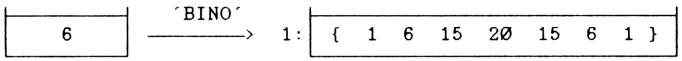
Le schéma fonctionnel est:



'BINO':

```
« → n
  « Ø n FOR i n i COMB NEXT
    n 1 + →LIST
  »
»
```

Exemple:



Répertoire 'PROBA' programme 'BNP'

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 binômiale de paramètres n,p.

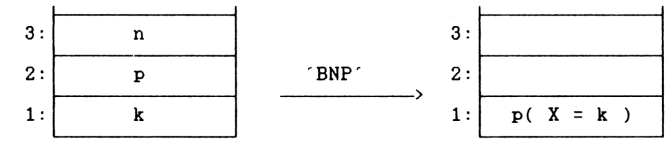
Autrement dit, p(X=k) est la probabilité de voir se réaliser k succès dans une suite de n épreuves indépendantes, la probabilité d'un succès à l'une quelconque de ces épreuves étant égale à p.

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:

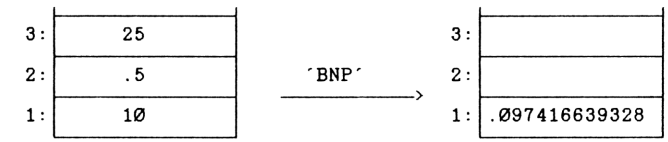


N.B: 'BNP' appelle le programme 'CNP'.

'BNP':

```
«  →  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).

Répertoire 'PROBA' programme 'FBNP'

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 binômiale de paramètres n,p. On obtient ainsi ce qu'on appelle la fonction de répartition de X.

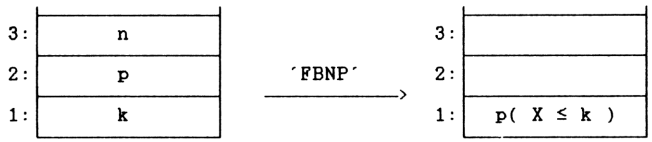
Autrement dit, $p(X \leq k)$ est la probabilité de voir se réaliser au plus k succès dans une suite de n épreuves indépendantes, la probabilité d'un succès à l'une quelconque de ces épreuves étant égale à p.

Nécessairement k,n sont entiers avec $0 \leq k \leq n$, et $0 \leq p \leq 1$.

La formule utilisée est:

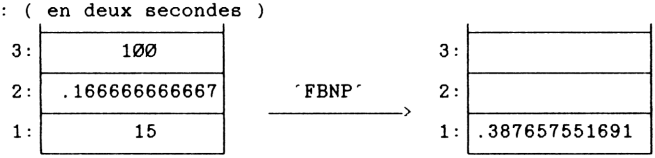
$$p(X \leq k) = \sum_{i=0}^{i=k} C_n^i \; p^i \; (1-p)^{(n-i)}.$$

Le schéma fonctionnel est:




```
'FBNP':
   «   DUP   0   ≥   1
       →  n   p   k   f   a
          «   IF   n   2   /   k   <   THEN
                   n  '1-p'  EVAL  'n-k-1'  EVAL  FBNP  NEG  1  +
              ELSE
                   IF   k   1   ≥   THEN
                      1   k   n   MIN   FOR   i
                         n  i   -   1   +   p   *   i   /   1   p   -
                         /   a   *   'a'   STO   f   a   +   'f'   STO
                      NEXT
                   END
                   1   p   -   n   ^   f   *
              END
          »
   »
```

Exemple: (en deux 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.

Répertoire 'PROBA' programme 'HNAP'

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.

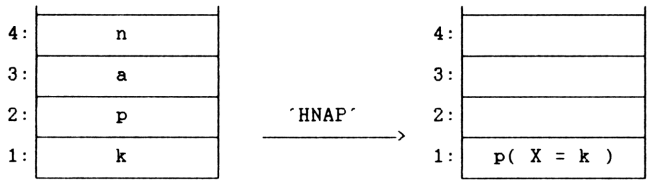
Pour prendre un exemple, p(X=k) est la probabilité d'obtenir, lors d'un tirage simultané de a individus différents dans une population totale d'effectif n , k individus ayant une propriété P donnée, si on suppose que la proportion d'individus ayant cette propriété dans la population entière est égale 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:

$$p(X=k) = \frac{C_{np}^{k} \; C_{n(1-p)}^{a-k}}{C_{n}^{a}}.$$

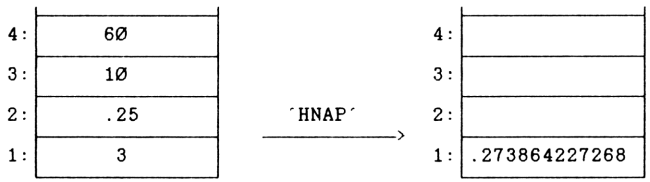
Le schéma fonctionnel est:



'HNAP':

```
« → n a p k
   « 'a-n*(1-p)' EVAL Ø MAX a n p * MIN
      → mink maxk
      « IF k mink < k maxk > OR THEN Ø
         ELSE
            n p * k COMB
            n 1 p - * a k - COMB *
            n a COMB /
         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 1Ø boules dans une urne qui en contient 6Ø , dont 15 blanches (la proportion 15/6Ø = .25 est au niveau 2 de la pile).

Répertoire 'PROBA' programme 'FHNAP'

FONCTION DE REPARTITION DE LA LOI HYPERGEOMETRIQUE H(n,a,p).

'FHNAP' 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. On obtient ainsi ce qu'on appelle la fonction de répartition de X.

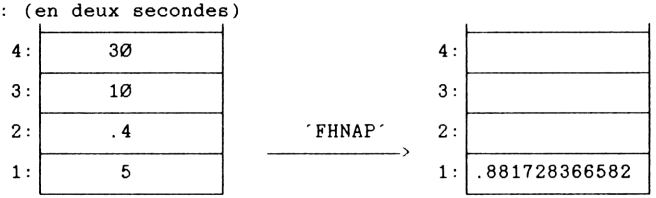
Pour prendre un exemple, p(X≤k) est la probabilité d'obtenir, lors d'un tirage simultané de a individus différents dans une population totale d'effectif n , au plus k individus ayant une propriété P donnée, si on suppose que la proportion d'individus ayant cette propriété dans la population entière est égale à p.

Le schéma fonctionnel est:

'FHNAP':

```
«   DUP   Ø   ≥   1
→   n   a   p   k   f   x
«   'a-n*(1-p)'   EVAL   Ø   MAX   a   n   p   *   MIN
    → mink   maxk
    «   IF   k   mink   >   THEN
            IF   k   maxk   ≥   THEN   1   ABORT
            ELSE
               mink   1   +   k   maxk   MIN   FOR   i
                  '(n*p-i+1)*(a-i+1)/(n-n*p-a+i)/i'   EVAL
                  x   *   'x'   STO   f   x   +   'f'   STO
               NEXT
            END
        END
        n   a   p   mink   HNAP   f   *
    »
»
»
```

Exemple: (en deux secondes)



autrement dit,

si on effectue un tirage exhaustif de 1Ø boules dans une urne qui en contient 3Ø (dont 12 blanches, la proportion de boules blanches étant .4), la probabilité d'en tirer au plus 5 est environ égale à Ø,88.

Répertoire 'PROBA' programme 'POIS'

LOI DE POISSON P(L).

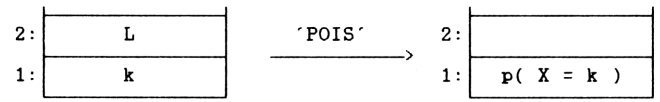
'POIS' calcule la probabilité p(X=k), où X est une v.a.r. (variable aléatoire réelle) suivant la loi de Poisson de paramètre L. L est un réel strictement positif et k est un entier naturel.

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.

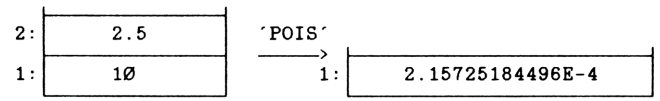
Le schéma fonctionnel est:



'POIS':

```
«   →  L  k
    «  L  NEG  EXP  L  k  ^  *  k  FACT  /
    »
»
```

Exemple:



Répertoire 'PROBA' programme 'FPOIS'

FONCTION DE REPARTITION DE LA LOI DE POISSON P(L).

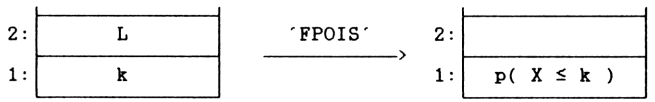
'FPOIS' calcule la probabilité p(X ≤ k), où X est une v.a.r. (variable aléatoire réelle) suivant la loi de Poisson de paramètre L. L est un réel strictement positif et k est un entier naturel.

la formule utilisée est:

$$p(X \leq k) = \exp(-L) \sum_{i=0}^{i=k} \frac{L^i}{i!}$$

Les lois de Poisson permettent de modéliser des problèmes de trafic et de files d'attente.

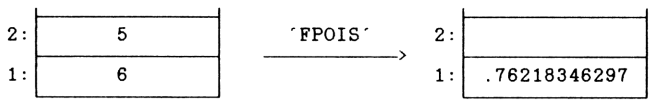
Le schéma fonctionnel est:



'FPOIS':

```
«    DUP   Ø   ≥   1
         L   k   f   a
         «   IF   k   1   ≥   THEN
                1   k   FOR   j
                   a   L   *   j   /   'a'   STO
                   f   a   +   'f'   STO
                NEXT
             END
             L   NEG   EXP   f   *
         »
»
```

Exemple: (en moins d'une seconde)



Répertoire 'PROBA' programme 'GEO'

LOI GEOMETRIQUE G(p).

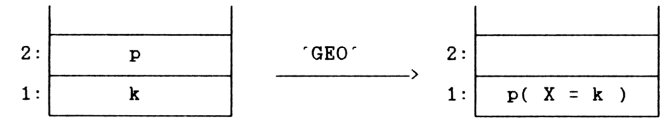
'GEO' calcule la probabilité p(X=k), où X est une v.a.r. (variable aléatoire réelle) suivant la loi géométrique de paramètre p. p est un réel compris entre Ø et 1, et k est un entier strictement positif.

la formule utilisée est:

$$p(X=k)= p*(1-p)^{(k-1)}.$$

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.

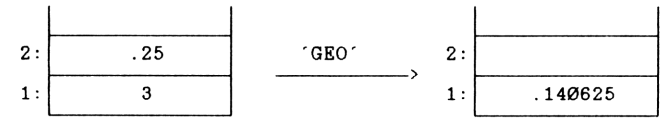
Le schéma fonctionnel est:



'GEO':

```
«  →  p  k
    «  IF  k  Ø  >  THEN  'p*(1-p)^(k-1)'  EVAL  ELSE  Ø  END
    »
»
```

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 Ø,14Ø625 (la probabilité de succès à chaque tirage est égale à .25).

Répertoire ´**PROBA**´ programme ´**FGEO**´

FONCTION DE REPARTITION DE LA LOI GEOMETRIQUE G(p).

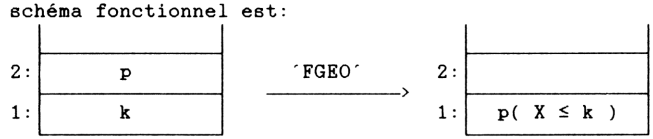
´FGEO´ calcule la probabilité p(X≤k), où X est une v.a.r. (variable aléatoire réelle) suivant la loi géométrique de paramètre p. p est un réel compris entre Ø et 1, et k est un entier strictement positif.

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´:

```
«  →  p  k
   «  IF  k  Ø  ≤  THEN  Ø  ELSE  ´1-(1-p)^k´  EVAL  END
   »
»
```

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 Ø,9375.

Répertoire 'PROBA' programme 'PRP'

LOI DE PASCAL P(r,p).

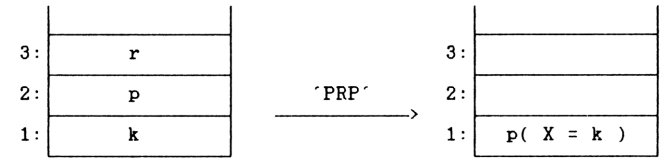
'PRP' calcule la probabilité p(X=k), où X est une v.a.r. (variable aléatoire réelle) suivant la loi de Pascal de paramètres r,p. p est un réel compris entre Ø et 1, r et k sont deux entiers ($1 \leq r \leq k$).

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:

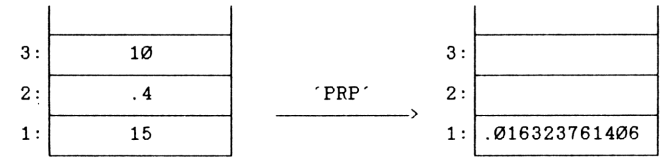


N.B: le programme 'PRP' appelle le programme 'CNP'.

'PRP':

```
«   →   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 4Ø% de boules balanches, la probabilité pour que la 1Øème boule blanche apparaisse au 15ème tirage est donc Ø,Ø16323761406.

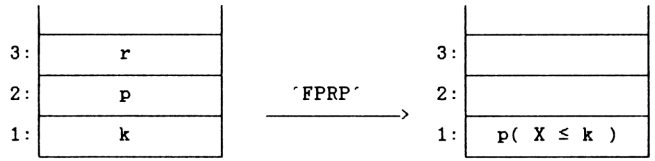
Répertoire 'PROBA' programme 'FPRP'

FONCTION DE REPARTITION DE LA LOI DE PASCAL P(r,p).

'FPRP' 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 Ø et 1, r et k sont deux entiers (1 ≤ r ≤ 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 tentaive.

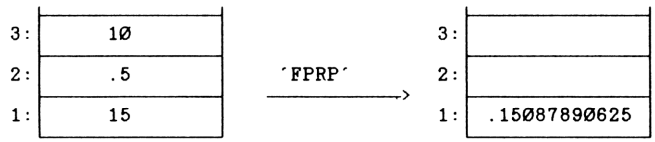
Le schéma fonctionnel est le suivant:



'FPRP':

```
« → r p k
  « IF k r < THEN Ø
    ELSE
      p r ^
      IF k r > THEN 1 1
      → f a
      « r 1 + k FOR i
          'a*(1-p)*(i-1)/(i-r)' EVAL DUP
          'a' STO f + 'f' STO
        NEXT f *
      »
      END
    END
  »
»
```

Exemple:



Si on jette plusieurs fois de suite une pièce équilibrée, la probabilité pour que le 1Ø-ème résultat "pile" intervienne au plus tard à la 15ème tentative est donc Ø,15Ø87890625.

Répertoire 'PROBA' programme 'JRP'

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 Ø et 1, r et k sont deux entiers (1 ≤ r, Ø ≤ k).

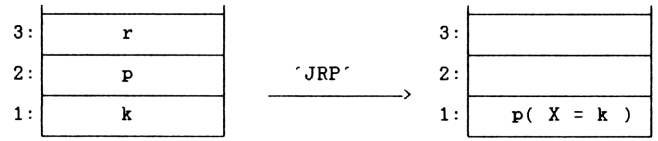
la formule utilisée est:

$$p(X=k)= C^{k}_{k+r-1}\ p^r*(1-p)^k.$$

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 cés 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:

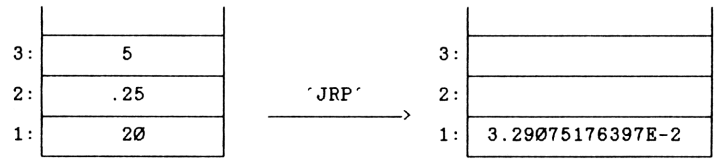


N.B: le programme 'JRP' appelle le programme 'PRP'.

'JRP':

« 3 PICK + PRP »

Exemple:



Si on effectue des tirages avec remise d'une carte dans un jeu normal, la probabilité de tirer exactement 2Ø cartes qui ne soient pas des trèfles avant de tirer le cinquième trèfle est donc environ égale à Ø,Ø329.

Répertoire 'PROBA' programme 'FJRP'

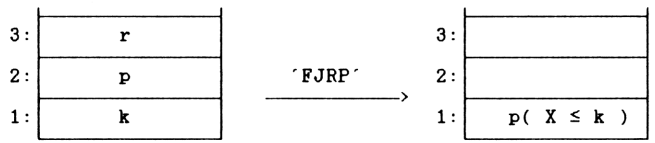
FONCTION DE REPARTITION DE LA LOI BINOMIALE NEGATIVE J(r,p).

'FJRP' 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 Ø et 1, r et k sont deux entiers (1 ≤ r, Ø ≤ 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:

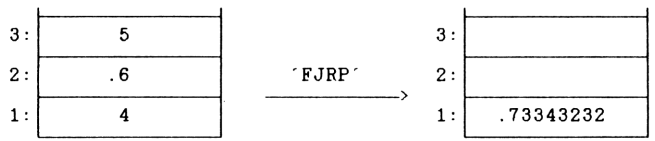


N.B: le programme 'FJRP' appelle le programme 'FPRP'.

'FJRP':

« 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 Ø,73343232.

Répertoire 'PROBA' programmes 'FEXP'

FONCTION DE REPARTITION DE LA LOI EXPONENTIELLE.

'FEXP' donne la probabilité p(X≤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≤x) = Ø si x≤Ø.

et, si x ≥ Ø: $p(X \leq x) = \int_{\emptyset}^{x} L \exp(-Lt)\, dt = 1 - \exp(-Lx)$

Le schéma fonctionnel est:

'FEXP':

```
«    →  L   x
     «  IF   x   Ø   ≥   THEN
            '1-EXP(-L*x)'   EVAL   ELSE   Ø   END
        »
»
```

Exemple:

Répertoire 'PROBA' programmes 'NORM' et 'N.C.R'

FONCTION DE REPARTITION DE LA LOI NORMALE N (m, σ).

'NORM' calcule la probabilité p(X ≤ 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, σ > Ø). x est un réel quelconque.

la formule utilisée est:

$$p(X \leq x) = \frac{1}{\sigma\sqrt{(2\pi)}} \int_{-\infty}^{x} \exp(-(t-m)^2/(2\sigma^2))\, dt$$

Les lois normales permettent de modéliser des problèmes liés à l'observation de populations d'effectif important. Le schéma fonctionnel est:

3:	m			3:	
2:	σ		'NORM'	2:	
1:	x		⟶	1:	p(X ≤ x)

'NORM': « SWAP SQ SWAP UTPN NEG 1 + »

Exemple:

3:	4.5			3:	
2:	.4		'NORM'	2:	
1:	5		⟶	1:	.894350226333

On obtient ainsi la probabilité p(X≤5), où X suit la loi normale d'espérance m=4.5, et d'écart type σ=Ø.4

FONCTION DE REPARTITION DE LA LOI NORMALE CENTREE REDUITE.

'N.C.R' calcule p(X ≤ x), où X est une v.a.r. suivant la loi normale centrée réduite N(Ø , 1). Il s'agit donc d'un cas particulier de ce qui précède: ici m=Ø et σ=1.

Le schéma fonctionnel est donc plus simple:

1:	x		'N.C.R'	1:	p(X ≤ x)
			⟶		

le programme 'N.C.R' appelle le programme 'NORM'.

'N.C.R': « Ø 1 ROT NORM »

Exemple:

1:	.71		'N.C.R'	1:	.76114793191
			⟶		

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 (Xi, Ni), où les Xi sont les valeurs du caractère X, et les Ni sont les effectifs (ou fréquences), on place les couples (Xi, Ni) dans la matrice ΣDAT (la première colonne pour les Xi, la seconde pour les Ni).

Dans le cas d´une statistique donnée par classes ([Ai, A(i+1)[, Ni), 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:

Xi	[Ø,2[	[2,6[	[6,8[	[8,14[
Ni	22	25	15	1Ø

sera représentée dans la variable DATA par la liste:
{ { Ø 2 6 8 14 } { 22 25 15 1Ø } }

Une variable 'ΣDAT1' est utilisée par certains programmes du répertoire ´STAT1´ pour sauvegarder ΣDAT ou pour présenter les calculs intermédiaires amenant à un résultat donné (le cacul 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.
MGEO : calcul de la moyenne géométrique.
MHAR : calcul de la moyenne harmonique.
MDLE : calcul de la médiale (tableau de calcul dans 'ΣDAT1').
GINI : calcul de l´indice de Gini (tableau de calcul dans 'ΣDAT1').
LRTZ : tracé de la courbe de Lorentz (ou de concentration).
HIST : tracé de l´histogramme.
PFC : tracé du polygône des fréquences cumulées.
CUM : création d´une colonne de cumuls.
COL.N: affichage d´une colonne quelconque de 'ΣDAT1' (pour 'GINI' & 'MDLE').
C.COL: calcul d´une colonne dépendante des 2 colonnes de ΣDAT.
MODΣ : modification d´une ou des deux colonnes de ΣDAT.

Répertoire 'STAT1' Programme 'G→D'

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 ΣDAT.

La pile n'est pas affectée par 'G→D'. Toutefois l'appel à 'G→D' se solde par un message d'erreur "Erreur dans 'DATA'" dans le cas où la première sous-liste de 'DATA' ne contient pas exactement un élément de plus que la seconde.

'G D':

```
«   'DATA'  1  GET  LIST→
    →  n
    «  1  n  1  -  START
          OVER  +  2  /  n  ROLLD
       NEXT
       DROP  'DATA'  2  GET  LIST→
       IF  n  1  -  ==  THEN
             2  n  1  -  2  →LIST  →ARRY  TRN  STOΣ
          ELSE
             CLEAR  "Erreur dans 'DATA'"
       END
    »
»
```

Exemple:

La statistique:

Xi	[Ø,2[	[2,6[	[6,8[	[8,14[
Ni	22	25	15	1Ø

,

représentée dans la variable DATA par la liste:
{ { Ø 2 6 8 14 } { 22 25 15 1Ø } },

est transformée en la matrice:

```
[ [  1    22   ]
  [  4    25   ]
  [  7    15   ]
  [  11   1Ø   ] ]
```

et cette matrice est placée dans ΣDAT.

Répertoire 'STAT1' Programme 'MK'

MOMENT D'ORDRE K D'UNE STATISTIQUE SIMPLE.

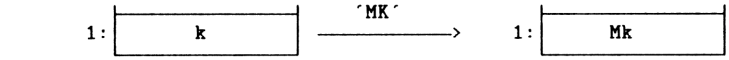
'MK' calcule le moment d'ordre k (noté Mk) de la statistique discrète contenue dans la variable ΣDAT. Dans le cas d'une statistique donnée par classe (et contenue dans la variable 'DATA'), il convient d'appeler tout d'abord le programme 'G→D', pour la transformer en statistique discrète.

La formule donnant le moment d'ordre k de la statistique (Xi, Ni) est:

$$M_k = \frac{1}{N} \Sigma\, Ni\, Xi^k \qquad (N = \text{effectif total} = \Sigma\, Ni).$$

Si k=1, on obtient la moyenne arithmétique (Σ Ni Xi) / N.
Avec k=2, on obtient le carré (Σ Ni Xi^2) / N de la moyenne quadratique.

Le schéma fonctionnel est:



'MK':

```
« → k
  « ΣDAT ARRY→ DROP TOT 2 GET Ø
    → sum mom
    « 1 NΣ START
        SWAP k ^ * mom + 'mom' STO
      NEXT
      mom sum /
    »
  »
»
```

Exemple:

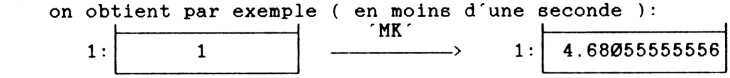
Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	1Ø

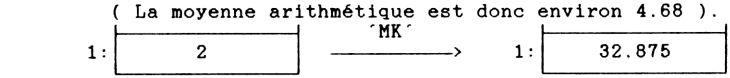
, représentée dans ΣDAT par la matrice à deux colonnes:

```
[ [ 1  22 ]
  [ 4  25 ]
  [ 7  15 ]
  [ 11 1Ø ] ],
```

on obtient par exemple (en moins d'une seconde):



(La moyenne arithmétique est donc environ 4.68).



(La moyenne quadratique est donc √32.875 ≈ 5.73).

Répertoire 'STAT1' Programme 'MCK'

MOMENT CENTRE D'ORDRE K D'UNE STATISTIQUE SIMPLE.

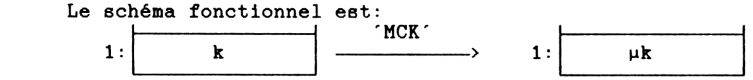
'MCK' calcule le moment centré d'ordre k (noté μk) de la statistique discrète contenue dans la variable ΣDAT. Dans le cas d'une statistique donnée par classe (et contenue dans la variable 'DATA'), il convient d'appeler tout d'abord le programme 'G→D', pour la transformer en statistique discrète.

La formule donnant le moment centré d'ordre k de la statistique (Xi, Ni) est:

$$\mu_k = \frac{1}{N} \Sigma\, Ni\, (Xi-\bar{X})^k \qquad (N = \Sigma\, Ni\ ,\ \bar{X} = \text{moyenne arithmétique }).$$

Si k=1, on obtient Ø.
Avec k=2, on obtient la variance: ($\Sigma\, Ni\, (Xi-\bar{X})^2$) / 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':

```
«  →  k
   «  ΣDAT  ARRY→  DROP  TOT  2  GET  1  MK  Ø
      →  n  moy  mu
      «  1  NΣ  START
            SWAP  moy  -  k  ^  *  mu  +  'mu'  STO
         NEXT
         mu  n  /
      »
   »
»
```

Exemple:

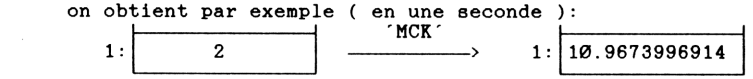
Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	1Ø

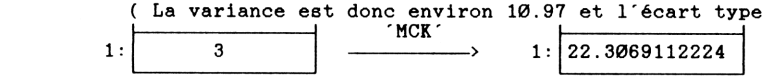
représentée dans ΣDAT par la matrice à deux colonnes:

```
[ [ 1  22 ]
  [ 4  25 ]
  [ 7  15 ]
  [ 11 1Ø ] ],
```

on obtient par exemple (en une seconde):



(La variance est donc environ 1Ø.97 et l'écart type $\sqrt{(\mu 2)} \approx 3.31$)



Répertoire ´**STAT1**´ Programme ´**MCRK**´

MOMENT CENTRE REDUIT D´ORDRE K D´UNE STATISTIQUE SIMPLE.

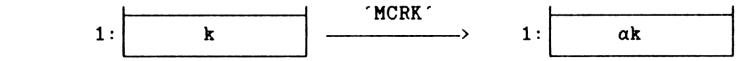
´MCRK´ calcule le moment centré réduit d´ordre k (noté αk) de la statistique discrète contenue dans la variable ΣDAT. Dans le cas d´une statistique donnée par classe (et contenue dans la variable ´DATA´), il convient d´appeler tout d´abord le programme ´G→D´, pour la transformer en statistique discrète.

La formule donnant le moment centré réduit d´ordre k de la statistique (Xi, Ni) est:

$$\alpha k = \mu k / (\sigma \hat{} k),$$

où μk est le moment centré d´ordre k (voir le programme ´MCK´) et où σ est l´écart-type.
Avec k=1, on obtient Ø.
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´:

```
«   →  k
    «  k  MCK  2  MCK  k  2  /  ^  /
    »
»
```

Exemple:

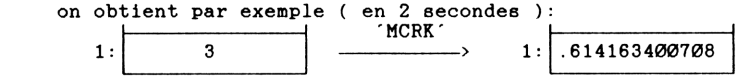
Avec la statistique :

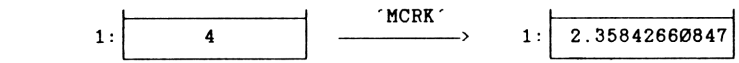
Xi	1	4	7	11
Ni	22	25	15	1Ø

, représentée dans ΣDAT par la matrice à deux colonnes:

```
[ [ 1  22 ]
  [ 4  25 ]
  [ 7  15 ]
  [ 11 1Ø ] ],
```

on obtient par exemple (en 2 secondes):





Répertoire 'STAT1' Programme 'YULE'

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= (q3+q1-2M)/(q3-q1).

Coefficient de Kelley: k= 2*(q3-q1)/(d9-d1).

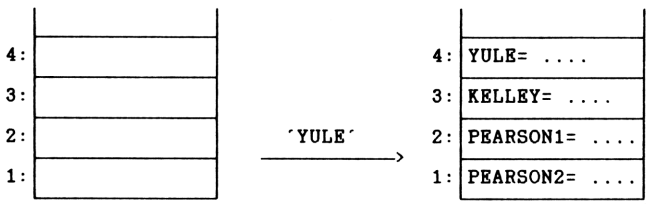
1er coefficient de Pearson: p1= 3*(m-M)/σ.

2ème coefficient de Pearson: p2= (m-md)/σ (pour une série statistique unimodale de mode md).

Avec: q1, q3 : premier et troisième quartiles.
d1, d9 : 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'.

Répertoire 'STAT1' Programme 'YULE'
(suite)

TEXTE DU PROGRAMME 'YULE' ET EXEMPLE D'UTILISATION.

```
'YULE':
    «    { .1 .25 .5 .75 .9 }   QTLE   LIST→   DROP   G→D
         1   MK   2   MCK   ∕   MAXΣ   2   GET   Ø   Ø
         →   d1   q1   med   q3   d9   moy   sig   max   num   lig
         «   1   NΣ   FOR   i
             'ΣDAT(i,2)'   EVAL
             IF   max   ==   THEN
                num   1   +   'num'   STO   i   'lig'   STO
             END
         NEXT
         CLLCD
         "YULE="   q3   q1   +   med   2   *   -
                   q3   q1   -   /   →STR   +   1   DISP
         "KELLEY="   q3   q1   -   d9   d1   -
                   /   2   *   →STR   +   2   DISP
         "PEARSON1="   moy   med   -   sig   /   3
                   *   →STR   +   3   DISP
         IF   num   1   ==   THEN
            "PEARSON2="   moy   'ΣDAT(lig,1)'   EVAL   -
                          sig   /   →STR   +   4   DISP
         END
      »
   »
```

Exemple:

Avec la statistique

Xi	[Ø,5[	[5,1Ø[	[1Ø,15[	[15,2Ø[
Ni	22	25	15	1Ø

représentée dans la variable 'DATA' par la liste:
{ { Ø 5 1Ø 15 2Ø } { 22 25 15 1Ø } },

L'appel du programme 'YULE' provoque au bout de 6 secondes l'affichage de l'écran suivant:

4:	YULE=9.99999999951E-2
3:	KELLEY=1.11658456486
2:	PEARSON1=.355182651996
1:	PEARSON2=.177318528263

Répertoire ´**STAT1**´ Programme ´**QTLE**´

QUANTILES D´UNE STATISTIQUE SIMPLE.

´QTLE´ calcule les quantiles de la statistique groupée (donnée par classes) qui est contenue dans la variable ´DATA´.

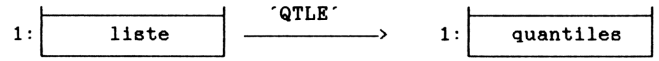
Rappel:

si q est un réel compris entre Ø 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/1Ø (k entier ,1 ≤ k ≤ 9), on obtient le kième décile dk.
Si q= k/4 (k entier ,1 ≤ k ≤ 3), on obtient le kième 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]Ø,1[pour lesquelles sont demandés les quantiles.
où "quantiles" est la liste correspondante des quantiles.

Exemple:

```
liste= { .5 } si on ne veut que la médiane.
liste= { .25 .75 } si on veut le premier et le troisième quartile.
```

N.B: Le programme est interrompu par un message "Erreur dans ´DATA´" si la première liste de ´DATA´ ne contient pas exactement un élément de plus que la seconde.
Si une proportion q dans "liste" n´est pas comprise entre Ø et 1, le quantile correspondant (qui n´existe donc pas), est remplacé dans la liste obtenue par le message "paramètre erroné".

Répertoire 'STAT1' Programme 'QTLE' (suite)

TEXTE DU PROGRAMME 'QTLE' ET EXEMPLE D'UTILISATION.

'QTLE':

```
« DUP SIZE → liste long
« 'DATA' 1 GET SIZE → n
« 'DATA' 2 GET {Ø} + LIST→ DUP
IF n ≠ THEN CLEAR "Erreur dans 'DATA'"
ELSE
1 →LIST →ARRY DUP n 1 →LIST
1 CON DOT / → eff
« 1 long FOR j
liste j GET Ø → q cumul
« IF q Ø ≤ q 1 ≥ OR
THEN "Parametre errone"
ELSE
eff {1}
WHILE cumul q <
REPEAT
GETI cumul + 'cumul' STO
END
LIST→ DROP 1 - 1 →LIST DUP 3
ROLLD GET q cumul - SWAP / 1 +
→ pos coeff
« 'DATA' 1 GET pos GETI 3
ROLLD GET OVER - coeff * +
»
END
»
NEXT long →LIST
»
END
»
»
»
```

Exemple:

Avec la statistique:

Xi	[Ø,2[	[2,6[	[6,8[	[8,14[
Ni	22	25	15	1Ø

qui sera représentée dans la variable DATA par la liste:
{ { Ø 2 6 8 14 } { 22 25 15 1Ø } }.

On met { .1 .25 .5 .75 .9 } au niveau 1 et on appelle 'QTLE':
Au bout de 4 secondes on obtient, au niveau 1 de la pile, la liste
(en mode 3 FIX):
{ Ø.655 1.636 4.24Ø 6.933 9.68Ø }.
Ce qui signifie que, par exemple:
le premier décile d1 ≈ .655
la médiane Me ≈ 4.24Ø et le 3ème quartile ≈ 6.933.

Répertoire ´**STAT1**´ Programme ´**ECAR**´

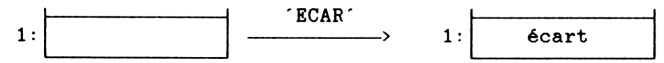
ECART ARITHMETIQUE MOYEN D´UNE STATISTIQUE SIMPLE.

´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: $\frac{1}{N} \Sigma\, Ni \mid Xi - \overline{X} \mid$

où N est l´effectif total ΣNi, et où $\overline{X}$ est la moyene arithmétique.

Le schéma fonctionnel est le suivant:



N.B: le programme ´ECAR´ appelle le programme ´MK´.

´ECAR´:

```
«   ΣDAT   ARRY→   DROP   TOT   2   GET   1   MK   Ø
    →  n   moy   ecart
    «  1   NΣ   START
         SWAP   moy   -   ABS   *   ecart   +   'ecart'   STO
       NEXT
       ecart   n   /
    »
»
```

Exemple:

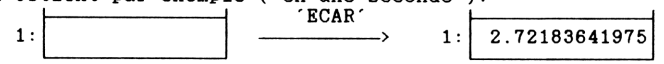
Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	1Ø

, représentée dans ΣDAT par la matrice à deux colonnes:

```
[ [ 1  22 ]
  [ 4  25 ]
  [ 7  15 ]
  [ 11 1Ø ] ],
```

on obtient par exemple (en une seconde):



Répertoire 'STAT1' Programme 'MGEO'

MOYENNE GEOMETRIQUE D'UNE STATISTIQUE SIMPLE.

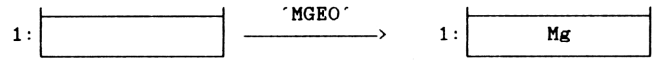
'MGEO' calcule la moyenne géométrique Mg de la statistique discrète contenue dans la variable ΣDAT. Dans le cas d'une statistique donnée par classe (et contenue dans la variable 'DATA'), il convient d'appeler tout d'abord le programme 'G→D', pour la transformer en statistique discrète.

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':
    «   ΣDAT   ARRY→   DROP   TOT   2   GET   1
        →   sum   prod
        «   1   NΣ   START
                sum   /   ^   prod   *   'prod'   STO
            NEXT
            prod
        »
    »
```

Exemple:

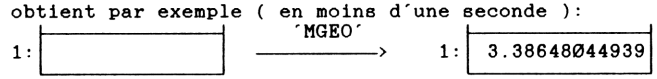
Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	1Ø

, représentée dans ΣDAT par la matrice à deux colonnes:

```
[ [ 1  22 ]
  [ 4  25 ]
  [ 7  15 ]
  [ 11 1Ø ] ],
```

on obtient par exemple (en moins d'une seconde):



Répertoire 'STAT1' Programme 'MHAR'

MOYENNE HARMONIQUE D'UNE STATISTIQUE SIMPLE.

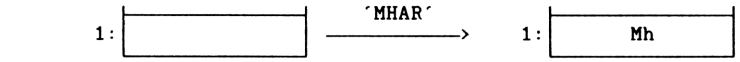
'MHAR' calcule la moyenne harmonique Mh de la statistique discrète contenue dans la variable ΣDAT. Dans le cas d'une statistique donnée par classe (et contenue dans la variable 'DATA'), il convient d'appeler tout d'abord le programme 'G→D', pour la transformer en statistique discrète.

La formule donnant la moyenne harmonique de la statistique (Xi,Ni) est:

$$\frac{1}{Mh} = \frac{1}{N} \Sigma \frac{Ni}{Xi}$$

où N est l'effectif total ΣNi.

Le schéma fonctionnel est le suivant:



'MHAR':

```
«   ΣDAT   ARRY→   DROP   TOT   2   GET   Ø
    →  sum   moy
    «  1   NΣ   START
          SWAP   /   moy   +   'moy'   STO
       NEXT
       sum   moy   /
    »
»
```

Exemple:

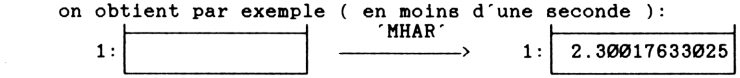
Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	1Ø

représentée dans ΣDAT par la matrice à deux colonnes:

```
[ [ 1  22 ]
  [ 4  25 ]
  [ 7  15 ]
  [ 11 1Ø ] ],
```

on obtient par exemple (en moins d'une seconde):



Répertoire ´**STAT1**´ Programme ´**MDLE**´

MEDIALE D´UNE STATISTIQUE SIMPLE.

´MDLE´ calcule la médiale Ml 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 5Ø% des "masses" cumulées du caractère. Si la statistique X est donnée par les couples ([Ai, A(i+1)[, Ni), la masse correspondant à la classe [Ai,A(i+1)[d´effectif Ni est Ni*(A(i+1) + A(i))/2.

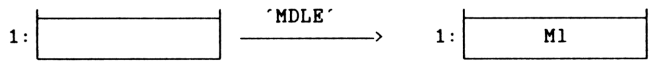
Le calcul de la médiale est effectué par interpolation linéaire dans la classe qui permet de franchir le cap des 5Ø% de masse cumulée.

Le programme ´MDLE´ place dans ´ΣDAT1´ 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	proportion de la masse totale M
Xi	Ni	NiXi	$\Sigma_{j\le i}$ NjXj	$\frac{1}{M}\Sigma_{j\le i}$ NjXj

Le schéma fonctionnel est:



N.B: le programme ´MDLE´ appelle le programme ´G→D´.

Répertoire 'STAT1' Programme 'MDLE' (suite)

TEXTE DU PROGRAMME 'MDLE' ET EXEMPLE D'UTILISATION.

'MDLE':

```
«   G→D   ΣDAT   [1 Ø]   *   ΣDAT   [Ø 1]   *   DOT   ΣDAT   'ΣDAT1'
    STO   'ΣDAT1'   TRN   'ΣDAT1'   5   NΣ   2   →LIST   RDM   'ΣDAT1'
    TRN   Ø   Ø   Ø
    →   masse   cumul   lig   coeff
    «   'ΣDAT1'   {1 1}   1   NΣ   FOR   i
          GETI   3   ROLLD   GETI   4   ROLL   *   DUP
          →   nx
          «   PUTI   nx   cumul   +   DUP   'cumul'   STO   PUTI
              cumul   masse   /   DUP   4   ROLLD   PUTI
              IF   3   ROLL   .5   >   lig   Ø   ==   AND   THEN
                 i   'lig'   STO   masse   .5   *   cumul   -   nx
                 /   1   +   'coeff'   STO
              END
          »
      NEXT
      DROP2   'DATA'   1   GET   lig   GETI   3   ROLLD
      GET   OVER   -   coeff   *   +
    »
»
```

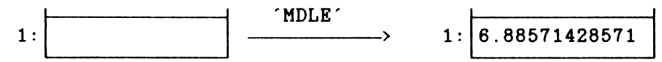
Exemple:

Avec la statistique:

Xi	[Ø,2[	[2,6[	[6,8[	[8,14[
Ni	22	25	15	1Ø

qui sera représentée dans la variable DATA par la liste:
{ { Ø 2 6 8 14 } { 22 25 15 1Ø } }.

On obtient au bout de quatre secondes:



Répertoire 'STAT1' Programme 'GINI'

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 Ø 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 'ΣDAT1'.

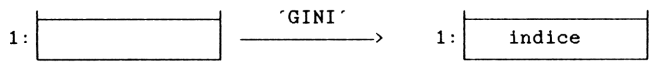
La ligne N°i de ce tableau a la forme suivante (et il y a autant de lignes que de valeurs différentes du caractère étudié):

centres classes	effectifs	effectifs cumulés	% Ui des eff. cumulés	masses	masses cumulées	
Xi	Ni	$\sum_{j\le i} Nj$	$\frac{100}{N}\sum_{j\le i} Nj$	NiXi	$\sum_{j\le i} NjXj$	
colonne1	colonne2	colonne3	colonne4	col.5	colonne6	

	% Vi des masses cum.	base du trapèze	hauteur du trapèze	aire du trapèze
	$\frac{100}{M}\sum_{j\le i} NjXj$	Ui-U(i-1)	$\frac{Vi+V(i+1)}{2}$	base * hauteur
	colonne7	colonne8	colonne9	colonne10

L'indice de Gini est égal à 1-(aire totale)/5000, où "aire totale" représente la somme des coefficients de la dixième colonne de 'ΣDAT1'.

Le schéma fonctionnel de 'GINI' est:



Répertoire 'STAT1' Programme 'GINI' (suite)

TEXTE DU PROGRAMME 'GINI' ET EXEMPLE D'UTILISATION.

'GINI':

```
«   ΣDAT  TRN  1Ø  NΣ  2  →LIST  RDM  TRN  'ΣDAT1'  STO
    TOT  2  GET  ΣDAT  [1 Ø]  *  ΣDAT  [Ø 1]  *  DOT
    Ø  Ø  Ø  Ø  Ø  Ø
    →  t  m  a  b  c  d  e  aire
    «  'ΣDAT1'  {1 1}  1  NΣ  START
         GETI  3  ROLLD  GETI  4  ROLL
         →  n  x
         «  n  a  +  DUP  DUP  'a'  STO  4  ROLLD  PUTI
            3  ROLL  t  /  1ØØ  *  PUTI  n  x  *  DUP
            4  ROLLD  PUTI  3  ROLL  b  +  DUP  DUP  'b'
            STO  4  ROLLD  PUTI  3  ROLL  m  /  1ØØ  *
            DUP  4  ROLLD  PUTI  n  t  /  1ØØ  *
            DUP  4  ROLLD  PUTI  4  PICK  e  +  2  /
            DUP  4  ROLLD  PUTI  3  ROLL  4  ROLL  *  DUP
            aire  +  'aire'  STO  PUTI  3  ROLL  'e'  STO
         »
       NEXT
       DROP2  1  aire  5ØØØ  /  -
    »
»
```

Exemple:

Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	1Ø

,

```
[ [ 1  22 ]
  [ 4  25 ]
  [ 7  15 ]
  [ 11 1Ø ] ],
```

représentée dans ΣDAT par la matrice à deux colonnes:

on obtient par exemple (en 7 secondes):

```
1:                    'GINI'
                  ──────────→      1: .383489943952
```

Répertoire 'STAT1' Programme 'LRTZ'

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 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.

Important:
Pour que le programme 'LRTZ' fonctionne, il faut préalablement avoir éxécuté le programme 'GINI' (et ceci parce qu'on utilise le contenu du tableau 'ΣDAT1').

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 poins (Ui,Vi). Elle est située en dessous de la première bissectrice.

Le tracé remplit tout l'écran de la HP28S; ce qui signifie que l'échelle horizontale est "137 pixels=100 unités", et que l'échelle verticale est "32 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 'ΣDAT1'.

A la fin du tracé, on peut numériser des points à l'aide du réticule (instruction DGTIZ). On revient à l'affichage de la pile en appuyant sur "ON". On trouve alors au niveau 1 de la pile une chaîne de caractères représentant l'image du tracé précédent (instruction LCD→).

```
'LRTZ':
    «   { (0,0) (100,100) constant 1 (0,0) }   'PPAR'   STO
        CLLCD   DRAX   0   0   0   1   NΣ   FOR   i
           'ΣDAT1(i,4)'   EVAL   'ΣDAT1(i,7)'   EVAL
           → x   xa   ya   xb   yb
              «   WHILE   x   xb   <   REPEAT
                    x   ya   yb   ya   -   x   xa   -   *   xb   xa   -
                    /   +   R→C   PIXEL   x   x   R→C   PIXEL   x   100
                    137   /   +   'x'   STO
                  END
                  0   yb   FOR   j
                     xb   j   R→C   PIXEL
                  100   32   /   STEP
                  xb   DUP   yb
              »
        NEXT   3   DROPN   DGTIZ   LCD→
    »
```

Exemple: Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	10

, représentée dans ΣDAT par la matrice à deux colonnes:

```
[ [ 1  22 ]
  [ 4  25 ]
  [ 7  15 ]
  [ 11 10 ] ],
```

le tracé est effectué en 54 secondes.

Répertoire 'STAT1' Programme 'HIST'

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 peut numériser des points à l'aide du réticule (instruction DGTIZ). On revient à l'affichage de la pile en appuyant sur "ON". On trouve alors au niveau 1 de la pile une chaîne de caractères représentant l'image du tracé précédent (instruction LCD→).

```
'HIST':
    «  'DATA' 1 GET DUP SIZE SWAP OVER GETI 3 ROLLD GET DUP2
       - 'DATA' 2 GET LIST→ 2 SWAP START + NEXT Ø Ø
       → num xmax xmin long sum hm hd
       « num 1 - 1 FOR i
             'DATA' 1 GET 1 GETI 3 ROLLD GET SWAP DUP2 -
             'DATA' 2 GET i GET / sum * INV DUP
             hm MAX 'hm' STO
          -1 STEP
          xmin Ø R→C xmax hm R→C 2 →LIST { constant 1 }
          + xmin Ø R→C + 'PPAR' STO CLLCD DRAX
          1 num 1 - FOR i
             → a x h
             « IF h hd > THEN
                  hd h FOR j
                     x j R→C PIXEL hm 32 /
                  STEP
               END
               WHILE x a <
               REPEAT
                  x h R→C PIXEL x long 137 / + 'x' STO
               END
               Ø h FOR j
                  a j R→C PIXEL
               hm 32 / STEP
               h 'hd' STO
             »
          NEXT
          DGTIZ LCD→
       »
    »
```

Exemple:
Avec la statistique:

Xi	[Ø,2[	[2,6[	[6,8[	[8,14[
Ni	22	25	15	1Ø

qui est représentée dans la variable 'DATA' par la liste:
{ { Ø 2 6 8 14 } { 22 25 15 1Ø } },
le tracé est obtenu en 4Ø secondes.

Répertoire 'STAT1' Programme 'PFC'

POLYGONE DES FREQUENCES CUMULEES D'UNE STATISTIQUE SIMPLE.

'PFC' trace le polygône 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 peut numériser des points à l'aide du réticule (instruction DGTIZ). On revient à l'affichage de la pile en appuyant sur "ON". On trouve alors au niveau 1 de la pile une chaîne de caractères représentant l'image du tracé précédent (instruction LCD→).

```
'PFC':
    «   'DATA' 1 GET DUP SIZE SWAP OVER GETI 3 ROLLD GET DUP2
        -  OVER  'DATA' 2 GET LIST→ 2 SWAP START + NEXT Ø Ø
        →  num  max  min  long  x  sum  fc  y
        «  min  Ø  R→C  max  1  R→C  2  →LIST  { constant 1 }  +
           min  Ø  R→C  +  'PPAR'  STO  CLLCD  DRAX
           'DATA'  1  GET  2  1  num  1  -  FOR  i
              GETI  'max'  STO  'DATA'  2  GET  i  GET  sum  /
              →  fi
              «  WHILE  x  max  <
                 REPEAT
                    x  x  min  -  max  min  -  /  fi  *  fc
                    +  DUP  'y'  STO  R→C  PIXEL  long  137  /
                    x  +  'x'  STO
                 END
                 max  'min'  STO  fc  fi  +  'fc'  STO
                 Ø  y  FOR  j
                    max  j  R→C  PIXEL
                 32  INV  STEP
              »
           NEXT
           DROP2  DGTIZ  LCD→
        »
    »
```

Exemple:

Avec la statistique:

Xi	[Ø,2[	[2,6[	[6,8[	[8,14[
Ni	22	25	15	1Ø

qui est représentée dans la variable 'DATA' par la liste:
{ { Ø 2 6 8 14 } { 22 25 15 1Ø } },
le tracé est obtenu en 4Ø secondes.

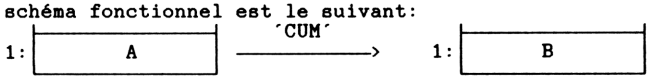
Répertoire 'STAT1' Programme 'CUM'

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.

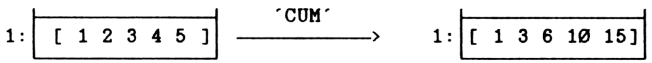
Le schéma fonctionnel est le suivant:

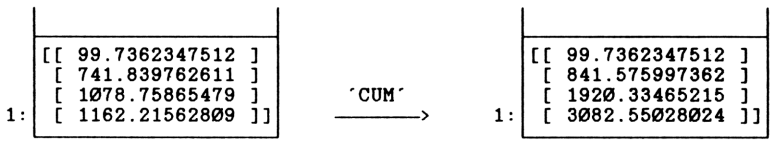


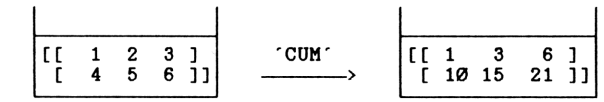
'CUM':

```
«   ARRY→   DUP   LIST→   IF   1   >   THEN   *   END
    →   dim   num
    «   IF   num   1   >   THEN
            num   PICK
            2   num   START
               num   ROLL   +   DUP
            NEXT
            DROP
        END
        dim   →ARRY
    »
»
```

Exemples:







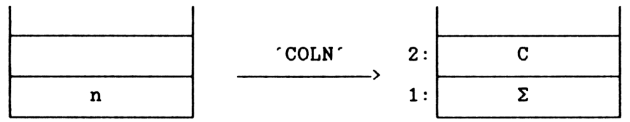
Répertoire 'STAT1' Programme 'COL.N'

EXTRACTION D'UNE DES COLONNES DE ΣDAT1.

'COL.N' extrait la colonne de numéro n de la matrice 'ΣDAT1' 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 'ΣDAT1' 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) sans avoir à placer tout le tableau sur la pile et à utiliser 'EDIT'.

'COL.N':

```
« → n
  « ΣDAT1 MATR n GETC STAT1 DUP ARRY→
    1 GET 2 SWAP START + NEXT
  »
»
```

Exemple:

Avec la statistique :

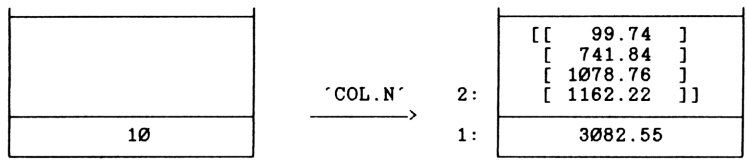
Xi	1	4	7	11
Ni	22	25	15	1Ø

,

```
[ [ 1  22 ]
  [ 4  25 ]
  [ 7  15 ]
  [ 11 1Ø ] ],
```

représentée dans ΣDAT par la matrice à deux colonnes:

Si on appelle le programme 'GINI', le tableau des calculs intermédiaires est placé dans 'ΣDAT1'. On peu alors 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:

$$I \approx 1 - 3082.55/5000 \approx 0.38$$

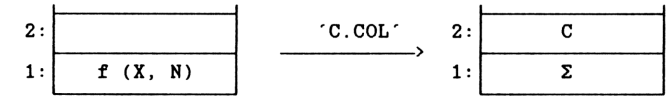
Répertoire 'STAT1' Programme 'C.COL'

CREATION D'UNE COLONNE FONCTION DE CELLES DE ΣDAT.

La matrice 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':

```
«   Ø
→   prog   sum
«   prog   ΣDAT   ARRY→   DROP
    NΣ   2   *   1   -   NΣ   FOR   i
       'N'   STO   'X'   STO   prog   EVAL   DUP
       sum   +   'sum'   STO   i   ROLLD
    -1   STEP
    NΣ   1   2   →LIST   →ARRY
    SWAP   DROP   sum
»
{ X N }   PURGE
»
```

Exemple:

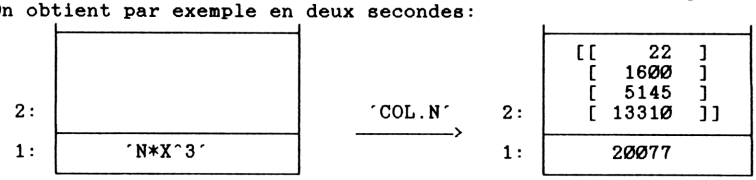
Avec la statistique :

Xi	1	4	7	11
Ni	22	25	15	1Ø

, représentée dans ΣDAT par la matrice à deux colonnes:

```
[ [ 1  22 ]
  [ 4  25 ]
  [ 7  15 ]
  [ 11 1Ø ] ],
```

On obtient par exemple en deux secondes:

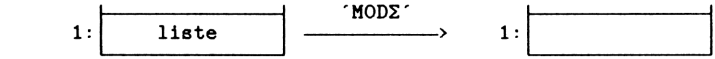


Répertoire 'STAT1' Programme 'MODΣ'

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). 'MODΣ' permet de modifier l'une ou l'autre (ou les deux) des colonnes de ΣDAT.

Le scéma 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 'ΣDAT1'.

```
'MODΣ':
    «   ΣDAT  'ΣDAT1'  STO  LIST→
        →  n
        «  'ΣDAT'
           1  NΣ  FOR  j
              j  1  2  →LIST
              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:

```
[ [ 55 220 ]
  [ 65 250 ]
  [ 75 150 ]
  [ 85 100 ] ],
```

On place { « 5 / 14 - » « 10 / » } au niveau 1 de la pile et on appelle 'MODΣ'. Au bout de 2 à 3 secondes, la matrice ΣDAT contient:

```
[ [ -3  22 ]
  [ -1  25 ]
  [  1  15 ]
  [  3  10 ] ]
```

et le contenu précédent de ΣDAT est sauvegardé dans ΣDAT1.

Remarque: 'MODΣ' est très utile pour les changements 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 (Xi,Yj) étant connue par son effectif ou par sa fréquence. On notera Nij 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 Nij doit être placé dans une variable 'ΣEFF', sous forme de matrice,
avec les conventions suivantes:

	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

La statistique X est donc celle qui figure en première colonne.
Les valeurs de Y figurent donc sur la première ligne de ce tableau.
La matrice 'ΣEFF' contient le tableau à n lignes et p colonnes dont le terme général est Nij, effectif de la valeur (Xi,Yj) du couple (X,Y).

Remarque: il est courant que l'on ait à étudier une statistique double (X,Y) consistant en un ensemble de points (Xi,Yi) tous d'effectif 1, c'est-à-dire se présentant sous la forme:

X	X1	X2	X3	.....	Xn
Y	Y1	Y2	Y3	.....	Yn

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 condition de placer dans la variable 'ΣEFF' la matrice identité d'ordre n.

Les programmes du répertoire 'STAT2' sont les suivants:

- **'EFFX' et 'EFFY'**: calcul des effectifs marginaux.
- **'MX' et 'MY'** : calcul des moyennes des statistiques marginales.
- **'VX' et 'VY'** : calcul des variances des statistiques marginales.
- **'NXY'** : création de la matrice des Nij*Xi*Yj, 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.
- **'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.
- **'VM'** : transformation d'un vecteur en matrice colonne et réciproquement (utile pour lire les vecteurs longs).

Les programmes 'TOTAL', 'MULT', 'MODT', 'VM' 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'.

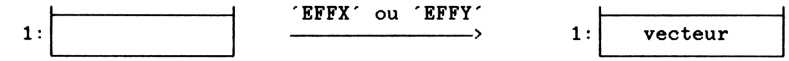
Répertoire 'STAT2'

Programmes 'EFFX' et 'EFFY'

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' décrite en introduction).

Le résultat est obtenu sous forme d'un vecteur au niveau 1:



'EFFX':

```
« ΣEFF DUP SIZE 2 GET 1 →LIST 1 CON * »
```

'EFFY'

```
« ΣEFF TRN DUP SIZE 2 GET 1 →LIST 1 CON * »
```

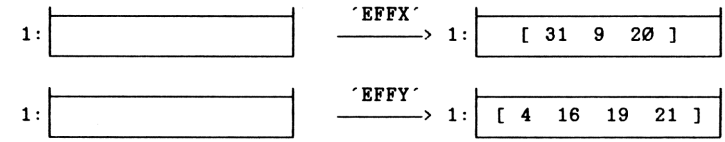
Exemple:

Avec la statistique:

X\Y	1Ø	2Ø	3Ø	4Ø
1	1	7	11	12
5	3	5	Ø	1
1Ø	Ø	4	8	8

On place dans 'ΣEFF'
la matrice: [[1 7 11 12]
[3 5 Ø 1]
[Ø 4 8 8]],

dans 'X' le vecteur [1 5 1Ø],
et dans 'Y' le vecteur [1Ø 2Ø 3Ø 4Ø].
On obtient alors: (en une seconde)



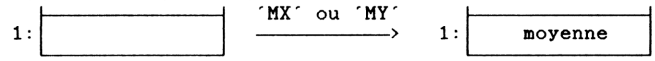
Répertoire 'STAT2'

Programmes 'MX' et 'MY'

MOYENNES ARITHMETIQUES DES STATISTIQUES MARGINALES.

'MX' et 'MY' permettent de connaître les moyennes arithmétiques respectives des statistiques marginales X et Y.
(voir signification de la matrice 'ΣEFF' décrite en introduction).

Le schéma fonctionnel est:



N.B: le programme 'MX' appelle 'EFFX' et 'TOTAL'. De même 'MY' appelle 'EFFY' et 'TOTAL'.

'MX':

```
« EFFX DUP TOTAL SWAP X DOT SWAP / »
```

'MY':

```
« EFFY DUP TOTAL SWAP Y DOT SWAP / »
```

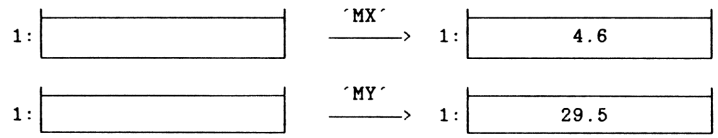
Exemple:

Avec la statistique:

X\Y	1Ø	2Ø	3Ø	4Ø
1	1	7	11	12
5	3	5	Ø	1
1Ø	Ø	4	8	8

On place dans 'ΣEFF'
la matrice: [[1 7 11 12]
[3 5 Ø 1]
[Ø 4 8 8]],

dans 'X' le vecteur [1 5 1Ø],
et dans 'Y' le vecteur [1Ø 2Ø 3Ø 4Ø].
On obtient alors: (en une seconde)



Les moyennes arithmétiques de X et Y sont donc $\bar{X} = 4.6$ et $\bar{Y} = 29.5$

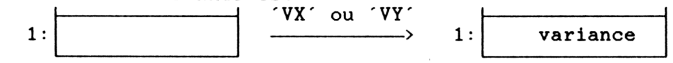
Remarque: dans le cas d'une statistique double (X,Y) consistant en un ensemble de points d'effectif 1: on peut encore utiliser 'MX' et 'MY', à condition de mettre la matrice identité d'ordre n dans 'ΣEFF'. Le plus simple est tout même d'appliquer le programme 'TOTAL' à X et à Y, et de diviser par n.

X	X1	X2	X3	.....	Xn
Y	Y1	Y2	Y3	.....	Yn

Répertoire ´**STAT2**´ Programmes ´**VX**´ et ´**VY**´

VARIANCES DES STATISTIQUES MARGINALES.

´VX´ et ´VY´ permettent de connaître les variances respectives des statistiques marginales X et Y.
(voir signification de la matrice ´ΣEFF´ décrite en 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´:
« EFFX DUP TOTAL / X DUP MULT DOT MX SQ - »

´VY´:
« EFFY DUP TOTAL / Y DUP MULT DOT MY SQ - »

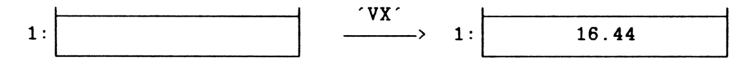
Exemple:

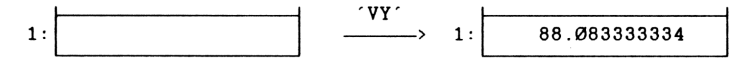
Avec la statistique:

X\Y	1Ø	2Ø	3Ø	4Ø
1	1	7	11	12
5	3	5	Ø	1
1Ø	Ø	4	8	8

On place dans ´ΣEFF´
la matrice: [[1 7 11 12]
[3 5 Ø 1]
[Ø 4 8 8]],

dans ´X´ le vecteur [1 5 1Ø],
et dans ´Y´ le vecteur [1Ø 2Ø 3Ø 4Ø].
On obtient alors: (en moins de deux secondes)





Les variances de X et Y sont donc var(X) = 16.44 et var(Y) ≈ 88.Ø8; on en déduit leurs écarts-types: σ(X) = √ 16.44 ≈ 4.Ø5, et σ(Y) ≈ 9.39 .

Remarque: dans le cas d´une statistique double (X,Y) consistant en un ensemble de points d´effectif 1: 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´utiliser les programmes ´MODT´ (pour calculer les vecteurs des Xi^2 et celui des Yi^2) et ´TOTAL´, en se basant sur la formule de Huyghens:

X	X1	X2	X3	.....	Xn
Y	Y1	Y2	Y3	.....	Yn

$$var(X) = \overline{X^2} - (\overline{X})^2 \quad (\overline{X} = \text{moyenne arithmétique de X}).$$

Répertoire ´**STAT2**´ Programme ´**NXY**´

CREATION DE LA MATRICE DE TERME GENERAL Nij*Xi*Yj.

´NXY´ constitue la matrice de même format que ´ΣEFF´ (voir l´introduction pour les conventions utilisées) et de terme général NijXiYj.
Le résultat est placé dans une variable nommée ´ΣNXY´.

La pile n´est pas concernée par le programme ´NXY´.

```
´NXY´:
    «   ´ΣNXY´   { 1 1 }
        ΣEFF   SIZE   DUP   Ø   CON   ´ΣNXY´   STO   LIST→ DROP
        →  lig   col
        «  1   lig   FOR   i
              1   col   FOR   j
                 ´ΣEFF(i,j)*X(i)*Y(j)´   EVAL   PUTI
              NEXT
           NEXT
           DROP2
        »
    »
```

Exemple:

Avec la statistique:

X\Y	1Ø	2Ø	3Ø	4Ø
1	1	7	11	12
5	3	5	Ø	1
1Ø	Ø	4	8	8

On place dans ´ΣEFF´
la matrice: [[1 7 11 12]
[3 5 Ø 1]
[Ø 4 8 8]],

dans ´X´ le vecteur [1 5 1Ø],
et dans ´Y´ le vecteur [1Ø 2Ø 3Ø 4Ø].

Le programme ´NXY´ place alors, en 6 secondes, la matrice:
[[1Ø 14Ø 33Ø 48Ø]
[15Ø 5ØØ Ø 2ØØ]
[Ø 8ØØ 24ØØ 32ØØ]]
dans ´ΣNXY´. On a par exemple:
´ΣNXY(2,1)=15Ø´ car X(2)=5, Y(1)=1Ø
et ´ΣEFF(2,1)´=3.

Remarque: dans le cas d´une statistique double (X,Y) consistant en un ensemble 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 XiYi).

X	X1	X2	X3	.....	Xn
Y	Y1	Y2	Y3	.....	Yn

Répertoire 'STAT2' Programmes 'CV' et 'COR'

COVARIANCE ET COEFFICIENT DE CORRELATION LINEAIRE.

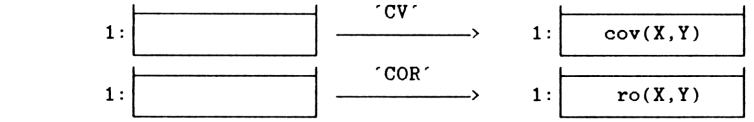
'CV' calcule la covariance cov(X,Y) de la statistique double (X,Y).
'COV' calcule leur coefficient de corrélation linéaire ro(X,Y).
Rappelons que:

$$cov(X,Y) = \frac{1}{N} \Sigma\, Nij\ (Xi-\bar{X})(Y-\bar{Y}) = \frac{1}{N} \Sigma\, NijXiYj - \bar{X}\,\bar{Y}$$

et $$ro(X,Y) = \frac{cov(X,Y)}{\sigma(X)\ \sigma(Y)}$$

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':

```
«  X  ΣEFF  Y  *  DOT  EFFX  TOTAL  /  MX  MY  *  -  »
```

'COR':

```
«  CV  VY  √  /  VX  √  /  »
```

Exemple:

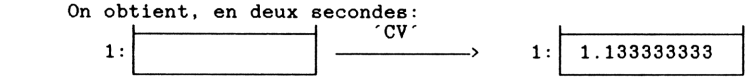
Avec la statistique:

On place dans 'ΣEFF'
la matrice: [[1 7 11 12]
[3 5 Ø 1]
[Ø 4 8 8]],

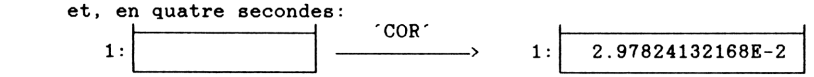
X\Y	1Ø	2Ø	3Ø	4Ø
1	1	7	11	12
5	3	5	Ø	1
1Ø	Ø	4	8	8

dans 'X' le vecteur [1 5 1Ø],
et dans 'Y' le vecteur [1Ø 2Ø 3Ø 4Ø].

On obtient, en deux secondes:



et, en quatre secondes:



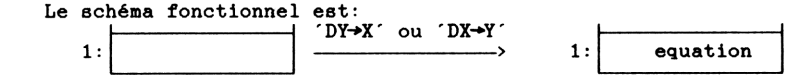
(X et Y sont donc ici faiblement corrélées).

Répertoire ´STAT2´ Programmes ´DY→X´ et ´DX→Y´

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 de la droite 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 que l´équation de la droite des moindres carrés de Y en X est: $y = \frac{cov(X,Y)}{var(X)}(x - \bar{X}) + \bar{Y}$

et que celle des moindres carrés de X en Y est: $x = \frac{cov(X,Y)}{var(Y)}(y - \bar{Y}) + \bar{X}$

(avec les notations habituelles).

N.B: ´DY→X´ appelle ´CV´,´VX´, MY´,´MX´;´DX→Y´ appelle ´CV´,´VY´,´MY´,´MX´.

´DY→X´:

```
«   CV   VX   /
    →  a
    «  ´y´   a   ´x´   *   MY   MX   a   *   -   +   =
    »
»
```

´DX→Y´:

```
«   CV   VY   /
    →  a
    «  ´x´   a   ´y´   *   MX   MY   a   *   -   +   =
    »
»
```

Exemple:

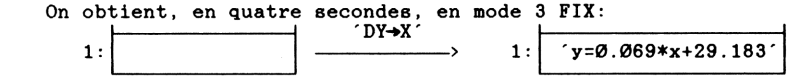
Avec la statistique:

X\Y	1Ø	2Ø	3Ø	4Ø
1	1	7	11	12
5	3	5	Ø	1
1Ø	Ø	4	8	8

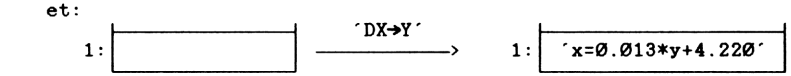
On place dans ´ΣEFF´
la matrice: [[1 7 11 12]
[3 5 Ø 1]
[Ø 4 8 8]],

dans ´X´ le vecteur [1 5 1Ø],
et dans ´Y´ le vecteur [1Ø 2Ø 3Ø 4Ø].

On obtient, en quatre secondes, en mode 3 FIX:



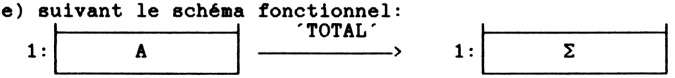
et:



Répertoire 'STAT2' Programmes 'TOTAL' et 'MULT'

SOMME DES COEFFICIENTS D'UN TABLEAU.

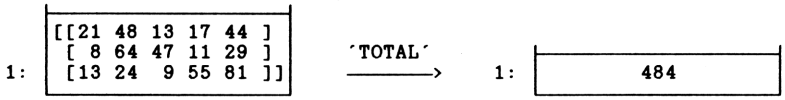
'TOTAL' calcule la somme Σ des termes du tableau A (vecteur ou matrice) suivant le schéma fonctionnel:



'TOTAL':

```
«   ARRY→  LIST→  IF  2  ==  THEN  *  END
    2  SWAP  START  +  NEXT
»
```

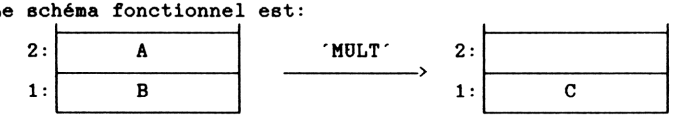
Exemple: (en moins d'une seconde)



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].

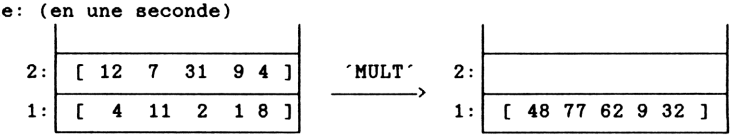
Le schéma fonctionnel est:



'MULT':

```
«  DUP  SIZE  1  GET  →  a  n
«  ARRY→  DROP  a  ARRY→  DROP  2  n  2 →LIST  →ARRY  TRN  ARRY→ DROP
      2  n  *  1  -  n  FOR  i
         *  i  ROLLD
      -1  STEP
      n  1  →LIST  →ARRY
»
»
```

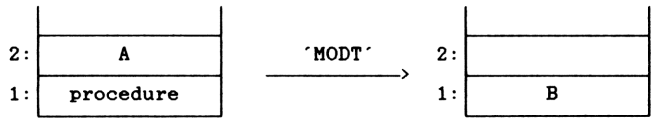
Exemple: (en une seconde)



Répertoire 'STAT2' Programme 'MODT'

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:

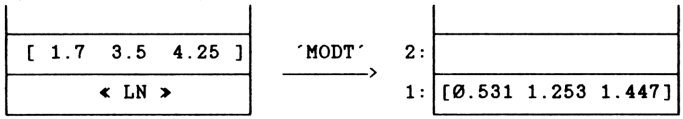


où "procedure" est un programme destiné à s'appliquer à chacun des éléments de A. Ce programme est censé s'appliquer à l'élément figurant au niveau 1 de la pile, et retourner un résultat numérique à ce même niveau.

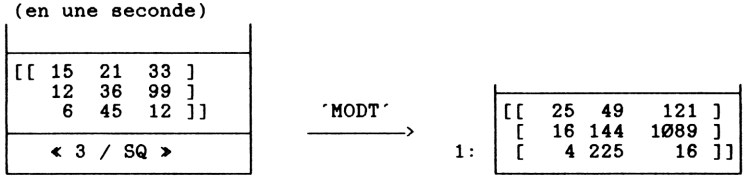
'MODT':

```
«   →  prog
    «  ARRY→  DUP  →  liste
       «  LIST→  IF  2  ==  THEN  *  END
          →  n
          «  1  n  START  prog  EVAL  n  ROLLD  NEXT
             liste  →ARRY
          »
       »
    »
»
```

Exemple 1: (en une seconde)



Exemple 2: (en une seconde)



(tous les coefficients ont été divisés par 3 puis élevés au carré).

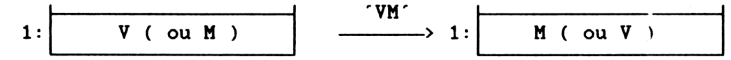
Remarque: 'MODT' est utile lorsqu'il faut rechercher entre deux statistiques X et Y une relation de type Y=K*X^α ou Y=K*A^X. On est alors amené à transformer, au moyen du programme 'MODT', le vecteur des Xi et/ou des Yi en celui des LN(Xi) (des LN(Yi)) et à rechercher une corrélation linéaire entre ces deux nouvelles variables (programmes 'CV', 'COR', 'DY X', etc...).

Répertoire 'STAT2' Programme 'VM'

TRANSFORMATION D'UN VECTEUR EN MATRICE COLONNE (et vice-versa).

'VM' transforme un vecteur V en une matrice colonne M, et inversement, transforme M en V.

Le schéma fonctionnel est:



L'intérêt de 'VM' est qu'on peut ainsi visualiser un vecteur trop long pour apparaître entièrement au niveau 1 de la pile. Il suffit de le transformer en matrice unicolonne et d'utiliser les touches VIEW↑ et VIEW↓.

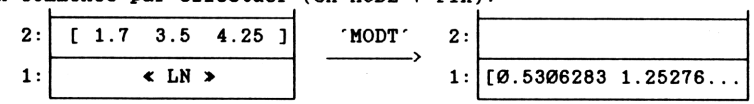
D'autre part cette consultation d'un vecteur sous forme de matrice colonne permet de conserver le format d'affichage obtenu par MODE FIX. En effet la consultation d'un vecteur par EDIT provoque le passage au mode STD (mode standard: 12 chiffres significatifs).

Une fois le vecteur lu par cette méthode, on revient à la forme initiale de celui-ci en appelant 'VM' de nouveau (on pourrait bien sûr utiliser UNDO s'il n'est pas désactivé).

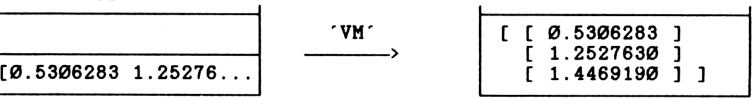
```
'VM':
    «  ARRY→
       IF   DUP   SIZE   1   ==   THEN
            { 1 }   +
          ELSE
            IF   DUP   2   GET   1   ==   THEN   1   1   SUB   END
       END
       →ARRY
    »
```

Exemple: on veut transformer les coordonnées du vecteur [1.7 3.5 4.25] en leur logarithme népérien et lire les résultats en MODE 7 FIX.

On commence par effectuer (en MODE 7 FIX):



Puis on appelle 'VM':



BASES DE DONNEES

La capacité mémoire importante de la HP28S permet d'envisager la gestion de bases de données. Bien sûr, il ne sera pas possible de rentrer un nombre grandiose d'enregistrements, et les programmes qui suivent n'ont pas l'ambition de dépasser DBASE III +. 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.

Dans la mémoire de la HP28S, une base de données se présentera sous la forme d'une liste ayant la forme suivante:

$$\{ \quad n \quad e_1 \quad e_2 \quad \ldots\ldots \quad e_m \quad \} ,$$

où n est un entier représentant le nombre d'éléments de la liste (n y compris), et où e_1 ,e_2 , ..., e_m représentent les différents enregistrements. La présence de n au début est uniquement destinée à permettre de connaître la dimension de la base de données (sans avoir à utiliser SIZE qui nécessite de recopier toute la base de données sur la pile). La gestion de n se fait par programme et est donc totalement transparente pour l'utilisateur.

Un enregistrement est lui-même une liste formée de 1 à 4 chaînes de caractères, chacune de ces chaînes étant formée d'au plus 23 caractères (pour être totalement visible à l'écran):

e_1 = { "chaîne1" "chaîne2" "chaîne3" "chaîne4" }.

IMPORTANT: Une variable 'CLE' doit être présente dans le répertoire et contenir l'une des valeurs 1,2,3, ou 4. Elle est utilisée pour les opérations de tri, les enregistrements étant triés suivant l'ordre alphabétique croissant de la chaîne n°CLE de chaque enregistrement.

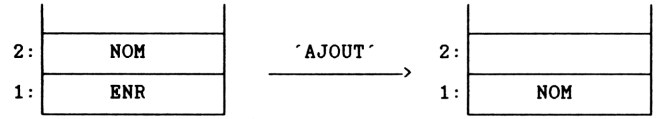
Les programmes du répertoire 'DATA' sont les suivants:

'AJOUT': ajouter un enregistrement.
'LIRE' : lecture de la base de données.
'TRI' : tri de la base de données.
'EFFAC': effacement d'un enregistrement.
'MODI' : modification d'un enregistrement.
'DIM' : nombre d'enregistrements d'une base de données.
'FIND' : recherche d'un enregistrement.

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.

Le schéma fonctionnel est le suivant:



où NOM est le nom de la base de données à compléter (où à créer) et où ENR représente le nouvel enregistrement (voir syntaxe dans l'introduction aux programmes de ce répertoire).

Si la base de données est triée suivant la chaîne N° i (i étant égal au contenu de la variable 'CLE' c'est à dire 1,2,3, ou 4), alors le nouvel enregistrement s'insère dans la base de données de telle manière que celle-ci soit encore convenablement triée.

On notera que le programme 'AJOUT' appelle le programme 'FIND', ceci afin de trouver l'endroit où positionner le nouvel enregistrement.

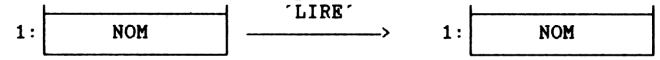
'AJOUT':

```
«   IF   DUP   1   GET   TYPE   2   ==   THEN
    →   c
    «   DUP
        IFERR   DUP   2   GET   THEN
                DROP2   { 2 }   c   1   →LIST   +   SWAP   STO
            ELSE
                DROP   c
                IFERR   CLE   GET   THEN   DROP2   ""   END
                FIND
                →   n
                «   DUP   RCL   1   n   1   -   SUB   c   1   →LIST
                    +   OVER   RCL   n   32000   SUB   +   1   GETI
                    SWAP   DROP   1   +   1   SWAP   PUT   SWAP   STO
                »
            END
    »
    END
»
```

répertoire 'DATA' programme 'LIRE'

LECTURE DE LA BASE DE DONNEES.

'LIRE' permet de consulter le contenu d'une base de données. Le schéma fonctionnel est très simple:



où NOM est le nom de la base de données.

Dès l'appel du programme 'LIRE', le contenu du premier enregistrement apparaît à l'écran.

La touche ▷ (du menu du curseur, appelée aussi touche "RIGHT"), permet de passer d'un enregistrement au suivant.

La touche ◁ (du menu du curseur, appelée aussi touche "LEFT"), permet de passer d'un enregistrement au précédent.

Une touche quelconque (à l'exception des deux précédentes, et des touches "ENTER" et "ON" qui interrompent le programme) permet d'accéder à l'enregistrement dont la valeur de la clé (chaîne 1, 2, 3 ou 4) se rapproche le plus (au sens alphabétique croissant, mais d'une façon plus générale suivant les codes ASCII croissants) de la touche appuyée. Il s'agit donc là d'un véritable accès indexé.

On cesse la lecture en appuyant sur "ENTER".

```
'LIRE':
    «   DUP   Ø   SAME   IF   NOT   THEN   1   END   NOT
        →  flag
        «  DUP   1   GET   2
           →  dim   i
           «  DO   DUP   i   GET
                   1   4   FOR   j
                      DUP
                      IFERR   j   GET   THEN   DROP2   ""   END
                      j   DISP
                   NEXT   DROP
                   DO   UNTIL   KEY   END
                   →  c
                   «  IF   c   "RIGHT"   ==   THEN
                             i   1   +   dim   MIN   'i'   STO
                          ELSE
                             IF   c   "LEFT"   ==   THEN
                                   i   1   -   2   MAX   'i'   STO
                                ELSE
                                   IF   c   "ENTER"   ≠   THEN
                                      c   FIND   dim
                                      MIN   'i'   STO
                                   END
                             END
                      END   c
                   »
                   UNTIL   "ENTER"   ==   END
                   IF   flag   THEN   i   END
                   CLMF
           »
        »
    »
```

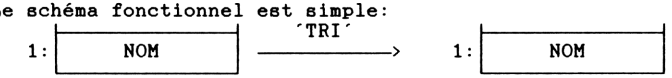
Répertoire 'DATA' programme 'TRI'

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), en se basant sur le contenu de la chaîne n°i de chaque enregistrement (i étant ici le contenu de la variable 'CLE', obligatoirement présente dans le répertoire, c'est-à-dire 1,2,3 ou 4).

N.B: Si un enregistrement ne contient que deux chaînes de caractères alors que le contenu de 'CLE' est 3 (c'est un exemple), alors on considère que la chaîne absente est vide, et les chaînes vides sont triées en tête.

Le schéma fonctionnel est simple:



où NOM représente le nom de la base de données.

```
'TRI':
    «   IF   DUP   TYPE   6   ==   THEN
           DUP   1   GET
           IF   DUP   2   >   THEN   2   SWAP   ELSE   DROP   ABORT   END
        END
        DUP2
        →  g   d   i   j
        «   DUP   i   j   +   2   /   FLOOR   GET
            IFERR   CLE   GET   THEN   DROP2   ""   END
            →  aide
            «   WHILE   i   j   <
                REPEAT
                   WHILE   DUP   i   GET   IFERR   CLE   GET   THEN
                           DROP2   ""   END   aide   <
                   REPEAT   i   1   +   'i'   STO   END
                   WHILE   DUP   j   GET   IFERR   CLE   GET   THEN
                           DROP2   ""   END   aide   >
                   REPEAT   j   1   -   'j'   STO   END
                   IF   i   j   <   THEN
                      DUP   DUP   i   GET   OVER   DUP   j   GET   i
                      SWAP   PUT   j   SWAP   PUT
                   END
                   IF   i   j   ≤   THEN
                      i   1   +   'i'   STO   j   1   -   'j'   STO
                   END
                END
                IF   g   j   <   THEN   g   j   TRI   END
                IF   i   d   <   THEN   i   d   TRI   END
            »
        »
    »
```

Remarque: Il n'est pas possible de donner d'indication de durée de la procédure de tri. Celle-ci peut s'avérer relativement longue, et la durée 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 45 secondes.

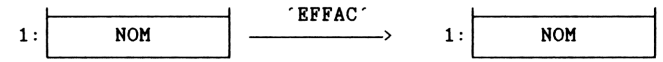
Répertoire 'DATA' programme 'EFFAC'

EFFACEMENT D'UN ENREGISTREMENT.

'EFFAC' permet d'effacer l'un quelconque des enregistrements d'une base de données. Dans le cas où l'enregistrement effacé est le seul de la base de données, celle-ci est supprimée du répertoire.

Attention: le programme 'EFFAC' place toute la base de données sur la pile. La mémoire restante doit donc être suffisante.

Le schéma fonctionnel est:



où NOM est le nom de la base de données (exception: le nom disparaît de la pile si la base de données est supprimée par suite de l'effacement de son unique enregistrement).

Le principe de fonctionnement est le suivant:

- A l'appel du programme 'EFFAC', on entre en consultation de la base de données (c'est le programme 'LIRE' qui est éxécuté).

- Quand on est devant l'enregistrement à effacer, on appuie sur "ENTER" pour valider l'effacement. Pour y renoncer au contraire, il suffit d'appuyer sur "ON". Dans les deux cas le programme est interrompu (dans le deuxième cas, l'état de la pile diffère de ce qui est annoncé ci-dessus).

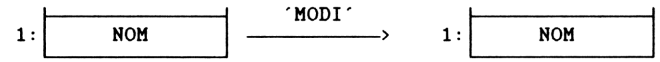
'EFFAC':

```
«   DUP   Ø   LIRE
    →  n
    «  DUP   RCL   1   n   1   -   SUB   OVER   RCL   n   1   +
       32ØØØ   SUB   +   1   GETI   1   -   SWAP   DROP
       IF   DUP   1   ==   THEN
            3   DROPN   PURGE
         ELSE
            1   SWAP   PUT   SWAP   STO
       END
    »
»
```

Répertoire 'DATA' programmes 'MODI' et 'DIM'

MODIFICATION D'UN ENREGISTREMENT.

'MODI' permet de modifier l'un quelconque des enregistrements d'une base de données. Le schéma fonctionnel est simple:



où NOM est le nom de la base de données.

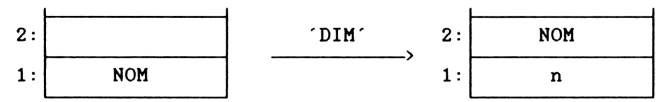
Le principe de fonctionnement est le suivant:
- A l'appel du programme 'MODI', on entre en consultation de la base de données (c'est le programme 'LIRE' qui est éxécuté).
- Quand on est devant l'enregistrement à modifier, on appuie sur "ENTER", et le programme est suspendu. La pile a alors l'aspect suivant:
  * Au niveau 3, le nom de la base de données.
  * Au niveau 2, le numéro de l'enregistrement à modifier.
  * Au niveau 1, l'enregistrement à modifier.
- Si on veut renoncer à modifier cet enregistrement, on appuie directement sur 'CONT'. Sinon, on appuie sur 'EDIT' pour passer en édition de l'objet situé au niveau 1 (Ne pas toucher aux niveaux 2 et 3). Quand les modifications ont été validées par "ENTER", on peut alors faire 'CONT' pour poursuivre le programme.
- Le programme 'MODI' est alors terminé.

'MODI':

```
« DUP Ø LIRE DUP2 GET HALT PUT »
```

DIMENSION D'UNE BASE DE DONNEES.

'DIM' permet de connaître le nombre d'enregistrements contenu dans une base de données (sans avoir à rappeler celle-ci sur la pile). Le schéma fonctionnel est:



où NOM est le nom de la base de données et n le nombre d'enregistrements qu'elle contient.

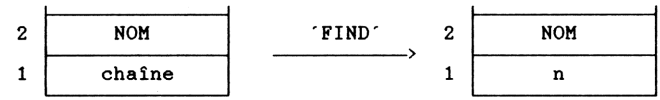
'DIM':

```
« DUP 1 GET 1 - »
```

Répertoire 'DATA' programme 'FIND'

RECHERCHE D'UN ENREGISTREMENT.

'FIND' effectue la recherche dichotomique d'un enregistrement dans une base de données. Le schéma fonctionnel est:



où NOM est le nom de la base de données, et "chaîne" représente la chaîne de caractères à rechercher. L'entier n, à la sortie, représente le numéro d'enregistrement où la chaîne a été trouvée (en cas de succès), ou le numéro de l'enregistrement où la recherche s'est arrêtée (en cas d'échec).

N.B: Le programme 'FIND' compare dans tel ou tel enregistrement, la chaîne N°i de cet enregitrement (si la variable 'CLE' contient i, égal à 1,2,3 ou 4) avec la chaîne de caractères donnée initialement. La base de données est supposée triée, sinon 'FIND' ne fonctionne pas normalement.

N.B: En principe, l'utilisateur n'a pas à appeler lui-même le programme 'FIND'. Ce sont plutôt les programmes 'AJOUT' et 'LIRE' qui appelent et utilisent 'FIND'.

'FIND':

```
«    OVER   1   GET   DUP   2   Ø   Ø
     →   c   dim   fin   deb   mil   ok
     «   WHILE   deb   fin   ≤   ok   NOT   AND
         REPEAT
            fin   deb   +   2   /   FLOOR   'mil'   STO   DUP   mil
            GET   IFERR   CLE   GET   THEN   DROP2   ""   END
            → d
            «   IF   c   d   <   THEN
                     mil   1   -   'fin'   STO
                ELSE
                   IF   d   c   <   THEN
                        mil   1   +   'deb'   STO
                   ELSE
                      1   'ok'   STO
                   END
               END
            »
         END   deb
     »
»
```

Achevé d'imprimer à PARAGRAPHIC - TOULOUSE
Dépôt légal : avril 1989

TABLE DES MATIERES

ISBN 2-908005-00-X

Prix 150 F TTC