Il serait équivalent mais moins commode d'écrire

```
. (variable) = (valeur initiale)
.
.
. (variable) = (variable) + (incrément)
. IF (variable) <= (valeur finale) THEN
```


● SOUS - PROGRAMMES

Lorsqu'un même groupe d'instructions est utilisé plusieurs fois dans un programme, il est préférable de lui conférer une certaine autonomie en le considérant comme un sous-programme. On utilise le schéma ci-dessous

```
. GOSUB
.
.
. END
.
.
. RETURN
```

L'instruction d'appel GOSUB suivie du numéro de ligne où débute le sous-programme déroute l'exécution vers cette ligne. A la rencontre de l'instruction de retour RETURN on revient immédiatement après l'appel.

Les sous-programmes sont placés en général à la fin, après le END du programme principal, pour ne pas se mêler à ce dernier, bien que ce ne soit ni la seule, ni toujours la meilleure possibilité. Cela explique le terme GOSUB (de to go sub signifiant "aller dessous").

Les sous-programmes peuvent s'appeler en cascade; nous reviendrons plus loin sur le sujet.

Même lorsqu'il n'y a pas utilisation multiple il peut être commode de diviser un programme en sous-programmes et chacun lui-même en sous-programmes ... pour faciliter l'élaboration et la lecture ultérieure.

la maquette de la couverture a été réalisée par le L.E.P. Cyfflé - NANCY

B.P. 239 - 54506 VANDOEUVRE-les-NANCY CEDEX
Dépôt légal : 4e trimestre 1985
n° de la publication : 2-85406-087-3
Le Responsable de la collection : Philippe LOMBARD

Réf. N 510

DEUG mention Sciences Section A

Filière M. P. I. S. E.

MODULE MP 1

*

MATHEMATIQUES

ET

PETITS LOGICIELS (1)

INTRODUCTION

L'intention des inventeurs du langage BASIC (T. Kurtz et J. Kemeny) était simplement de rendre immédiat l'accès à l'ordinateur pour attirer les étudiants vers les mathématiques qui, grâce à la puissance de calcul de l'ordinateur, retrouvent un côté expérimental et concret qu'elles ont perdu.

C'est un peu dans cet esprit que la présente initiation a été rédigée. On a joué à fond l'extraordinaire simplicité du BASIC originel : pas de déclaration de variables, un seul type numérique, un petit nombre d'instructions... simplicité qui implique en contrepartie de sévères limitations: mauvaise utilisation de la mémoire, difficulté d'élaborer de longs programmes.

Il est souhaitable, après la phase d'initiation, de s'affranchir de certaines limitations du BASIC. Cependant on n'a pas voulu s'orienter vers des BASIC étendus, voire structurés. A vouloir corriger les insuffisances du BASIC, on aboutit à des langages boursouflés qui n'ont plus du BASIC les qualités d'origine. On a préféré une autre solution et choisi d'indiquer, en parallèle, comment chaque question est traitée dans un langage structuré, à savoir le langage PASCAL. Il ne s'agit pas pour le moment d'écrire un programme en PASCAL, mais seulement de comprendre un programme simple écrit dans ce langage, de manière à lire un ouvrage le prenant comme référence (comme Algorithms + Data Structures = Programs de N. Wirth).

Les références au langage PASCAL sont facultatives, de même que les indications entre *... * relatives à des détails concernant les calculatrices SHARP.

PREMIERE

SEMAINE

1

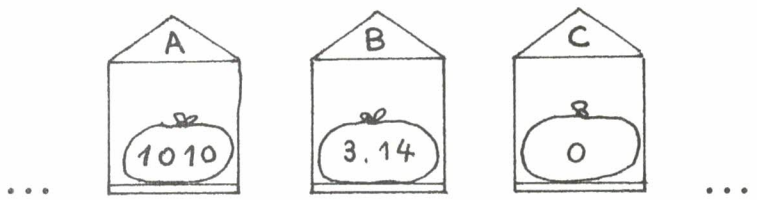
B... A... BA (sic)

● VARIABLES

Nous nous contentons pour le moment de parler de variables numériques simples.

En mathématiques une variable est désignée le plus souvent par une lettre de l'alphabet; on peut lui attribuer une valeur numérique particulière et cette valeur peut être modifiée aussi souvent qu'on le souhaite en fonction des nécessités.

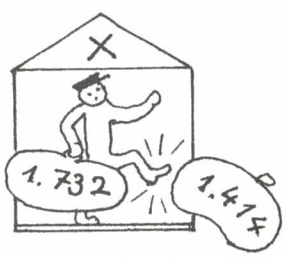
Pour le langage BASIC une variable peut être en particulier désignée (on dit identifiée) par une simple lettre de l'alphabet (son identificateur) . Il lui correspond une case précise dans la mémoire de l'ordinateur, case qui contient la valeur attribuée (on dit affectée) à la variable.



portion de mémoire de l'ordinateur

(on notera que suivant les habitudes anglaises la virgule décimale est remplacée par un point).

L'affectation d'une valeur à une variable consiste à déposer cette valeur dans la case correspondante de la mémoire en chassant la valeur qui s'y trouvait précédemment.



En BASIC elle se fait par une instruction d'affectation du modèle

```
X = 1.732
```

ou

```
LET X = 1.732
```

(impératif de to let signifiant "soit").

Les variables A...Z peuvent représenter des nombres entiers ou décimaux de dix chiffres au plus. Elles peuvent également représenter des nombres réels de dix chiffres significatifs compris entre -10^{100} et 10^{100} à peu près.

Les mots variable et mémoire sont considérés comme synonymes; on parlera du contenu d'une variable.

En PASCAL on doit obligatoirement déclarer les variables pour préciser leur type; on distinguera ainsi un entier (compris entre -2^{15} et $2^{15}-1$) et un nombre réel de 7 chiffres significatifs. Chaque type se verra allouer une place mémoire adaptée et les calculs seront optimisés. En début de programme on écrira

```
var x : real ;
```

Beaucoup d'autres types sont autorisés.

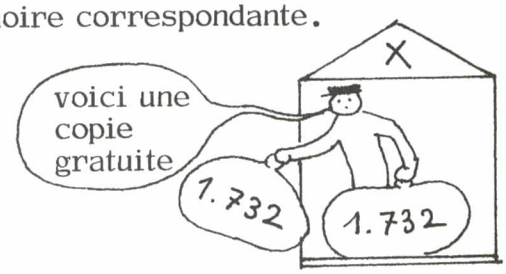
Le terme LET est facultatif sauf dans un cas précis que l'on verra plus loin. Il ne faut pas l'employer lorsqu'on écrit des instructions en mode direct.

Afin d'éviter la confusion avec l'égalité qui exprime une relation entre deux expressions données, l'affectation se fait à l'aide du signe : = ;on écrira ainsi

```
x : = 1.732
```

pour affecter la valeur 1.732 à la variable réelle x .

Chaque fois que la variable sera rencontrée dans une formule à calculer (on dit une expression) l'ordinateur la remplacera par la dernière valeur numérique affectée. Le fait de communiquer cette valeur ne détruit pas le contenu de la case mémoire correspondante.



Par exemple quand le programme rencontre l'instruction

$$Y = X + 1 ,$$

il va d'abord chercher la valeur de X, c'est à dire 1.732; il ajoute 1 pour trouver 2.732 et affecte cette dernière valeur à Y.

Contrairement aux apparences, l'instruction

$$X = X+1$$

est vraiment licite ; elle a simplement pour effet d'augmenter de 1 la valeur de X.

● ENTRÉES / SORTIES

Comme on le verra au cours d'exemples ultérieurs, un programme réalise une suite d'opérations plus ou moins complexes ayant pour effet de modifier la valeur de certaines variables.

Lorsque ces opérations aboutissent au résultat cherché, il est important de le transmettre en l'affichant par exemple à l'écran, ce qui se fait par l'instruction de sortie PRINT (du verbe to print signifiant "imprimer") utilisée suivant le modèle

PRINT X ,

qui ici affiche la valeur de la variable X.

A l'inverse, un programme qui a été conçu pour traiter différents cas ne peut fonctionner que si on lui transmet les valeurs des données; on les introduit par l'instruction d'entrée INPUT (de to put in qui signifie "introduire") suivant le modèle

INPUT X .

Le fonctionnement est le suivant : lorsque le programme rencontre cette instruction il affiche un point d'interrogation "?". L'opérateur doit alors frapper

Essayer l'affectation de valeurs à des variables et le calcul d'expressions en mode direct (mode RUN) en frappant [ENTER] après chacune. Pour vérifier la valeur d'une variable, c'est à dire le contenu de la case mémoire correspondante, frapper par exemple X [ENTER].

Par exemple

X = 1.414 [ENTER]
X [ENTER]
X = 1.732 [ENTER]
X [ENTER]
Y = X + 1 [ENTER]
X [ENTER]
Y [ENTER]
X = X + 1 [ENTER]
X [ENTER] ...

x := x+1 a le même effet en PASCAL alors que la relation x = x + 1 est en principe toujours fausse.

L'instruction PRINT provoque l'arrêt du programme; pour poursuivre il faut appuyer sur [ENTER] ; on peut utiliser l'instruction PAUSE qui affiche la valeur d'une variable pendant un temps bref sans interrompre le programme

L'équivalent en PASCAL est writeln(x).

L'équivalent en PASCAL est readln(x).

la valeur donnée au clavier et opérer un "retour de chariot". Alors la valeur est affectée à la variable, à savoir ici X .

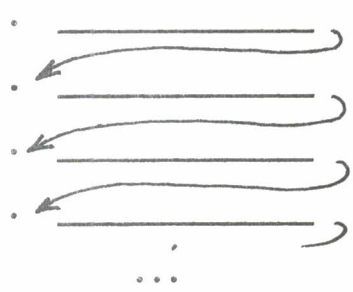
Nous verrons plus tard des variantes de ces instructions.

● LA NUMEROTATION DES LIGNES

Un programme BASIC consiste en une liste d'instructions données à la machine (calcul, impression de résultat...), écrites sur des lignes; chaque ligne est précédée d'un numéro. Lors de l'exécution (lancée par l'ordre RUN) les instructions sont, sauf indication contraire, exécutées l'une après l'autre, dans l'ordre des numéros de lignes, en commençant par le plus petit. L'instruction END termine l'exécution.

N.B.: Les numéros de lignes ne sont pas nécessairement consécutifs; l'usage est de numéroter de 10 en 10; cela permet de modifier le programme en insérant des lignes supplémentaires.

un programme pourra ainsi se présenter sous la forme



exemple

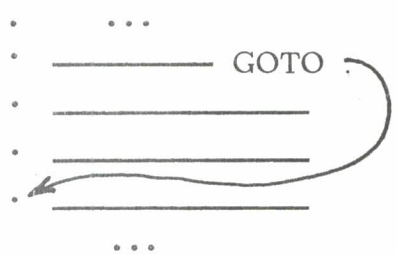
```
10   A = 1
20   PRINT A
30   A = A+1
40   PRINT A
50   END
```

● LE BRANCHEMENT GOTO

GOTO vient de l'anglais to go to signifiant "aller à".

L'instruction GOTO suivie d'un numéro de ligne provoque, lorsqu'elle est rencontrée lors de l'exécution d'un programme, le saut au début de la ligne en question. Le saut peut se produire

- en avant suivant un schéma tel que



Dans ce cas les lignes situées immédiatement après GOTO sont omises lors de l'exécution; nous verrons plus tard l'intérêt de ce type d'instruction. Signalons qu'il ne faudra pas en abuser pour la lisibilité des programmes.

Particularités du BASIC
des calculatrices SHARP

Pour ceux qui veulent lire PASCAL

Le "retour de chariot" est la simple pression sur [ENTER]

Comme de nombreux BASIC, celui des calculatrices SHARP autorise plusieurs instructions par ligne, séparées par le signe de ponctuation " : "

En PASCAL il n'y a pas à proprement parler de lignes. Les instructions sont séparées par le signe de ponctuation "; " et groupées grâce aux "parenthèses" que sont les termes

begin... end.

L'exemple donné à gauche pourra ainsi n'occuper qu'une seule ligne

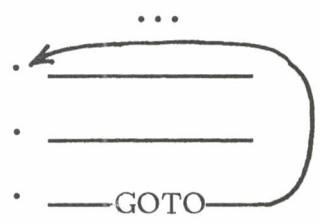
```
10 A=1 : PRINT A : A=A+1 : PRINT A : END
```

Pour essayer le programme on remplacera PRINT par PAUSE.

* L'instruction GOTO peut être "calculée", c'est à dire suivie d'une expression dont la valeur est calculée à l'exécution *

Il est, sinon rigoureusement prohibé, du moins contraire à l'éthique du langage PASCAL de mettre des étiquettes et donc de recourir à l'instruction GOTO.

- en arrière, suivant un schéma tel que



Exemple

```
10  A = 0
20  A = A+1
30  PRINT A
40  GOTO 20
```

On parle ici de boucle. Le programme, dans l'exemple ci-dessus, après avoir exécuté les lignes jusqu'à 40 revient à la ligne 20... Si rien n'arrête le programme dans les lignes intérieures à la boucle, ce dernier se poursuit indéfiniment. Pour cette raison il faut être prudent lorsqu'on programme des boucles.

● EXEMPLE : DECIMALES DE LA DIVISION

Etant donnés deux nombres entiers A, B pas trop grands, il s'agit de trouver leur quotient Q = A/B avec autant de décimales que l'on souhaite; on supposera pour simplifier que A est plus petit que B et on demandera au programme d'afficher l'un après l'autre les décimales en question.

Exemple

$A = 112$	245
$R_1 = 112$	0.457...
$A_1 = 1120$	$q_0\ q_1 q_2 q_3$
$R_2 = 140$	
$A_2 = 1400$	
$R_3 = 175$	
$A_3 = 1750$	
...	

On cherche d'abord le nombre q_0 de fois que A contient B : pour $A<B$ on a $q_0=0$ et un reste R_1 égal à A. On abaisse un zéro ce qui revient à multiplier le reste par 10. On fait ensuite jouer à $10 * R_1$ (* est le signe employé pour la multiplication en BASIC) le rôle d'un nouveau dividende A_1. Le nombre de fois que A_1 contient B donne q_1 et on trouve un reste R_2 que l'on multiplie par 10 pour former un nouveau dividende A_2 ...

On remarque que l'on doit toujours refaire la même suite d'opérations, mais avec un dividende qui change; on va donc utiliser une variable pour représenter ce dividende et assurer la répétition par une boucle.

Particularités du BASIC
des calculatrices SHARP

Remplacer PRINT par PAUSE pour essayer le programme

Avec les calculatrices SHARP, le programme fonctionnera pour des entiers A, B de moins de 9 chiffres.

Le nombre de fois que A contient B s'obtient grâce à l'opération de division A/B et celle de partie entière INT ; c'est simplement

INT (A/B) .

Pourquoi un tel programme alors que le BASIC contient l'opération de division? Pourquoi limiter A, B à neuf chiffres?

Pour ceux qui veulent lire PASCAL

En PASCAL la fonction a div b correspond au INT (A/B) du BASIC alors que a mod b donne le reste.

Pour des nombres a, b inférieurs à 3000, on utiliserait le programme suivant

```
Program division;
  var a, b, q : integer;
  begin repeat
    q := a div b; writeln(q);
    a := 10 * (a mod b)
  until false
  end.
```

Il est contre nature de programmer une boucle non contrôlée en Pascal, ce que l'on fait ici en attendant que les poules aient des dents pour arrêter la boucle. D'ailleurs la rapidité d'exécution d'un langage compilé comme PASCAL rend impossible la lecture des résultats.

Réponses et solutions

Le BASIC SHARP calcule avec seulement 10 chiffres significatifs; si l'on veut plus de 10 décimales, il faut un programme spécifique. Pour un diviseur B de 9 chiffres, le reste, plus petit que B, sera limité à 9 chiffres et les dividendes successifs à 10. La division A/B fournira une partie entière exacte (du moins faut-il raisonnablement l'espérer). Ce pourrait ne pas être le cas pour des nombres plus grands.

Exemple de programme BASIC

```
10  INPUT  A
20  INPUT  B
30  Q=INT (A/B)
40  PRINT  Q    (ou PAUSE Q)
50  A = 10 *(A - Q*B)
60  GOTO 30
```

DEUXIEME

SEMAINE

B... A... BA (sic) : suite

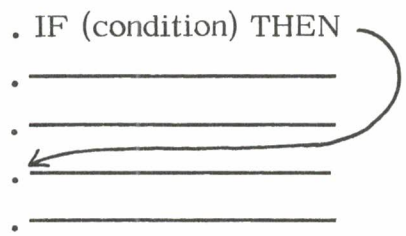
● BRANCHEMENTS CONDITIONNELS

Il arrive très souvent que la suite des opérations à réaliser dépende d'une condition; autrement dit, suivant que cette condition sera réalisée ou non, le programme devra exécuter les instructions de certaines lignes ou bien celles d'autres. On utilise pour cela l'instruction

IF (condition) THEN (numéro de ligne)

(de l'anglais if signifiant "si" et then signifiant "alors"), dans laquelle THEN peut être remplacé par GOTO ou encore THEN GOTO.

Dans un schéma tel que



avec comme exemple

```
10 INPUT A
20 IF A>10 THEN 50
30 PRINT A
40 END
50 A = 10
60 PRINT A
70 END
```

les lignes suivant immédiatement THEN sont sautées lorsque la condition est réalisée. Si elle ne l'est pas le programme exécute toutes les lignes. Dans l'exemple de droite si on introduit une valeur inférieure à 10 les lignes 30 et 40 sont exécutées et la valeur elle-même est affichée; sinon les lignes 50, 60 et 70 sont exécutées et la valeur 10 est affichée.

● EXERCICE : EQUATION DU SECOND DEGRE

Ecrire un programme donnant les racines de l'équation du second degré

$$AX^2 + BX + C = 0$$

en fonction des données A, B, C. On supposera que l'on n'introduit jamais la valeur 0 pour a et on rappelle que si le discriminant

$$D = B^2 - 4AC$$

est positif ou nul il y a deux solutions, éventuellement confondues, données par

$$-\frac{B \pm \sqrt{D}}{2A}$$

Particularités du BASIC
des calculatrices SHARP

Pour ceux qui veulent lire PASCAL

Pour un BASIC comme celui des calculatrices SHARP, autorisant plusieurs instructions par ligne on peut faire suivre IF (condition) THEN d'instructions qui ne seront alors exécutées que si la condition est réalisée; dans le cas contraire on saute directement à la ligne suivante. L'exemple de gauche peut être avantageusement réécrit sous la forme

```
10 INPUT A
20 IF A>10 THEN LET A=10
30 PRINT A
40 END
```

Une particularité du BASIC des calculatrices SHARP est que dans l'instruction d'affectation suivant THEN le LET est obligatoire alors que THEN est facultatif.

. IF (condition) THEN instructions
. ————————————————

équivaut à

. IF (condition opposée) THEN ——
. ———————— instructions
. ————————
. ————————————————

Réécrire l'exemple de cette façon.

En PASCAL on utilise if (condition) then (groupe d'instructions).

Si la condition est réalisée le bloc est exécuté; dans le cas contraire on passe directement à la suite.

On retrouve cette présentation en BASIC lorsque les instructions tiennent en une seule ligne comme dans l'exemple de gauche.

```
Program racines;
  var a, b, c, d, x1, x2, r, i1, i2: real;
  begin
    d:=b * b - 4 * a * c ;
    if d>=0 then
      begin
        x1:= (-b+sqrt(d))/2 ;
        x2:= (-b-sqrt(d))/2 ;
        writeln("racines réelles") ;
        writeln(x1); writeln(x2) ;
      end
```

(à suivre)

et s'il est négatif, les solutions imaginaires ont pour partie réelle $-B/2A$ et pour partie imaginaire $\pm \frac{\sqrt{-D}}{2A}$.

Attention! On utilise * pour la multiplication, ce qui fait écrire D = B * B - 4 * A * C et / pour la division avec des parenthèses là où il convient et encore SQR( ) pour la racine carrée.

Dans le cas des racines réelles, on donnera l'information par

PRINT "RACINES REELLES" ,

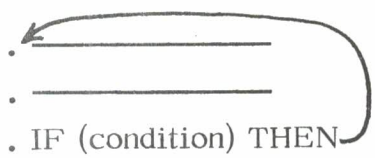
avant d'afficher les racines l'une après l'autre. Dans le cas contraire on pourra utiliser

PRINT "PARTIE REELLE" ...

Noter que l'instruction PRINT suivie d'un texte entre guillemets affiche ce texte tel quel.

● BOUCLES CONTROLEES

Un branchement conditionnel en arrière suivant le schéma

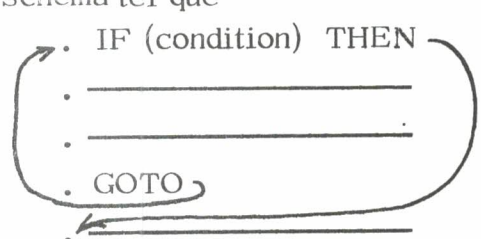


avec comme exemple

```
10  A = 0
20  A = A+1
30  PRINT  A
40  IF A< 9  THEN 20
50  END
```

produit une boucle qui s'exécute jusqu'à ce que la condition ne soit plus réalisée; dans ce dernier cas on passe à la suite du programme. Dans l'exemple on affiche simplement 1, 2, ... , 9 ; en effet, après l'affichage de 9 , la condition 9<9 n'étant pas réalisée, le programme passe à la ligne 50 où il trouve END et s'arrête.

Il est possible de contrôler une boucle par une condition placée en tête; cela permet qu'aucune exécution de la boucle n'ait lieu dans certains cas (alors que précédemment la boucle est toujours exécutée au moins une fois). On utilise un schéma tel que



avec comme exemple

```
10  A = 0
20  IF A >= 9  THEN 60
30  A = A + 1
40  PRINT A
50  GOTO 20
60  END
```

La condition est ici l'opposée de la précédente.

Pour ceux qui veulent lire PASCAL

```
else
  begin
    r := -b/2/a; i1/=sqrt(-d)/2/a ; i2 := -i1 ;
    writeln ("partie réelle"); writeln(r) ;
    writeln ("parties imaginaires") ;
    writeln (i1); writeln (i2)
  end
end.
```

Essayer le programme avec PAUSE
au lieu de PRINT

En PASCAL on écrit simplement

repeat
(instructions)
until (condition)

Par exemple

```
begin
  a := 0;
  repeat
    a := a+1 ;
    writeln (a)
  until  a>=9  (ou a=9)
end
```

La condition est l'opposée de celle du programme BASIC correspondant.

En PASCAL on écrira

while (condition) do
(groupe d'instructions)

Par exemple

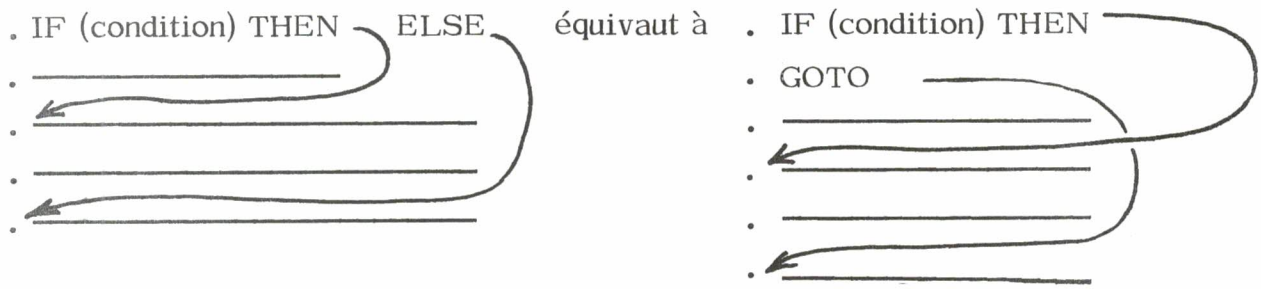
```
begin
  a := 0;
  while  a<9  do
    begin
      a := a + 1 ;
      writeln(a)
    end
end
```

● INSTRUCTION ELSE

L'instruction IF ... THEN ... peut être complétée par

ELSE (numéro de ligne)

(de else signifiant "sinon").



Si la condition n'est pas réalisée on saute à la ligne dont le numéro suit ELSE.

● Boucle FOR NEXT

Il arrive souvent qu'une boucle doive être répétée un nombre déterminé de fois, pour les valeurs d'une variable comprises entre deux extrêmes et espacées d'une même quantité (ou incrément). On utilisera alors un schéma du type

```
. FOR (variable) = (valeur initiale) TO (valeur finale) STEP (incrément).
.  ———————————  )
.  ———————————  )  instructions composant
.  ———————————  )  la boucle
. NEXT (variable)
```

La boucle est exécutée une première fois, la valeur initiale étant affectée à la variable. En fin de boucle on ajoute l'incrément à la variable et on compare à la valeur finale; si elle ne dépasse pas on exécute une nouvelle fois la boucle...

Lorsque l'incrément est égal à 1 on peut l'omettre. Par exemple

```
10 FOR A=1 TO 9
20 PRINT A
30 NEXT A
40 END
```

réalise encore l'affichage des nombres de 1 à 9.

L'incrément peut être négatif, les valeurs successives de la variable allant en décroissant.

Particularités du BASIC
des calculatrices SHARP

Pour ceux qui veulent lire PASCAL

Le BASIC des calculatrices SHARP n'offre pas cette instruction .

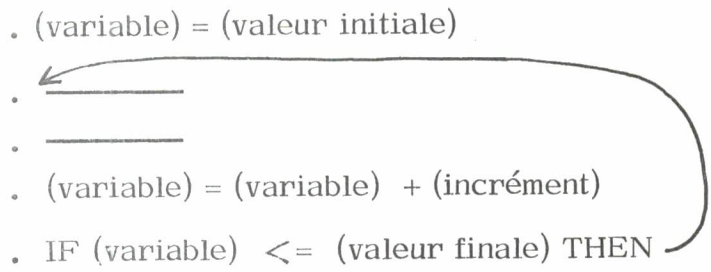
On trouve de même en PASCAL
if (condition) then (groupe d'instructions)
else (groupe d'instructions).

En PASCAL on écrirait

for (variable) := (valeur initiale) to (valeur finale)do(groupe d'instructions)

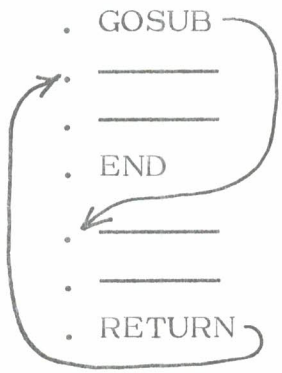
pour un incrément de 1 et on remplacerait to par downto pour un incrément de - 1 .

Il serait équivalent mais moins commode d'écrire



● SOUS - PROGRAMMES

Lorsqu'un même groupe d'instructions est utilisé plusieurs fois dans un programme, il est préférable de lui conférer une certaine autonomie en le considérant comme un sous-programme. On utilise le schéma ci-dessous



L'instruction d'appel GOSUB suivie du numéro de ligne où débute le sous-programme déroute l'exécution vers cette ligne. A la rencontre de l'instruction de retour RETURN on revient immédiatement après l'appel.

Les sous-programmes sont placés en général à la fin, après le END du programme principal, pour ne pas se mêler à ce dernier, bien que ce ne soit ni la seule, ni toujours la meilleure possibilité. Cela explique le terme GOSUB (de to go sub signifiant "aller dessous").

Les sous-programmes peuvent s'appeler en cascade; nous reviendrons plus loin sur le sujet.

Même lorsqu'il n'y a pas utilisation multiple il peut être commode de diviser un programme en sous-programmes et chacun lui-même en sous-programmes ... pour faciliter l'élaboration et la lecture ultérieure.

Pour ceux qui veulent lire PASCAL

En PASCAL on utilise des procédures qui se présentent exactement comme un programme. Elles doivent être déclarées obligatoirement avant le "begin" du programme principal. L'équivalent d'un sous-programme du BASIC est une procédure sans paramètres. Non seulement les procédures peuvent s'appeler en cascades, mais, à la différence du BASIC, elles peuvent contenir paramètres et variables locales .

Les calculatrices SHARP PC 1245 autorisent 10 niveaux d'imbrication des sous-programmes.

* Une instruction GOSUB peut être "calculée" comme l'instruction GOTO*

Pour l'équation du second degré on peut écrire un programme tel que le suivant

```
10 INPUT  A, B, C
20 D = B * B - 4 * A * C
30 IF  D < 0  THEN  50
40 R = √D : PRINT "RACINES REELLES" : PRINT (-B - R)/2/A : PRINT
   (-B + R) / 2/A : END
50 R = √(-D) : PRINT "PARTIE REELLE" : PRINT -B/2/A
60 PRINT "PARTIES IMAGINAIRES" : PRINT R/2/A : PRINT  -R/2/A
70 END
```

TROISIEME

SEMAINE

2

TEXTES ET DESSINS

Le chapitre qui débute ici n'apporte rien de nouveau pour la construction d'algorithmes mais, en plus d'un aspect directement utilitaire, permet d'illustrer la manière dont les données sont comprises par l'ordinateur.

● CARACTERES

Les textes s'obtiennent en mettant à la suite les uns des autres des caractères qui peuvent être des lettres majuscules ou minuscules, des chiffres, des signes de ponctuation...

L'ordinateur n'enregistre et ne traite que des données numériques; plus exactement même, il ne connait que des suites de 0 et de 1. L'information élémentaire, le bit, ne prend que deux valeurs: 0 et 1 ; elle est matérialisée par un système physique pouvant avoir deux états distincts. On obtient une information plus complexe en utilisant plusieurs bits; par exemple si on dispose de N bits, il est facile de voir que l'on a 2^N suites distinctes; pour N = 4, ce seront les 16 suites ci-dessous.

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

On reconnaîtra les développements en base 2 des nombres entiers de 0 à 2^N-1 . Cependant il n'est pas toujours nécessaire d'interpréter ces suites comme des nombres en base 2.

On donnera dans ce chapitre des instructions de la version UCSD adaptée à l'Apple II qui n'ont pas été prévues dans le PASCAL standard; elles sont précédées par le signe * .

Le type char correspond aux caractères.

Revenons à nos caractères. Dans la convention la plus utilisée, désignée sous le nom de code ASCII (American Standard Code for Information Interchange), on dispose de 128 caractères, chacun étant codé par un nombre entier de 0 à 127. D'après ce que l'on vient de voir, cela nécessite 7 bits. Cette convention, américaine, ne prend pas en compte les caractères accentués du français; pour en disposer on a plusieurs possibilités: modifier le code ASCII en prenant le code de certains signes peu usités (Apple IIe) ou utiliser 256 caractères et coder alors sur 8 bits, ou encore utiliser le caractère contre-oblique \ en écrivant par exemple \'e pour é (les deux dernières sont utilisées par les ordinateurs Thomson).

On notera que les minuscules de l'alphabet sont codées entre 97 (pour a) et 122 (pour z), les majuscules entre 65 et 90, les chiffres entre 48 (pour 0) et 57 (pour 9); on notera encore que le signe d'espace a pour code 32.

La fonction CHR$ fait correspondre à un nombre entier entre 0 et 127 le caractère dont le numéro de code est cet entier; par exemple CHR$ (97) est la lettre a .

● CHAINES DE CARACTERES

Un fragment de texte, c'est à dire une suite de caractères, est appelée chaîne de caractères.

En BASIC une chaîne de caractères explicite (on parle de constante de chaîne) peut être obtenue en plaçant ledit fragment de texte entre guillemets; par exemple on écrira

```
"ceci est un texte"
```

(cette manière interdit d'utiliser les guillemets dans le texte lui-même).

On dispose également de variables de chaîne qui sont aux chaînes de caractère ce que sont les variables numériques aux nombres. Pour les distinguer on utilise un nom (ou identificateur) se terminant par le signe $, comme A$, B$ ...

L'opération fondamentale sur les chaînes est celle, notée + , qui met bout à bout deux chaînes données; par exemple le programme

Les caractères accentués ne sont pas disponibles sur les petites calculatrices SHARP.

chr en PASCAL

Attention! Si on a utilisé la variable numérique A on ne dispose plus de la variable de chaîne A$; elles partagent le même espace en mémoire. Il faut utiliser des lettres différentes pour les unes et les autres.

* Le type string correspond aux chaînes. Il ne faut pas confondre les caractères avec les chaînes réduites à un seul caractère, sauf pour ce qui est des constantes.

* la fonction à plusieurs arguments concat joue ce rôle.

```
10  A$ = "ceci est"
20  B$ = "  un texte"
30  C$ =  A$ + B$
```

donne à C$ la valeur "ceci est un texte"

Le langage BASIC fournit également les fonctions LEFT$, RIGHT$ et MID$ servant à extraire des sous-chaînes.

LEFT$(X$,I) est la sous-chaîne comprenant les I premiers caractères de la chaîne X$.

RIGHT$ (X$,I) sélectionne à l'inverse les I derniers.

MID$ (X$,I, J) sélectionne J caractères à partir du caractère de rang I de X$.

L'utilisation de ces fonctions nécessite le plus souvent de connaître la longueur de la chaîne; cette dernière est fournie par la fonction LEN.

Pour finir, notons que ASC(X$) fournit le code du premier caractère de la chaîne X$.

● FONCTIONS STR$ ET VAL

Il faut bien faire la différence entre un nombre et son écriture décimale; par exemple entre le nombre 123 et la chaîne de caractères "123". La différence apparaît clairement lorsqu'on réfléchit à la manière dont l'un et l'autre sont enregistrés par l'ordinateur.

Le nombre 123 peut ainsi être considéré comme un entier entre 0 et 255 codé sur 8 bits sous la forme (son développement en base 2)

01111011.

En revanche, pour coder la chaîne "123" on devra notamment coder chacun de ses caractères, ce qui fera apparaître les codes de 49, 50 et 51; en fait on devra coder aussi la longueur qui est ici 3.

La fonction STR$ fournit la chaîne correspondant à l'écriture décimale d'un nombre donné, alors qu'à l'opposé la fonction VAL essaie d'interpréter une chaîne comme l'écriture décimale d'un nombre (si ce n'est pas le cas elle prend la valeur 0).

* copy remplace exactement MID$.

* length .

En PASCAL la fonction ord fournit le même résultat mais elle s'applique à un caractère et non à une chaîne.

* La procédure str transforme un entier en chaîne.

● AFFICHAGE DU TEXTE

L'affichage d'une chaîne de caractères à l'écran se fait comme pour une valeur numérique par l'instruction PRINT. Elle se fait normalement au début de la ligne suivante. Si on veut que l'affichage se fasse immédiatement après le dernier caractère affiché, il faut que l'instruction PRINT précédente se termine par le point-virgule ";" .

Comparer l'effet des programmes ci-dessous

```
10  A$ = "To"
20  PRINT A$
30  PRINT A$
40  END
```

```
10  A$ = "To"
20  PRINT A$ ;
30  PRINT A$
40  END
```

● EXERCICE : CALCULS SUR LES NOMBRES COMPLEXES

On veut un programme réalisant des opérations sur les nombres complexes, par exemple des multiplications. Les nombres complexes seront introduits sous la forme $a \pm bi$, où a et b sont des valeurs numériques acceptées; on permet également les écritures a, bi, $a \pm i$, i et -i .

Il n'est pas question d'introduire a+bi dans une variable numérique; il en faudra deux, une pour a et une pour b. Comme on veut introduire a+bi en une seule fois il faudra le faire sous la forme d'une chaîne de caractères.

Le programme analysera cette chaîne pour détecter notamment la présence éventuelle de +, - et i et convertir en valeurs numériques ce qui doit l'être.

Pour les textes sur les chaînes, on dispose de deux possibilités: le test direct d'égalité entre deux chaînes, du genre

```
IF X$ = Y$ ...
```

ou l'utilisation de la fonction ASC; la condition

```
LEFT$ (X$, 1) = "a"
```

équivaut par exemple à

```
ASC (X$) = 97.
```

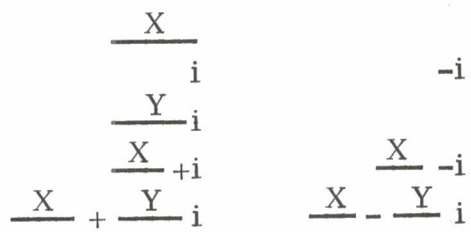
● INDICATIONS

La chaîne de caractères qui représente un nombre complexe se présente d'après nos conventions sous l'une des huit formes suivantes dans lesquelles

Pour ceux qui veulent lire PASCAL

En PASCAL on utilise writeln pour écrire avec retour à la ligne et write si le retour n'est pas souhaité.

—— représente un nombre réel



Une manière de traiter l'exercice consiste à tirer de cette chaîne, que nous appellerons Z$, deux chaînes X$ et Y$ qui représenteront respectivement la partie réelle et la partie imaginaire. Nous allons lire Z$ à partir de la droite; en effet c'est la présence à droite d'un "i" qui signale l'existence d'une partie imaginaire.

Mettons tout Z$ dans X$ pour commencer et au contraire rien dans Y$ (qui sera la chaîne vide). S'il n'y a pas "i" à droite c'est fini. Sinon on va décaler vers la gauche la coupure entre X$ et Y$ jusqu'à rencontrer un +, un - ou simplement l'extrémité gauche (cas où il n'y a pas de partie réelle). Il faut prendre deux précautions supplémentaires :

- le signe "-" ne doit pas être enlevé de Y$

- s'il y a une partie imaginaire et que Y$ est vide (cas de i et a+i) ou que Y$="-" (cas de -i et a-i) alors Y vaut 1 ou -1 .

On écrira donc par exemple le sous-programme commenté suivant :

on met tout dans X$

```
100  X$ = Z$ : Y$ =""
```

on teste le caractère de droite : si ce n'est pas i c'est fini

```
110  C$ = RIGHT$(X$, 1) : IF C$ <> "i" THEN 170
```

dans le cas contraire on enlève i (pour plus de propreté)

```
120  X$ = LEFT$(X$, LEN(X$)-1)
```

s'il n'y a rien dans X$ c'est presque fini

```
130  IF X$="" THEN 160
```

sinon sortons un caractère de X$; si c'est "+" c'est encore presque fini

```
140  C$ = RIGHT$(X$,1) : X$ = LEFT$ (X$,LEN(X$)-1) :
       IF C$ = "+" THEN 160
```

L'esquisse d'un programme PASCAL tirant de la chaîne z des chaînes x et y suivant la méthode indiquée pourra être la suivante :

```
begin
    let x:=z and y:=" ;
    select last character c of x ;
    if c='i' then begin
        repeat
            select last character c of x
            and remove it;
            if c<>'+' then add it to y
        until c='+' or c= '-' or x=" ;
        remove last character of y;
        if y=" then y:='1';
        if y='-' then y:='-1'
    end
end
```

si c'est un autre caractère alors il faut le mettre dans Y$ et, sauf si c'est le signe "-", reprendre la suite des tests

```
150  Y$ = C$ + Y$ : IF C$ <> "-" THEN 130
```

il ne reste plus qu'à corriger éventuellement Y$ et terminer

```
160  IF  Y$ = "" THEN  Y$ = "1"
165  IF  Y$ = "-" THEN  Y$ = "-1"
170  X = VAL(X$) : Y = VAL (Y$) : RETURN
```

Cela étant, le programme principal demandera d'introduire le premier facteur, puis le second; le sous-programme calculera les parties réelle et imaginaire de chaque facteur. Noter que pour utiliser le sous-programme il faut impérativement mettre la chaîne à analyser dans la variable de chaîne choisie dans l'écriture du sous-programme (Z$ dans notre exemplaire); ensuite les valeurs de X et Y rendues doivent être transférées dans au moins une autre paire de variables dans le cas du premier facteur. On utilisera la formule de multiplication des nombres complexes

$$(a+bi)\,(c+di) = (ac-bd) + (bc+ad)\,i \quad .$$

Il restera à présenter le résultat. Différents cas seront à considérer si l'on conserve les conventions d'écriture indiquées. Pour améliorer la présentation des valeurs numériques on aura intérêt à passer par l'intermédiaire de variables de chaîne. On supposera ces valeurs pas trop grandes et on n'exigera pas une grande précision.

● CONSEILS POUR L'ECRITURE DE PETITS PROGRAMMES

Il est impossible d'écrire un programme même aussi court que le sous-programme du paragraphe précédent en le frappant directement sur le clavier d'un ordinateur. C'est évident pour les ordinateurs de poche, mais c'est également vrai pour les ordinateurs de table qui offrent la possibilité de voir plusieurs lignes simultanément. On se munira impérativement d'une feuille de papier, d'un crayon et d'une bonne gomme.

En pratique on ne lésinera pas sur les repères et les commentaires. On pourra soit commencer par l'écriture d'un plan général, soit au contraire par la rédaction de petits morceaux, sachant que plusieurs va-et-vient entre l'une et l'autre de ces activités est souvent nécessaire. L'écriture d'un sous-

On notera que l'on a besoin de connaître le caractère suivant (ou l'absence d'un tel caractère) pour arrêter la boucle . Ici on a simplifié l'écriture en ajoutant i à y dans un premier temps.

On pourra utiliser des lignes telles que

```
10 INPUT "1ER FACTEUR=";Z$ :
   GOSUB 100 : A=X : B=Y
20 INPUT "2EME FACTEUR=";Z$ :
   GOSUB 100: C=X : D=Y
30 X=A*C-B*D : Y=B*C+A*D
```

Il faut supprimer le point qui suit un nombre entier et la notation avec exposant utilisée pour un nombre inférieur à 0.1 si on a du courage.

Comme on l'a vu plus haut, il est très facile d'écrire une esquisse de programme en respectant la syntaxe du langage PASCAL. La langue anglaise courante se mêle aux instructions du langage. Les ennuis commencent lorsqu'on veut parler français. Mêler les instructions en anglais au français produit un franglais épouvantable. Il faut donc traduire les instructions; malheureusement le français se révèle moins souple (end est à la fois le substantif fin et l'impératif finis ou finissez). L'emploi de l'impératif

programme en BASIC exige d'avoir une idée du nom que l'on donnera aux variables; pour s'y retrouver on a intérêt à être méthodique dans le choix.

La numérotation des lignes viendra à la fin, la liberté d'insérer une ligne entre d'autres ne devant servir qu'à corriger ou améliorer un programme déjà bâti. Bien que cela ne soit pas "officiel" il est assez pratique d'employer des flèches après les THEN et GOTO si le branchement se fait sur une ligne proche ou,au contraire,s'il se fait sur une partie plus lointaine ou pour un GOSUB d'indiquer le sous-titre que cette partie ou le sous-programme pourrait avoir.

Il est parfaitement inutile d'écrire des organigrammes, vu que cela n'est possible que pour des programmes très simples ne posant aucun problème sérieux.

au singulier suffit par exemple à rendre désagréable le langage LOGO. Il faudrait chercher la forme la plus neutre possible, réhabiliter l'infinitif...

Bien sûr on n'est pas obligé de se servir du langage PASCAL. Quelle économie de moyens cependant et quel naturel (sans la lourdeur des "fin si" ou "fin tant que") ?

Réponses et solutions

On obtiendra un exemple de programme en complétant par les lignes

```
40  X = INT (X*10000) : X=X/10000 : Y=INT (Y*10000) : Y=Y/10000
50  X$=STR$(X): IF Y=0 THEN Y$="": GOTO 90
60  IF X=0 THEN X$ ="": GOTO 80
70  IF Y>0 THEN X$ = X$+""
80  X$= STR$(Y) + "i": IF Y=1 THEN Y="i"
85  IF Y=-1 THEN Y$ = "-i"
90  PRINT X$ ; Y$: END
```

N.B : En Applesoft il faut remplacer LEFT$(,) par MID$(,1,) aux lignes 120 et 140 car LEFT$(,I) n'accepte pas I=0 .

QUATRIEME

SEMAINE

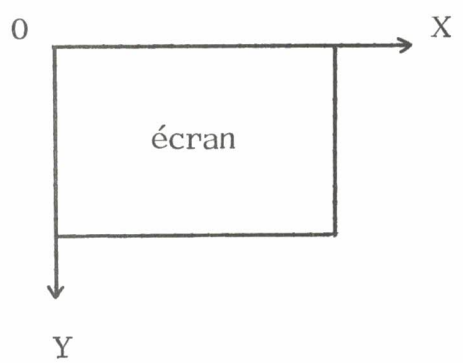
2

TEXTES ET DESSINS : suite

● COORDONNEES

Pour repérer un point dans un plan, ou la portion de plan que représente l'écran, l'ordinateur a besoin de passer par l'intermédiaire de nombres; en l'occurence ce seront simplement les coordonnées dans un repère convenable.

Pour l'affichage sur l'écran, l'usage en BASIC est de choisir l'origine dans le coin en haut à gauche (par analogie avec le texte qui débute dans ce coin) et les axes comme ci-dessous :



On notera que l'axe de Y est orienté positivement vers le bas, à l'opposé des habitudes prises en mathématiques.

Le point lumineux n'est pas infiniment petit; ses dimensions dépendent de la bande passante du moniteur et de ses dimensions dépend le nombre de points que l'on peut afficher en largeur et en hauteur. Pour un écran de 320×200 (Thomson) un point sera repéré par ses coordonnées X, Y qui seront des nombres entiers entre 0 et 319 pour X et 0 et 199 pour Y. Les valeurs non entières de X, Y sont automatiquement arrondies à l'entier le plus proche. Un bon BASIC prévient par une erreur si on demande l'affichage d'un point dont les coordonnées sortent des limites autorisées (ce qui n'est pas le cas pour le BASIC Thomson).

La seconde partie du chapitre ne concerne pas les calculatrices de poche qui ne disposent pas de possibilités graphiques comme la calculatrice SHARP PC 1245.

Cependant on pourra préparer sur ces petites machines des programmes destinés aux ordinateurs à écran, en leur faisant simplement afficher les coordonnées des points utiles; on les reportera sur du papier millimétré pour dessiner les figures.

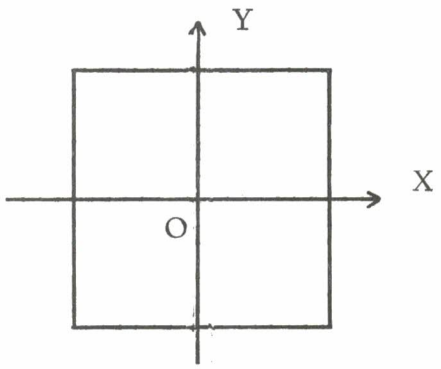
Comme au chapitre précédent on donnera des instructions de la version UCSD adaptées à l'Apple II, qui figurent dans l'unité "Turtlegraphics"; elles sont précédées par le signe * .

* L'origine est placée au coin gauche et la dimension de l'écran est 280×192.

Les instructions graphiques varient d'un BASIC à l'autre. Pour l'affichage du point de coordonnées X, Y on écrira PSET(X,Y) en BASIC Microsoft et HPLOT X,Y en Applesoft. Pour afficher le segment de droite joignant les points de coordonnées (X_1,Y_1) et (X_2, Y_2) on écrira LINE (X_1,Y_1) - (X_2,Y_2) en BASIC Microsoft et HPLOT X_1, Y_1 TO X_2, Y_2 en Applesoft.

● CENTRAGE DU REPERE

Pour des applications géométriques on peut souhaiter placer l'origine au centre de l'écran et orienter les axes de manière usuelle suivant le croquis ci-dessous

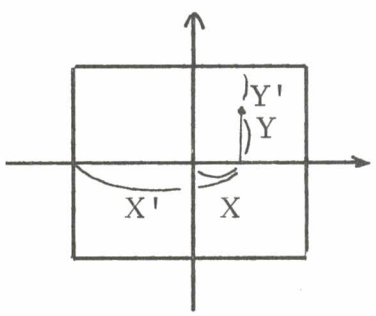


Si XO et YO sont les coordonnées du centre de l'écran dans l'ancien repère, celles dans ce même repère du point dont les coordonnées dans le nouveau repère sont X, et Y seront données par

$$X' = X + XO$$

$$Y' = -Y + YO$$

(le signe - devant Y provient du changement d'orientation) .



On pourra réaliser l'affichage d'un point avec le sous-programme écrit ici pour ordinateur Thomson

```
PSET (X+XO, YO-Y) : RETURN ,
```

et celui d'une ligne avec

```
LINE (X1+XO, YO-Y1) - (X2+XO, YO-Y2) : RETURN .
```

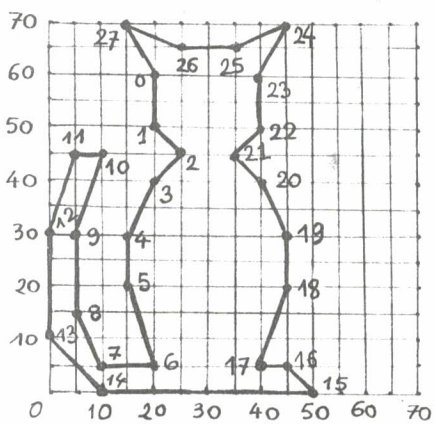
Auparavant il aura fallu affecter les valeurs convenables à XO et YO; dans notre cas XO = 160 et YO = 100.

* La procédure moveto permet d'amener la tortue au point (X,Y) choisi alors que la procédure pencolor décide si le déplacement se fait sans tracé (:= none) ou avec un tracé de couleur (:= white ...).

On voit ici un défaut du BASIC, qui ne possède pas de procédure avec paramètre. Pour utiliser le sous-programme d'affichage d'un point il faut impérativement affecter aux variables X et Y les valeurs des coordonnées; si elles se trouvent dans les variables A, B on devra ainsi commencer par effectuer X = A : Y = B. Naturellement les variables X, Y ne pourront avoir d'autre usage, sous peine de risquer perdre leurs valeurs .

● TRANSFORMATIONS D'UNE FIGURE

On peut réaliser une figure en se donnant un certain nombre de points et en les joignant de proche en proche par un segment de droite. N'ayant pas encore introduit les tableaux, nous allons utiliser une commodité du BASIC qui est l'instruction DATA pour faire figurer un nombre important de données dans un programme

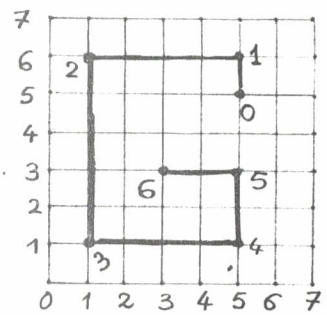


On écrira dans le programme une ou plusieurs lignes précédées de l'instruction et dans lesquelles figureront, séparées par des virgules, les coordonnées des points dans l'ordre, abscisse d'abord, ordonnée ensuite; on répètera le premier point à la fin pour boucler la figure. Dans l'exemple du croquis ci-dessus cela donnera

DATA 20, 60, 20, 50, 25, 45, 20, 40, 15, 30,15, 20,20, 5, 10, 5, 5, 15, 5, 30, 10, 45, 5, 45, 0, 30, 0, 10, 10, 0, 50, 0, 45, 5, 40, 5, 45, 20, 45, 30, 40, 40, 35, 45, 40, 50,40, 60, 45, 70, 35, 65, 25, 65, 15, 70, 20, 60 .

Le dessin de la figure se fera simplement par un programme incluant les lignes

Pour une calculatrice de poche nécessitant de tracer les lignes à la main en reportant les points sur du papier millimétré, on utilisera une figure plus simple, comme par exemple



ce qui donnera

DATA 5,5 , 5,6 , 1,6 , 1,1 , 5,1 ,
5,3 , 3,3 .

La solution la plus rationnelle en PASCAL (comme en BASIC d'ailleurs) serait de mettre les données dans un fichier indépendant du programme lui-même. Détailler la question nous emmènerait beaucoup trop loin.

```
RESTORE
READ X : READ Y : PSET (X+XO, YO-Y)
FOR I=1 TO 28
READ X : READ Y : LINE -(X+XO, YO-Y)
NEXT I
```

L'instruction RESTORE positionne le système pour qu'il lise la première donnée de la liste DATA; elle est inutile pour la première lecture mais il ne faut pas l'oublier si l'on doit lire plusieurs fois la liste de données.

Chaque instruction READ lit une donnée de la liste et prépare le système à lire la suivante; la valeur est affectée à la variable dont le nom suit l'instruction READ.

Pour l'affichage du segment de droite on a omis le point initial; celui-ci est le dernier point considéré par une instruction PSET ou LINE. L'instruction LINE -(X,Y) indique au programme de joindre le dernier point à celui des coordonnées X et Y.

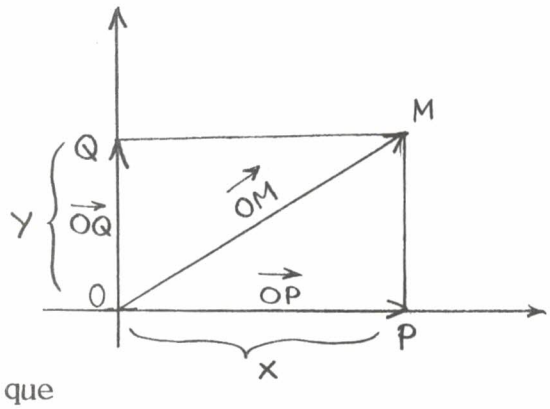
Notre figure est donc prête. Si nous voulons lui faire subir une transformation géométrique il suffit après chaque couple d'instructions READ de modifier les coordonnées X, Y en conséquence.

On voit ici que les transformations géométriques ne peuvent être comprises par l'ordinateur que sous leur forme "analytique", c'est à dire à l'aide des formules sur les coordonnées. Cela n'est cependant vrai que pour les transformations de base. Une transformation qui peut se décomposer en une succession de transformations déjà programmées ne nécessitera pas l'emploi de nouvelles formules; il suffira de faire enchaîner par le programme des sous-programmes effectuant les transformations de base.

● UN PEU DE GEOMETRIE

Nous allons considérer un exemple de transformation linéaire, c'est à dire une transformation qui respecte l'addition des vecteurs et la multiplication des vecteurs par un coefficient scalaire.

Pour calculer l'image d'un point M on commence par décomposer le vecteur $\overrightarrow{OM}$ en le projetant sur les axes



de façon que

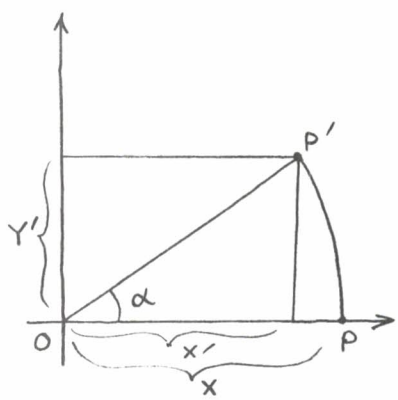
$$\overrightarrow{OM} = \overrightarrow{OP} + \overrightarrow{OQ} \ .$$

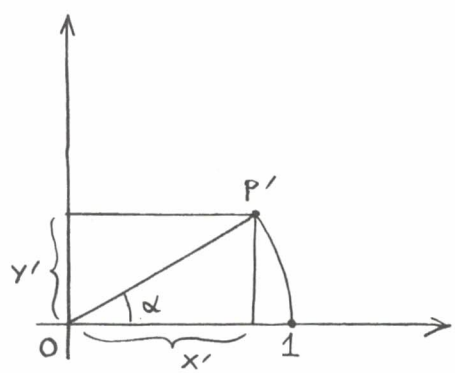
La connaissance des images P' et Q' de P et Q permet alors de trouver celle M' de M par une simple addition vectorielle

$$\overrightarrow{OM'} = \overrightarrow{OP'} + \overrightarrow{OQ'} \ ,$$

qui se traduira par des additions sur les coordonnées. On est ainsi ramené à étudier les transformées des points situés sur les axes.

Prenons le cas de la rotation autour de O d'angle α en radians. Le transformé P' de P est donné par les figures ci-dessous





Dans le cas où X = 1, le point P' est sur le cercle de centre O et de rayon 1; par définition même des fonctions trigonométriques on a

$$X' = \cos \alpha$$
$$Y' = \sin \alpha .$$

Pour obtenir le cas général à partir de ce cas particulier, il faut tout multiplier par le coefficient X de sorte que l'on aura

$$X' = X \cos \alpha$$
$$Y' = X \sin \alpha \ .$$

De façon analogue on aurait obtenu pour les coordonnées de Q' les expressions

$$X' = -Y \sin \alpha$$
$$Y' = Y \cos \alpha$$

et par suite pour celles du point M' lui-même en additionnant

$$X' = X \cos\alpha - Y \sin\alpha$$
$$Y' = X \sin\alpha + Y \cos\alpha .$$

● EXERCICE : VISUALISATION DE TRANSFORMATIONS

Lorsqu'on introduit les coefficients A, B, C, D d'une transformation linéaire d'équations

$$X' = AX + BY$$
$$Y' = CX + DY,$$

le programme doit dessiner en superposition avec la figure de référence choisie, la transformée de cette figure, puis, à la demande, la transformée de la transformée elle-même...

Attention! On ne peut transformer les coordonnées d'un point en écrivant les instructions

```
X = A*X + B*Y : Y= C*X + D*Y .
```

En effet la première instruction modifie X alors que la seconde a besoin de la valeur initiale de cette variable; il faut utiliser au moins une variable intermédiaire pour X et écrire par exemple

```
U = A*X + B*Y: Y= ... : X = U
```

On essaiera le programme avec les valeurs suivantes

1) A = 0.707 , B = -0.707 , C = 0.707 , D = 0.707 .

De quelle transformation s'agit-il? Même question avec

A = 0.866 , B = -0.5 , C = 0.5 , D = 0.866

2) A = 1.2 ou 0.8 , B = 0 , C = 0 , D = 1

ou

A = 1 , B = 0 , C = 0 , D = 1.2 ou 0.8 .

Ce sont des exemples d'affinités . Comparer avec les homothéties

A = 1.2 , B = 0 , C = 0 , D = 1.2 .

et

A = 0.8 , B = 0 , C = 0 , D = 0.8 .

3) $A = 0.866$, $B = 0.5$, $C = 0.5$, $D = -0.866$

On a ici un exemple de symétrie . Que se passe-t-il après deux symétries?

4) $A = 1$, $B = 0.2$, $C = 0$, $D = 1$

et

$A = 1$, $B = 0$, $C = 0.2$, $D = 1$.

Ce sont des transvections.

Le programme qui suit demande les coefficients A, B, C, D ainsi que le nombre F de fois que l'on veut effectuer la transformation; pour F=0 on n'aura que la figure de référence.

```
10 INPUT A, B, C, D : INPUT F: XO=160: YO=100: CLS
20 FOR N=0 TO F
30 RESTORE
40 GOSUB 150 : PSET (X+XO, YO-Y)
50 FOR I=1 TO 28
60 GOSUB 150 : LINE -(X+XO, YO-Y)
70 NEXT I
80 NEXT N
90 END
100 DATA (voir page 4-6)

150 M=N: READ X: READ Y
160 IF M>0 THEN U=A*X + B*Y : Y=C*X+D*Y : X=U : M=M-1: GOTO 160
170 RETURN
```

Pour la préparation sur une calculatrice de poche, on remplace les lignes 10 à 80 par

```
10 INPUT Á, B, C, D : INPUT N
30 RESTORE
50 FOR I=0 TO 6
60 GOSUB 150 : PRINT X: PRINT Y
70 NEXT I
```

On prendra d'abord N=1 (une seule transformation) et on reportera les coordonnées sur du papier millimétré.

CINQUIEME

SEMAINE

3

VERIFICATIONS

Le présent chapitre fournit surtout l'occasion de revoir la notion de dérivée, et plus généralement l'étude d'une fonction au voisinage d'un point, en donnant accessoirement un exemple d'utilisation de la fonction hasard.

● NOMBRES PSEUDOALEATOIRES

La plupart des langages BASIC disposent d'une fonction RND (de l'anglais "random") qui fournit un nombre réel pseudo aléatoire compris entre 0 et 1 avec 1 exclu. Pour affecter à X une valeur au hasard de l'intervalle $[0, 1[$, on écrira simplement

X = RND ;

pour une valeur au hasard dans l'intervalle $[A, B[$ on modifiera la fonction en appliquant une fonction du premier degré valant A en 0 et B en 1 , à savoir l'application

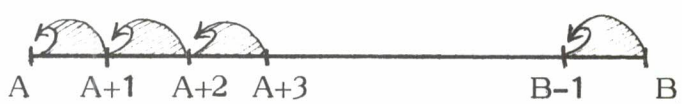
$$x \longmapsto A + (B-A)x \ ,$$

ce qui donnera

X = A + (B - A) * RND .

Si l'on veut obtenir un nombre entier au hasard inclus dans l'intervalle $[A, B[$, avec B exclu , où A et B sont deux nombres entiers donnés, on pourra utiliser la formule

X = INT (A + (B - A) * RND) .



Chaque valeur entière de A à B - 1 est obtenue à partir d'intervalles de valeurs réelles de même longueur et aura donc la probabilité $\frac{1}{B-A}$.

Avec les calculatrices SHARP, on utilisera RND 0 ; l'argument 0 peut être remplacé par n'importe quelle valeur de l'intervalle [0, 1[.

On peut obtenir directement un nombre entier au hasard entre 1 et N en écrivant

RND (N) ,

pour N entier supérieur ou égal à 1 .

En PASCAL la fonction random fournit un nombre entier aléatoire entre 0 et 32767.

Pour obtenir un nombre entier entre a et b avec b exclu, il suffit d'utiliser

b + random mod (b - a) .

Noter que

random/ 32768

fournit un bon équivalent de la fonction RND du BASIC.

En réalité chaque appel de la fonction RND dans un même programme fournit un nombre qui est obtenu à partir du précédent à l'aide d'un procédé parfaitement déterminé mais judicieusement choisi pour donner l'impression du hasard; c'est la raison pour laquelle on peut parler de nombres pseudo aléatoires et non aléatoires.

Avec les ordinateurs Thomson, le premier appel de la fonction RND donne toujours le même nombre; c'est pratique pour la mise au point des programmes mais moins pour les applications car il n'y a plus de hasard.

En Applesoft, on dispose de plus de souplesse : la fonction RND appliquée à une expression dont la valeur est strictement positive génère chaque fois un nombre au hasard nouveau.

● EXERCICE : VERIFICATION D'UNE IDENTITE

On cherche à réaliser un petit programme permettant de déterminer si deux expressions (par exemple $\cos 2x$ et $\frac{1+\cos^2 x}{2}$) sont égales sans grand risque de se tromper.

Le principe est simple : on donnera successivement trois valeurs au hasard à la variable et on comparera les valeurs prises par l'une et l'autre des deux expressions. S'il y a concordance, le programme affichera "EGALITE PROBABLE" et dans le cas contraire "DIFFERENCE PROBABLE".

L'adaptation du programme au cas des deux expressions particulières se fera en complétant des lignes de programmes destinées à calculer la valeur des expressions, commençant par exemple par

F = (première expression avec X comme variable)

et

G = (deuxième expression avec X comme variable).

On prévoira également l'introduction des bornes A, B de l'intervalle sur lequel on souhaite tester l'égalité.

Plutôt que d'avoir à compléter un programme il serait plus agréable d'introduire les expressions comme des données, sous forme de chaînes de caractères par exemple. Cependant il faudrait alors soit faire que le programme se modifie lui-même (ce qui demande de travailler directement sur la mémoire de l'ordinateur), soit faire analyser l'expression par le programme à la manière du langage BASIC (ce qui est trop complexe pour une petite application).

Pour éviter cet inconvénient avec les calculatrices SHARP, il suffit d'employer l'instruction RANDOM avant d'utiliser la fonction RND.

La procédure randomize joue le même rôle en PASCAL.

Le langage PASCAL n'est pas (sauf exception) interprété mais compilé. Cela signifie qu'après avoir écrit un programme dans la mémoire de la machine, on ne peut pas le lancer directement; il faut passer par la phase de compilation qui traduit le programme, dit programme-source, en codes plus directement assimilables par la machine. Chaque modification d'un programme nécessite une nouvelle compilation. La méthode envisagée ici n'est donc pas transposable.

De toute manière la transposition n'aurait pas eu d'intérêt pratique. Il n'existe pas de machine légère à mémoire permanente programmable en PASCAL.

On optera pour la solution consistant à modifier légèrement le programme à chaque utilisation. Bien sûr, il faudra que cette modification, dont l'objet est d'inscrire les formules de F et G, puisse être opérée simplement et rapidement. En particulier ces formules ne devront apparaître qu'une fois dans le programme; la meilleure garantie à cet égard est de les faire figurer dans un sous-programme, qui, pour être facilement accessible et contrairement à l'habitude, sera placé vers le début.

● DISCUSSION

Il n'est pas question de tester l'égalité sous la forme F = G. En effet, même si les deux expressions sont rigoureusement égales , le calcul peut donner lieu à une légère différence.

Prenons l'exemple d'une machine calculant avec 10 chiffres significatifs (les ordinateurs Thomson calculent avec 7 chiffres significatifs en simple précision et à 15 en double précision lorsque cette dernière est accessible; d'autres donnent couramment 9 chiffres). Cela signifie que si F ou G est donnée par une fonction de base (par exemple SINX, X*X..., on peut espérer 10 chiffres significatifs exacts, ou encore une erreur relative de 10^{-9} dans le pire des cas, c'est à dire une majoration de l'erreur par $10^{-9}\,|F|$. En revanche il n'y a pas de règle universelle permettant de prévoir l'erreur dans le calcul d'une formule complexe. Des erreurs importantes peuvent apparaître soit par accumulation d'erreurs, soit dans le calcul de différences entre grandes valeurs voisines.

On supposera donc que F et G ne mettent pas en jeu des quantités trop grandes et que l'erreur sur leur calcul est majorée par 10^{-8}; on supposera encore que les valeurs de F et G ne sont pas trop petites pour rester grandes vis à vis de 10^{-8}. Dans ce cas on peut tester l'égalité par $|F - G| < E$ où $E = 10^{-8}$ ou 10^{-7}.

Si l'on devait manipuler des expressions grandes ou petites mettant en jeu des puissances de 10 , il serait préférable de raisonner en termes d'erreur relative, c'est à dire de faire figurer en facteur au second membre un terme représentant la grandeur de F et G, par exemple $|F| + |G|$. On effectuera alors le test sous la forme

$$|F - G| < E(|F| + |G|),$$

en prenant un peu de marge pour E, par exemple $E = 10^{-6}$ ou 10^{-5}.

L'adaptation de E se fera expérimentalement. Le programme ne donnera de touta manière qu'une indication, sans valeur de preuve, ni dans un sens ni dans l'autre.

Les calculatrices SHARP calculent avec 12 chiffres mais n'en retiennent que 10 lorsque le résultat est affecté à une variable.

Essayer par exemple en mode direct le calcul de

$$(10^9 + 10^3)^2 - (10^9)^2 - 2 . 10^{12}$$

expression dont la valeur est

$$10^6 .$$

On écrira dans le BASIC des calculatrices SHARP l'expression sous la forme

(E9+E3) * (E9+E3) - E9* E9 -2*E12 .

Que trouve-t-on? Peut-on expliquer pourquoi ?Demander la valeur de chaque terme de l'expression.

Faire le même essai avec

$$(10^8+10^3)^2-(10^8)^2-2 . 10^{11} .$$

Essayer encore $A = (10^8+10^3)^2$, $B = (10^8)^2$, $C = 2 . 10^{11}$ et demander A -B- C , et enfin $A = 10^8$, $B = 10^3$ en demandant

(A+B) * (A+B)-2 A *A - 2*A*B .

Réponses et solutions

$(10^9+10^3)^2$ vaut exactement

1 000 002 000 001 000 000 .

La machine ne calculant qu'avec 12 chiffres significatifs, le 1 de droite est ignoré.

Les différences dans les derniers essais viennent du fait que la machine ne retient que 10 chiffres lorsque le résultat doit être affecté à une variable; tout se passe comme si le calcul se faisait tantôt avec 12 chiffres tantôt avec 10.

Voici un programme adapté aux calculatrices SHARP

```
10  REM EGALITE
15  REM VAR X
20  RANDOM : GOTO 60
30  F =
40  G =
50  RETURN
60  INPUT "BORNES "; A, B
65  FOR I = 1 TO 3
70  X = A+(B-A)* RND 0
80  GOSUB 30
90  IF ABS(F-G)> E-7 PRINT "ERREUR PROBABLE" : END
95  NEXT I
100 PRINT "C EST POSSIBLE" : END
```

On peut supprimer les lignes 65 et 95 et remplacer le END final par GOTO 70 ; le programme effectuera un essai qui pourra être répété autant de fois que l'on voudra en appuyant sur [ENTER] .

La ligne 90 comporte un arrêt en milieu de boucle qui n'est pas très propre; ce n'est pas un exemple à copier.

Essayer d'abord F = SIN X * SINX et G = 1 - COSX * COSX avec des bornes quelconques pour tester le programme.

Essayer ensuite, après être passé en radians par RADIAN [ENTER] , F = SINX et G = X - X * X * X / 6 , d'abord avec les bornes 0 et 1 , puis avec les bornes 0 et 0.1 . Voir l'explication dans le cours d'Analyse AN1 : on a pris un développement limité de sinx .

SIXIEME

SEMAINE

VERIFICATIONS : SUITE

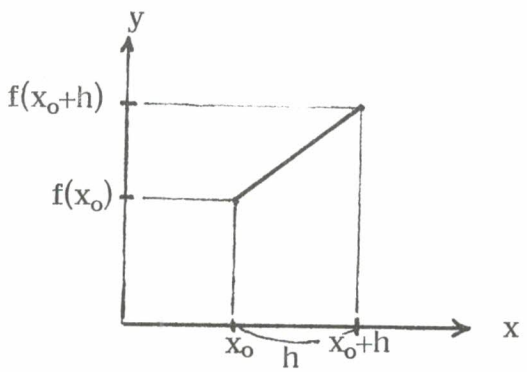
● CALCUL APPROCHE D'UNE DERIVEE

Considérons une fonction f à valeurs réelles de la variable réelle x. Etant donnés un point x_o et un accroissement h de la variable, l'accroissement correspondant de la fonction est la quantité

$$f(x_o + h) - f(x_o) .$$

Si on divise par l'accroissement de la variable, on obtient le taux d'accroissement

$$\frac{f(x_o + h) - f(x_o)}{h} .$$



Pour de nombreuses fonctions (que l'on appelle dérivables) ce rapport se rapproche d'une valeur limite $f'(x_o)$ appelée dérivée lorsqu'on donne à l'accroissement h des valeurs de plus en plus petites.

Il est naturel de penser que l'on obtiendra une valeur approchée de la dérivée en calculant simplement le taux d'accroissement pour une petite valeur de h.

Il est donc tentant de donner à h la plus petite valeur autorisée par l'ordinateur, qui, dans le cas d'un ordinateur Thomson, vaudrait 10^{-38}. Malheureusement l'ordinateur a toutes les chances de ne faire aucune différence entre x_o et $x_o + 10^{-38}$ puisqu'il n'autorise qu'un nombre de chiffres significatifs très inférieur à 38. On obtiendrait alors zéro dans tous les cas, ce qui est absurde.

La précision sur le taux d'accroissement est d'autant meilleure que l'accroissement h est moins petit. Il faudra donc réaliser un compromis entre la nécessité de précision sur le taux et le fait d'avoir à s'approcher suffisamment de la limite.

Nous reprendrons le cas d'une machine calculant avec 10 chiffres significatifs (il faudra effectuer les adaptations nécessaires pour un ordinateur Thomson) et nous supposerons que les valeurs intervenant dans le calcul de f ne soient pas trop grandes de façon à espérer une erreur sur la différence $f(x_o+h) - f(x_o)$ inférieure à 10^{-8}.

● DISCUSSION

Soit $d = f'(x_o)$ la valeur exacte de la dérivée. L'erreur commise en remplaçant d par un taux d'accroissement est

$$\frac{f(x_o+h) - f(x_o)}{h} - d = \frac{f(x_o+h) - f(x_o) - d \cdot h}{h} .$$

La première question qui se pose est de majorer le numérateur du membre de droite.

Le discours abstrait autour de la définition de la limite est le suivant : la valeur $\varepsilon(h)$ de chaque membre de l'égalité ci-dessus tend vers zéro quand h tend vers zéro; par suite

$$f(x_o+h) = f(x_o) + d \cdot h + h \cdot \varepsilon(h) ,$$

où l'on distingue au second membre une partie linéaire affine $f(x_o) + d \cdot h$ et un terme $h \cdot \varepsilon(h)$ qui est infiniment petit devant h. Notre problème exige davantage d'information sur le terme $h \cdot \varepsilon(h)$. Nous allons voir que dans la plupart des cas on peut trouver u ne constante C pour laquelle

$$\left| f(x_o+h) - f(x_o) - d \cdot h \right| \leqslant C h^2 .$$

Ainsi le terme complémentaire tendra vers zéro comme le carré de l'accroissement h. Et nous aurons une information quantitative, plus précise et plus élémentaire à la fois.

● QUELQUES EXEMPLES

a) Considérer la fonction $f(x) = \cos x - (1 - \frac{x^2}{2})$; montrer que sa dérivée seconde est toujours positive ou nulle; en considérant les variations de $f'(x)$ puis de $f(x)$ sur l'intervalle $[0, +\infty[$ montrer que $f'(x)$ puis $f(x)$ sont positives ou nulles sur cet intervalle. En déduire l'inégalité

$$1 - \frac{x^2}{2} \leqslant \cos x \leqslant 1$$

et donc

$$|\cos x - 1| \leqslant \frac{x^2}{2} \ .$$

b) Montrer de façon analogue que pour $x \geqslant 0$ on a

$$x - \frac{x^3}{6} \leqslant \sin x \leqslant x$$

et donc que pour $0 \leqslant x \leqslant 1$ en particulier

$$|\sin x - x| \leqslant \frac{x^2}{6} \ .$$

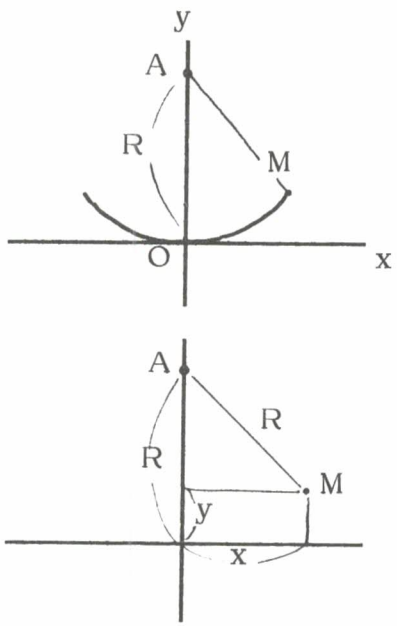
c) Montrer que pour un polynôme $p(x)$ on a toujours

$$|p(x) - p(0) - x\,p'(0)| \leqslant C \cdot x^2$$

où C est une constante convenable.

La démonstration générale de ce phénomène, pour une fonction supposée deux fois dérivable, est faite dans le cours d'Analyse AN1 : c'est un cas particulier de la formule de Taylor.

Terminons par une interprétation géométrique. Plaçons-nous pour simplifier dans le cas d'une fonction f telle que $f(0) = 0$ et $f'(0) = 0$. Sa courbe représentative sera alors tangente à l'axe des x en O.



Le cas le plus fréquent est celui où f présente un minimum ou un maximum en O, la courbe ayant une allure arrondie au voisinage. Supposons pour voir que ce soit exactement un arc de cercle de rayon R, centré au point A de coordonnées (O, R).

Une application du théorème de Pythagore au triangle ci-contre donne pour un point M de coordonnées $(x, y = f(x))$ sur la courbe

$$R^2 = x^2 + (y - R)^2 \ .$$

Faire un petit programme qui, pour une valeur donnée de X, calcule successivement les expressions

$1 - \cos X$,

$X^2/2$ (équation d'une parabole),

$1 - \sqrt{1 - X^2}$ (équation d'un arc de cercle).

Essayer les valeurs (en radians) 1, 0.1, 0.01 et 0.001.

On en tire, sachant que $y \leqslant R$ et donc $y - R \leqslant 0$ dans notre exemple

$$y = R - \sqrt{R^2 - x^2} \ ,$$

ce qui, par une utilisation adéquate de la quantité conjuguée $R + \sqrt{R^2 - x^2}$, aboutit à

$$y = \frac{x^2}{R + \sqrt{R^2 - x^2}} \ .$$

On se limite aux petites valeurs de x (en particulier x R). Il vient donc

$$|y| \leqslant \frac{x^2}{R} \ .$$

En réalité pour x petit on a presque $|y| \leqslant \frac{x^2}{2R}$. On voit que la propriété cherchée ainsi que la constante C sont liées au rayon d'un arc de cercle auquel on pourrait presque identifier la courbe.

● EXERCICE : PROGRAMME DE VERIFICATION D'UNE DERIVEE

On demande d'écrire un petit programme chargé de vérifier une formule de dérivation en choisissant au hasard trois valeurs de la variable et en comparant les valeurs de l'expression donnée pour la dérivée d'une part et celle d'un taux de variation convenable, calculé à partir de l'expression de la fonction, d'autre part. On introduira l'expression de la fonction par

F =

et celle de la dérivée par

D = ,

de la même manière que dans le programme chargé de vérifier une égalité.

Le problème principal réside dans le choix de h et dans la manière de tester la compatibilité entre dérivée et taux de variation.

Supposons que la dérivée d et la constante C apparaissant dans la discussion ne soient pas très grandes, comme dans les exemples étudiés. Pour $h = 10^{-4}$ l'erreur commise sur

$$f(x_o + h) - f(x_o) - d \cdot h \ ,$$

due essentiellement à l'imprécision du calcul de f, restera de l'ordre de 10^{-8}.

Si d est bien la dérivée de la fonction f ce terme devra être lui-même majoré en valeur absolue par $C h^2$ qui sera aussi du même ordre. La valeur de h choisie est optimale dans le sens qu'elle équilibre les deux causes l'empêchant d'être nul.

En se donnant un peu de marge on pourra contrôler que l'expression ci-dessus est bien majorée par une quantité E égale à 10^{-7} ou 10^{-6}. On ne doit pas prendre E beaucoup plus grand, car l'inégalité s'écrit encore

$$\left| \frac{f(x_o+h) - f(x_o)}{h} - d \right| \leqslant \frac{E}{h}$$

et il faut bien que $\frac{E}{h}$ soit petit pour que le test soit significatif. Une valeur de 10^{-3} ou 10^{-2} convient mais on n'a pas beaucoup de marge.

Réponses et solutions

Un programme destiné aux petites calculatrices SHARP et inspiré de celui de la première partie du chapitre sera

```
10  REM DERIVEE
15  REM VAR X
20  RANDOM : GOTO 60
30  F =
35  RETURN
40  D =
45  RETURN
60  INPUT "BORNES"; A, B : H = E-4
65  FOR I = 1 TO 3
70  X = A + (B-A)* RND 0
80  GOSUB 30 : G = F : X = X+H : GOSUB 30 : GOSUB 40
90  IF ABS(F - G - D*H) > E-6  PRINT "ERREUR PROBABLE" : END
95  NEXT I
100 PRINT "C EST POSSIBLE" : END
```

Il attire les mêmes remarques.

SEPTIEME

SEMAINE

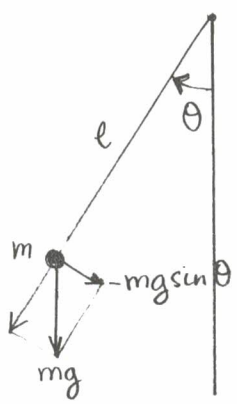
4

SIMULATION D'UN PENDULE

On trouve dans ce chapitre un premier rendez-vous avec les Sciences Physiques. On traite un problème très classique, l'étude du mouvement d'un pendule, par une simulation directe, sans passer par la formalisation mathématique. Cette démarche remplace l'expérimentation habituelle; elle en possède les principales vertus. Il est ainsi possible d'intégrer un brin de réalisme en ajoutant par exemple un facteur d'amortissement.

Dans un deuxième temps on considère les équations du problème et on discute leur linéarisation, toujours à partir de l'expérimentation à l'ordinateur. Cela donnera un petit aperçu de ce qu'est l'analyse mathématique.

● MOUVEMENTS DU PENDULE



Considérons un pendule formé d'une masse pesante au bout d'une tige sans masse de longueur l. Lorsque le pendule est écarté de l'angle θ par rapport à la position d'équilibre, son poids l'y rappelle par le couple

$$-mgl\sin\theta\ ,$$

obtenu en multipliant par l la composante du poids mg suivant la perpendiculaire à la tige, le signe $-$ venant de ce que le couple est de signe opposé à θ et $\sin\theta$.

On vérifiera la concordance des signes entre angles et couples en prenant par exemple pour sens positif le sens inverse des aiguilles d'une montre; on a ainsi $\theta<0$ dans le croquis.

On utilisera des noms de variables autorisés, par exemple

G pour l'accélération de la pesanteur g
L pour la longueur l
A pour l'angle θ
Q pour la vitesse angulaire ω
T pour le temps t
U pour l'intervalle Δt

Prenons le pendule à un instant donné t, dans la position définie par l'angle θ, animé d'une vitesse angulaire $\omega = \frac{v}{l}$, si v est la vitesse linéaire de l'extrémité sur l'arc de cercle qu'elle parcourt.

Pendant que s'écoule un court intervalle de temps Δt, l'angle θ varie peu, de même que le couple $-mgl\sin\theta$. Ce dernier fournit le produit du taux de variation de ω par le moment d'inertie

$$\frac{\Delta\omega}{\Delta t} \cdot m l^2 .$$

La vitesse angulaire aura donc au bout de ce laps de temps la nouvelle valeur $\omega' = \omega + \Delta\omega$ donnée par

$$\omega' = \omega - \frac{g}{l} \sin\theta . \Delta t .$$

(Ceux qui préfèrent raisonner en terme de vitesse linéaire diront que la composante $-mg \sin\theta$ de la force suivant l'arc fournit le produit de l'accélération $\Delta v / \Delta t$ par la masse m et aboutiront à $v' = v - g\sin\theta . \Delta t$, d'où encore le résultat).

Pour trouver la nouvelle valeur $\theta' = \theta + \Delta\theta$ de l'angle, en radians, on remarquera de même que les valeurs de ω et ω' sont voisines et on écrira que l'une ou l'autre donne le taux de variation $\frac{\Delta\theta}{\Delta t}$ de θ, d'où par exemple

$$\theta' = \theta + \omega' . \Delta t .$$

(Ou encore la distance Δs parcourue le long de l'arc telle $\Delta s / \Delta t = v'$).

Supposons le pendule lâché sans vitesse à partir de la position définie par l'angle θ_o, avec $\theta_o < 0$ comme dans notre figure. A partir de la situation initiale pour laquelle $\theta = \theta_o$ et $\omega = 0$, on peut simuler le mouvement du pendule en laissant s'écouler des intervalles de temps Δt et en actualisant les valeurs de ω et de θ à la fin de chaque laps de temps à l'aide des formules précédentes.

● DISCUSSION

Dans la seconde formule nous aurions aussi bien pu utiliser ω au lieu de ω' ou encore la moyenne $(\omega + \omega')/2$.

Ce dernier choix correspond à un couple de rappel constant pendant l'intervalle de temps Δt, égal à la valeur au début de l'intervalle; la vitesse varie alors linéairement et la distance angulaire parcourue s'obtenant par moyenne sur les vitesses extrêmes. Retenons ce choix.

D'abord on précise (voir plus loin) la valeur de G et on demande la longueur L et l'intervalle de temps U par

```
10  G=9.81: INPUT "L="; L, "U="; U
```

On initialise le temps par

```
15  T=0
```

et vitesse et angle par

```
20  Q=0 : INPUT "ANGLE"; A : A=-A ,
```

le programme se chargeant lui-même de mettre un signe - devant l'angle.

Le temps est enregistré par

```
30  T=T+U
```

et la poursuite du mouvement obtenue par

```
60  GOTO 30
```

Pour utiliser la vitesse angulaire ω , il suffit de conserver l'ancienne valeur de Q dans R avant d'actualiser, faisant débuter la ligne 40 par

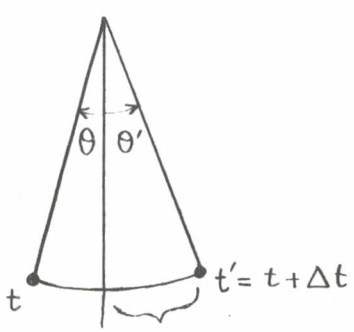
```
40  R=Q : Q=...
```

et de changer la ligne 50 en

```
50  A = A + (Q+R) * U/2
```

Dans ces conditions le mouvement approché que nous avons décrit commence par être plus rapide que le mouvement réel. En effet, dans chaque intervalle angulaire (θ, θ'), le couple de rappel retenu pour le mouvement approché, calculé sur la plus grande valeur absolue du sinus, est plus fort que le couple réel qui s'exerce dans le même intervalle. Précisément cela est vrai tant que l'on a $\theta \leqslant 0$ c'est à dire pendant la descente du pendule vers la position verticale. Au contraire l'argument s'inverse pendant la remontée; le mouvement approché est alors plus lent que le mouvement réel. Si on note ainsi les durées de chacune des deux premières phases du mouvement approché, on aura alors un encadrement de la durée réelle qui est la même pour la descente et la remontée, comme on pourrait le voir en inversant le temps.

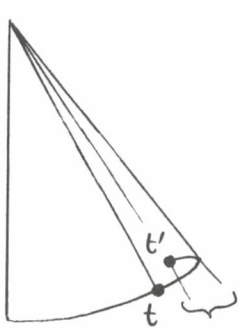
On repère le passage à la verticale par le fait que θ de négatif devient positif, et la fin de la remontée au fait que ω de positif devient négatif.



correction

En réalité, la première fois que l'on obtient $\theta' > 0$ le passage à la verticale a été effectué au cours de l'intervalle Δt précédent.

On peut corriger en retranchant θ'/ω' au temps puisque l'on a dépassé d'un temps τ tel que $(\theta' - 0)/\tau$ soit la vitesse angulaire.



correction

De même, lorsqu'on obtient pour la première fois $\omega' < 0$ la fin de la remontée a été dépassée.

On peut ajouter la quantité négative

$$l\,\omega'/g\sin\theta$$

puisque l'on aura dépassé d'un temps tel que $(\omega' - 0)/\tau$ soit égal à $-\frac{g}{l}\sin\theta$.

Pour ces corrections, qui jouent sur de petites quantités, on n'a pas besoin de valeurs précises de ω et θ ; vouloir corriger les corrections serait vain. En particulier on n'aura pas besoin de se préoccuper de choisir entre ω et ω' ou entre θ et θ' pour celle des grandeurs qui ne change pas de signe et varie peu en valeur relative.

Pour noter le passage à la verticale on écrit

```
60  IF  A< 0  THEN 30
70  PRINT  T
```

La remontée du pendule s'effectue après avoir noté le passage à la verticale par

```
25  F=0
70  IF  F=0  PRINT T : F=1
80  GOTO 30
```

Pour noter la fin de la remontée on précise

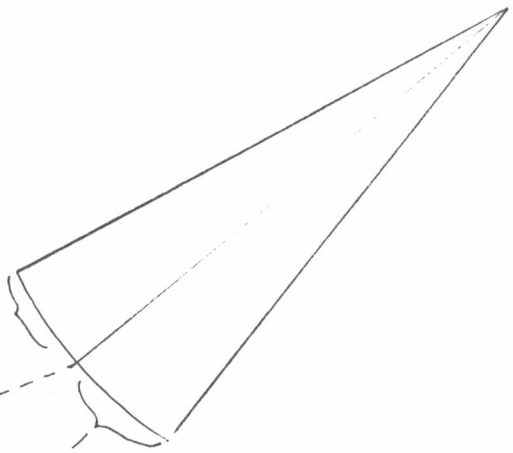
```
80  IF Q>0  THEN 30
90  PRINT T : END
```

On écrira donc PRINT T-A/Q en 70.

On écrira donc PRINT T+L*Q/G/SIN(A) en 90.

● RAFFINEMENT

On peut obtenir de meilleurs résultats en prenant la valeur du couple au milieu de chaque intervalle (θ, θ'). Pour ce faire on actualisera d'abord l'angle à moitié seulement, puis on actualisera la vitesse angulaire et enfin on terminera l'actualisation de l'angle. Pour actualiser l'une ou l'autre des deux fonctions ω et θ on prendra la dernière valeur calculée de l'autre .



- calcul de $\Delta\theta$, pour $\Delta t/2$, utilisant la dernière valeur de ω , et modification de θ .
- calcul de $\Delta\omega$ pour Δt, utilisant la dernière valeur de θ , et modification de ω .
- calcul de $\Delta\theta$ pour $\Delta t/2$, utilisant la dernière valeur de ω , et modification de θ .

● EXPERIENCES

On adoptera le système rationalisé d'unités et on fixera la longueur à 1 mètre; la valeur de g sera prise également à 9.81 ou 9.8088, peu importe.

Essayer d'abord le programme avec un angle θ_o de départ égal à 0.001 radian, dans les conditions de petites oscillations.

On s'intéressera au temps total (la demi-période) en secondes mis pour la descente et la remontée utilisant l'algorithme dans sa version corrigée mais non "raffinée"; de la comparaison du temps de descente et du temps de remontée (obtenu par différence), on déduira une fourchette sous la forme $\overline{\quad} \pm \_$.

Comparer avec la valeur théorique $\sqrt{\pi}/g$ qui sera fournie plus loin. Utiliser la version "raffinée".

Essayer ensuite un angle θ_o de départ égal à 0.01, 0.1 et même 1 radian .

Si on symbolise par A et a l'actualisation complète (A=A+Q*U) et par moitié (A=A+Q* U/2) de l'angle et par Q celle de la vitesse angulaire, laissant entre parenthèses ce qui est inopérant (parce que Q= 0 au début), l'algorithme utilisant θ' et le nouveau proposé donnent respectivement les successions d'opérations suivantes

La seule différence est un décalage; le second s'obtient à partir du premier en ajoutant U/2 au temps, ce qui peut se faire par

```
15  T=U/2
```

Cependant le test final en fin de remontée portant sur Q et non A , il faudra récupérer U/2 par

```
90  PRINT  T+L*Q/G/SIN(A) - U/2
```

Si on préfère, on pourra éviter cette analyse, laissant les lignes 15 et 90 comme avant et remplaçant 50 par

```
35  A=A+Q* U/2
50  A=A+Q*U/2
```

On répondra toujours L=1 et d'abord A=.001 .

Avec une calculatrice SHARP on choisira essentiellement U=0.01. On essaiera aussi un calcul grossier avec U=0.1 et, si on est patient, un calcul plus fin avec U=0.001 ou 0.002.

La première valeur affichée est le temps de la descente seule; la seconde est celle du temps total de descente et de remontée.

Commenter le principe d'isochronisme des petites oscillations. Vérifier si le facteur correctif $(1 + \frac{\theta_0^2}{16})$ donné par les livres de Physique convient.

● VARIANTE (facultative)

Si on évalue l'énergie totale on obtient

$$\frac{1}{2} m l^2 \omega^2 - mgl\cos\theta = -mgl\cos\theta_0 ,$$

ce qui donne

$$\omega^2 = \frac{2g}{l}(\cos\theta - \cos\theta_0) .$$

Après avoir actualisé θ en ajoutant $\omega\Delta t$, on peut se servir de cette formule pour actualiser ω, le signe de ω étant déterminé par celui de θ_0 pour la première partie du mouvement.

● AMORTISSEMENT

On va maintenant tenir compte de la résistance de l'air qui, aux vitesses considérées, est simplement proportionnelle à la vitesse.

Au couple de rappel déjà considéré il faudra donc ajouter un couple d'amortissement donné par

$$-k\omega .$$

L'actualisation de ω se fera ainsi par

$$\omega' = \omega - (\frac{g}{l}\sin\theta + \frac{k}{ml^2}\omega) .$$

L'angle θ_1 de la fin de la phase de remontée n'est plus égal en valeur absolue à l'angle θ_0 de départ.

Reprendre les expériences en présence d'un petit facteur d'amortissement k/ml^2, en notant la demi-période et l'amplitude θ_1 en fin de remontée.

Commenter les résultats.

Si la variable K est utilisée par k/ml^2 , on ajoutera simplement INPUT "AMORT "; K ligne 10 et on transformera la ligne 40 en

```
40  Q = Q - (G * SIN(A) / L + K * Q) * U
```

On pourra supprimer les lignes 25, 60, 70 et ajouter

```
100  PRINT A
```

On se contentera ici de U = 0.01 et on essaiera diverses valeurs de K de 0.1 à 0.5 ainsi que par curiosité K = 10.

Réponses et solutions

Pour obtenir un encadrement, le programme complet sera

```
10  G=9.81 : INPUT "L="; L, "K="; K, "U="; U
15  T =0
20  Q=0 : INPUT "ANGLE"; A : A=-A
25  F=0
30  T=T+U
40  R=Q: Q=Q - (G*SIN(A)/L+K*Q) * U
50  A=A+(Q+R)* U/2
60  IF  A< 0 THEN 30
70  IF F=0 PRINT T-A/Q : F=1
80  IF Q>0 THEN 30
90  PRINT T+L* Q/G/SIN(A)
100 PRINT A
```

Pour une valeur précise on modifie les lignes suivantes

```
15  T=U/2
40  Q=Q -(G* SIN(A)/L+K* Q) *U
50  A=A+Q* U
90  PRINT  T + L*Q/G/SIN(A) -U/2
```

ou

```
35  A=A+Q* U/2
40  Q=Q - (G* SIN(A)/L + K*Q)*U
50  A=A +Q* U/2
```

On introduira K=0 pour le pendule non amorti.

HUITIEME

SEMAINE

SIMULATION D'UN PENDULE : suite

● LA SOLUTION MATHEMATIQUE

Nous avons jusqu'ici utilisé des relations approchées qui le sont d'autant mieux que l'intervalle de temps Δt est plus petit à savoir

$$\frac{\Delta\omega}{\Delta t} = -\frac{g}{l}\sin\theta$$

d'une part et

$$\frac{\Delta\theta}{\Delta t} = \omega \quad ,$$

d'autre part.

Ces deux relations deviennent exactes à la limite, c'est à dire lorsqu'on remplace Δt par l'accroissement infinitésimal dt, les taux d'accroissement devenant des dérivées. On obtient alors

$$\frac{d\omega}{dt} = -\frac{g}{l}\sin\theta$$

et

$$\frac{d\theta}{dt} = \omega \ ,$$

d'où

$$\frac{d}{dt}\left(\frac{d}{dt}\theta\right) = -\frac{g}{l}\sin\theta \ ,$$

c'est à dire encore

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin\theta \ ,$$

où on reconnait au premier membre la dérivée seconde de la fonction θ .

Une telle relation, qui lie les valeurs de la fonction $\theta(t)$ à celles de certaines de ses dérivées s'appelle une équation différentielle. L'équation ci-dessus est dite d'ordre 2 car elle ne met en jeu que des dérivées d'ordre au plus 2.

Il y a plusieurs manières de résoudre les équations différentielles. L'une consiste à chercher des formules explicites, utilisant des fonctions classiques, qui vérifient la relation considérée. Lorsque ce n'est pas possible, et c'est justement le cas dans notre exemple, on peut chercher des solutions approchées. Une possibilité est précisément de faire des approximations du type de celles que nous avons faites dans la première partie du chapitre.

Cependant le passage par une équation différentielle a des avantages ; il permet d'utiliser des méthodes générales pour effectuer lesdites approximations, dont certaines sont très performantes.

● L'EQUATION LINEARISEE

Une autre possibilité pour obtenir des solutions approchées est de modifier l'équation elle-même de manière à obtenir une équation voisine dont on sache trouver les solutions.

Ainsi pour de petites valeurs de θ (donc de petites oscillations du pendule) on sait que la fonction $\sin\theta$ n'est pas très différente de la fonction elle-même. Comme nous l'avons déja vu, on a pour $\theta \leq 0$ l'inégalité

$$\theta \leqslant \sin\theta \leqslant \theta - \frac{\theta^3}{3} .$$

Par exemple, pour $|\theta| \leqslant 10^{-2}$ en radians, l'erreur absolue commise en remplaçant $\sin\theta$ par θ est inférieure à 2.10^{-7} et l'erreur relative, quotient de la première par θ est inférieure à 2.10^{-5}.

Cela signifie encore que si on lâche le pendule dans une position définie par l'angle $\theta_o < 0$ avec $|\theta_o| \leqslant 10^{-2}$ on aura pendant la phase de descente du pendule

$$\theta \leqslant \sin\theta \leqslant (1-\varepsilon)\theta$$

avec $\varepsilon = 10^{-4}/6$.

On pourra alors comparer la solution de l'équation vraie

$$\frac{d^2\theta}{dt^2} = -k \sin\theta$$

à celles des solutions voisines

$$\frac{d^2\theta'}{dt^2} = -k\theta' \quad \text{, dite équation linéarisée}$$

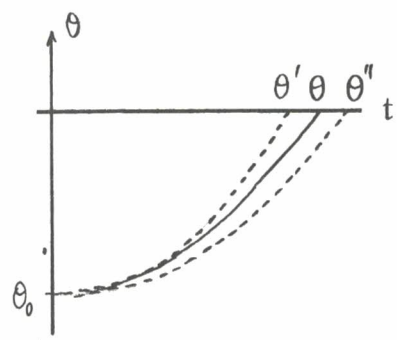
et

$$\frac{d^2\theta''}{dt^2} = -k(1-\varepsilon)\theta'' \ .$$

Pendant la phase de descente on aura

$$\theta''(t) \leqslant \theta(t) \leqslant \theta'(t) \ .$$

Nous ne formaliserons pas la démonstration de ce fait. Elle repose sur un argument que nous avons déjà utilisé : les fonctions θ, θ', et θ'' partent de la même valeur θ_0 avec une dérivée, c'est à dire une vitesse, nulle. Les inégalités sur les dérivées secondes sont des inégalités sur les accélérations en un même point du parcours. C'est θ' qui accélère le plus (car $\sin\theta$ est négatif) et avancera donc le plus vite et de même θ'' qui accélère le moins et avancera le plus lentement



● EXPERIENCES

Reprendre les calculs numériques de la première partie avec θ au lieu de $\sin\theta$. Comparer.

● SOLUTION DE L'EQUATION DIFFERENTIELLE LINEARISEE

On vérifie que la fonction

$$\theta(t) = \theta_0 \cos \omega t$$

vérifie l'équation différentielle

$$\frac{d^2\theta}{dt^2} = -k\theta$$

pour $\omega = \sqrt{k}$, avec les conditions initiales $\theta(0) = \theta_0$ et $\frac{d\theta}{dt}(0) = 0$. On peut montrer que c'est la seule.

En étudiant simplement les variations de la fonction cosinus, on constate aussitôt que le total de descente et de remontée, c'est à dire la demi-période, est donnée par

$$\frac{\pi}{\omega} = \frac{\pi}{\sqrt{k}} .$$

On obtient alors un encadrement de la demi-période du mouvement réel en donnant à k les valeurs $\frac{g}{l}$ et $(1-\varepsilon)\frac{g}{l}$; elle sera comprise entre

$$\pi\sqrt{\frac{l}{g}} \quad \text{et} \quad \pi\sqrt{\frac{l}{(1-\varepsilon)g}} ,$$

le second terme étant lui-même approximativement égal à $\pi\sqrt{\frac{l}{g}} . (1 + \frac{\varepsilon}{2})$. On obtient une valeur supérieure à celle qui correspond à l'équation linéarisée et d'autant plus proche que θ_0 et donc ε est petit.

● RECREATION

Adapter le programme du pendule pour résoudre l'équation différentielle

$$\frac{d^2\theta}{dt^2} = -\theta ,$$

avec conditions initiales $\theta(0) = -1$, $\frac{d\theta}{dt}(0) = 0$, dont la solution est

$$\theta = -\cos t .$$

En déduire une valeur de π avec six chiffres significatifs.

Réponses et solutions

On obtient une valeur approchée de π avec le programme suivant

```
200  INPUT U
210  A=-1 : Q= 0: T=0
220  T=T+U
230  Q=Q - A*U
240  A=A+ Q*U
250  IF Q<0 THEN 220
260  PRINT T+Q/A
270  END
```

Avec U= 0.001 on a 7 chiffres significatifs.

NEUVIEME

SEMAINE

LES BONS COMPTES

Ce chapitre aborde quelques applications pratiques de l'informatique. A l'inverse de ceux du début , on part de problèmes réalistes (voire réels) dont on présente une solution commentée. Ce sera aussi une occasion de manipuler pour la première fois des tableaux.

● UN PROBLEME DE REPARTITION

Imaginons un petit immeuble de six appartements (RdC, 1er étage,..., 5ème étage) en copropriété. Les charges y sont regroupées en cinq rubriques

1 : dépenses générales
2 : dépenses d'entretien
3 : dépenses de chauffage
4 : dépenses d'ascenseur
5 : dépenses de garage .

Pour chaque rubrique les charges sont réparties en 10.000 èmes entre les les 6 appartements suivant le relevé ci-dessous

rubrique / étage	1	2	3	4	5
0	433	490	500	45	0
1	1879	1902	1900	1350	1712
2	1873	1902	1900	1670	1657
3	2069	1902	1900	1991	3305
4	1879	1902	1900	2312	1712
5	1867	1902	1900	2632	1614

On souhaite établir pour chaque appartement un relevé des charges par trimestre se présentant sous la forme suivante

DEPENSES		REZ DE CH.
DEP. GENE.		DIX MILL. 433.
VID ORD	270.24	11.70
EDF 125	350.15	15.16
EAU ASS	1623.82	70.31
ENTRET	113.10	4.90
ENV EXT	8.00	0.35
CLEFS	- 50.00	-2.17
TOTAL=	2315.31	TOTAL= 100.25
DEP. CHAUF.		DIX MILL. 500
FUEL	8514.14	425.71
EDF 137	864.38	43.22
TOTAL=	9378.52	TOTAL= 468.93
DEP. ASC.		DIX MILL. 45
ASCENS	3092.56	13.92
EDF 163	509.41	2.29
TOTAL=	3601.97	TOTAL= 16.21
TOTAL GENE	15295.80	TOTAL GENE 585.39

On introduira les dépenses du trimestre en indiquant :
- la nature,
- le montant en francs,
- le numéro de la rubrique.

● QUELQUES MOTS SUR LA PARTIE ENTIERE

Lorsqu'on multiplie une somme comportant des centimes par une fraction telle que 1789/10000 pour obtenir la part d'une dépense incombant à un copropriétaire donné, on trouve le plus souvent des fractions de centimes. Il faut donc arrondir.

Pour simplifier nous convertirons toujours les sommes en centimes de façon à travailler sur des nombres entiers.

L'arrondi simple consiste à remplacer un nombre réel par le nombre entier le plus proche. Il se fait en utilisant la fonction partie entière, notée INT (de l'anglais integer) en BASIC.

La fonction INT donne l'entier immédiatement plus petit; pour obtenir l'entier le plus proche de X on peut utiliser

$$\text{INT}(X+0.5)\,.$$

Noter que INT réduit à 0 tous les nombres réels X tels que

$$0 \leqslant X < 1\,.$$

Si elle a été correctement définie (il peut y avoir un problème avec certains BASIC) elle a la propriété

$$\text{INT}(X+N) = \text{INT}(X)+N\ ,$$

pour N entier positif ou négatif. En particulier on a par exemple pour

$$X = -4 + 0.3 = -3.7$$

la valeur

$$\text{INT}(X) = -4 + \text{INT}(0.3) = -4$$

et non -3 comme on pourrait le croire!

● ARRONDIS SIMULTANES

Considérons un nombre entier T (somme totale en centimes) à diviser en 6 parts $P_0, P_1, \ldots, P_5$ conformément à des fractions $F_0, F_1, \ldots, F_5$. La première idée consiste à multiplier, pour $i = 0, 1, \ldots, 5$ le nombre T par la fraction F_i et à arrondir le résultat pour déterminer P_i. Malheureusement en opérant de cette manière la somme des parts n'est pas toujours égale au nombre T de départ. Même si la différence n'est que de quelques centimes, ce peut être très désagréable pour la bonne tenue de la comptabilité.

Si par exemple les fractions $F_0, F_1, \ldots, F_5$ sont toutes égales à 0.1333... et si $T = 10$, les parts obtenues de cette manière sont obtenues en arrondissant 1.333..., ce qui donne 1 ; le total des parts n'est que 6 .

On peut s'en tirer en opérant ainsi pour $R_0, R_1, \ldots, R_4$ et en calculant R_5 par différence avec T. Dans l'exemple précédent cela donnerait

$$P_0 = 1,\quad P_1 = 1,\quad P_2 = 1,\quad P_3 = 1,\quad P_4 = 1,\quad P_5 = 5\,.$$

On voit alors que la dernière part peut s'écarter notablement de la valeur théorique.

Essayer la fonction INT en mode direct sur diverses valeurs, dont

INT (4+0.3)

INT (-4+0.3)

Vérifier que INT (X+0.5) donne la valeur arrondie que l'on attend. On peut gagner un peu de temps et de place en écrivant .5 au lieu de 0.5.

Lorsqu'on n'est pas très sûr de X, il peut être utile d'ajouter une petite quantité comme E-9 avant de prendre la partie entière. Si par exemple on attend la valeur 1, mais que par suite d'imprécisions dans les calculs

X = 0.999999999 ,

cela permet de retrouver 1 dans la partie entière.

Essayer INT (1/3*3)

INT (3*1/3)

INT (1/3*3+E-9)

En Pascal, on affecte à la variable entière n la partie entière de la variable réelle x par l'instruction

n := trunc (x)

qui utilise la fonction trunc . Cependant cela ne fonctionne pas pour des valeurs de x supérieures à 32767.

Attention! La définition pour $x < 0$ n'est pas toujours correcte.

On dispose aussi de la fonction round qui détermine directement l'entier le plus proche, avec les mêmes restrictions.

Voici une méthode qui garantit le respect du total, un écart par rapport à la valeur théorique inférieur à 1 pour chaque part, et une égalité des chances; elle n'est pas la meilleure possible mais a l'avantage supplémentaire de la simplicité.

Une première façon de la présenter est la suivante : au lieu de calculer directement les parts, calculons les parts cumulées : la première T_1 qui sera aussi P_o, la somme T_2 des deux premières, la somme T_3 des trois premières... jusqu'à T_6 qui sera T. On obtiendra les parts par différence. Si on convient pour l'harmonie des formules que $T_o = 0$, on aura pour $i = 0, 1, \ldots, 5$ simplement

$$P_i = T_{i+1} - T_i \ .$$

Quant aux T_i on les obtiendra en arrondissant les valeurs théoriques

$$(F_o + \ldots + F_{i-1})\, T \ .$$

Il est facile de voir que les propriétés annoncées sont respectées. D'une part le total est $T_o + (T_1 - T_o) + \ldots + (T_4 - T_5) = T_6 = T$, et d'autre part chaque opération d'arrondi est génératrice d'une erreur au plus égale à 0.5 , ce qui, par différence, donne une erreur au plus égale à 1.

Une autre manière de présenter les choses va se révéler plus facile à adapter. Remarquons que le remplacement de la valeur théorique $F_o T$ par la valeur arrondie P_o pour la première part génère une erreur d'arrondi $E_o = F_o T - P_o$. Cette erreur est la fraction non encore répartie de la somme de départ T si elle est positive, ou répartie en trop si elle est négative. Avant d'arrondir $F_1 T$ pour la seconde part, on ajoutera l'erreur d'arrondi E_o . De cette manière on obtiendra P_1 et on trouveraune nouvelle erreur d'arrondi qui sera maintenant

$$E_1 = E_o + F_1 T - P_1 \ .$$

C'est encore, suivant le signe, la fraction non encore répartie ou au contraire répartie en trop. Ainsi de suite ...

A chaque stade l'erreur d'arrondi est au plus 0.5 . A la fin de la répartition il en sera de même. Mais alors, comme on aura réparti une somme entière en parties entières, l'erreur finale ne peut être qu'entière; c'est donc forcément zéro.

On vérifie de même que l'écart entre P_i et $F_i T$ est au plus de 1. On vérifie encore que le résultat est le même que précédemment : le démontrer soigneusement (en constatant par exemple que $F_1 T + E_o = F_1 T + F_o T - P_o$ s'arrondit en arrondissant $F_1 T + F_o T = T_2$ et lui retranchant $P_o = T_1$) ou le vérifier sur

Essayer le petit programme suivant

```
10  DIM F(5)
20  FOR I=0 TO 5 : INPUT F(I) : NEXT I
30  INPUT T
40  F=0 : U=0: E=0
50  FOR I=0 TO 5
60  F=F+F(I) : V=INT(T*F+0.5)
      :P=V-U : U=V
70  X=T*F(I) : Q=INT(X+E+0.5)
      :E=E+X-Q
80  PRINT P,Q
90  NEXT I
100 GOTO 30
```

On introduit d'abord 6 fractions $F(_0), \ldots, F(5)$ dont la somme est exactement 1. On essaie ensuite la répartition de différentes valeurs de T. La ligne 60 s'inspire de la première présentation en calculant d'abord la fraction cumulée F (initialisée en 40) qui sert au calcul de la part cumulée précédente U (initialisée en 40 et actualisée par U = V). La ligne 70 s'inspire en revanche de la seconde présentation. Le programme imprime sur la même ligne les deux résultats. (Essayer par exemple les fractions 1/21, 2/21, 3/21, 4/21, 5/21, 6/21, puis une valeur quelconque pour T ; appuyer sur [ENTER] après chaque impression pour relancer le programme).

quelques exemples.

L'avantage de la seconde présentation apparaît lorsqu'on doit répartir suivant les mêmes fractions $F_0, F_1, \ldots, F_5$ non pas une somme T mais plusieurs.

● TABLEAUX

Dans notre problème, nous allons devoir effectuer un même traitement sur chaque dépense, et à plusieurs reprises, ce qui nécessite d'avoir la totalité des dépenses présente en mémoire. Dans un tel cas il est souhaitable d'organiser les données en tableau.

Un tableau de variables est ce qu'en mathématiques on nomme une suite de variables; un tableau de dimension n est une suite de n+1 variables numérotées de 0 à n.

Un tableau est désigné comme une variable simple par un nom (ou identificateur) qui peut être une lettre de l'alphabet s'il s'agit d'un tableau de variables numériques, suivie du signe $ s'il s'agit d'un tableau de chaînes. L'élément de rang I du tableau A est désigné par

A(I).

Avant d'utiliser pour la première fois un tableau en BASIC, il faut le "dimensionner" par l'instruction DIM. Par exemple DIM A(24) crée un tableau de 25 variables en réservant la place nécessaire en mémoire.

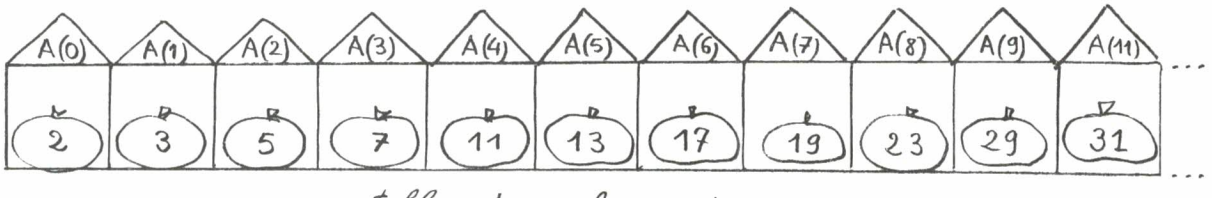


tableau des nombres premiers

● ORGANISATION DES DONNEES

Supposons que le nombre de dépenses au cours d'un trimestre n'excède jamais 25. Pour décrire ces dépenses nous allons utiliser plusieurs tableaux dimensionnés à 24 :

En Pascal, un tableau se dénomme array et se déclare, s'il s'agit par exemple d'un tableau de nombres entiers indexés de 2 à 8, par

a : array [2..8] of integer.

L'élément d'indice i du tableau est désigné par a[i]. Le langage Pascal utilise les crochets là où le langage BASIC utilise les parenthèses; les parenthèses en Pascal servent aux fonctions et procédures avec paramètres.

En mathématiques il n'y a pas de différence essentielle entre une suite et une fonction. La différence réside dans la manièrepratique de les définir: par la donnée de toutes les valeurs (cas d'un tableau) ou par celle d'une formule (cas des fonctions procédures et sous-programmes).

- un tableau N$ de variables de chaîne pour la nature des dépenses
- un tableau M pour le montant
- un tableau R pour la rubrique.

Il est souhaitable de conserver les montants en centimes (alors qu'ils sont introduits en francs) car toutes les opérations d'arrondi porteront sur des centimes. Si l'on veut gagner de la place en mémoire, on peut concentrer les tableaux M et R en un seul, noté encore M, qui contiendra

le montant (en centimes)

+

le numéro de rubrique divisé par 10.

On récupérera le montant en prenant la partie entière et le numéro de rubrique en multipliant la partie fractionnaire par 10. D'un autre côté, les dix-millièmes de répartition seront inscrits dans un tableau T à deux dimensions, la première pour l'étage, la seconde pour la rubrique. Un tel tableau peut toujours être ramené à un tableau à une seule dimension qui sera le produit des deux. Dans notre exemple on remplace simplement T(E,R) par T(5*E+R).

● LE PROGRAMME

Nous laissons de côté la question de la présentation des résultats parce que la réponse dépend beaucoup du matériel que l'on utilise.

Après dimensionnement des tableaux par

```
10  DIM N$(24) : DIM M(24) : DIM R(24) : DIM T(5,5)
```

on remplit, à partir des données DATA, le tableau T(E,R) par deux boucles convenablement emboitées

```
20  FOR E=0 TO 5 : FOR R=1 TO 5 : READ T(E,R) : NEXT R : NEXT E
```

on réserve un petit tableau A pour les erreurs d'arrondi, que l'on initialise à 0 ; il faut une variable par rubrique car les fractions diffèrent d'une rubrique à l'autre

```
30  DIM A(5) : FOR R=1 TO 5 : A(R)=0 : NEXT R
```

les données sont introduites par la boucle suivante, arrêtée par une réponse vide à la première question

```
100  I = 0
```

Pour dimensionner un tableau N$ de 25 chaînes dont la longueur est au maximum 7 on écrira

DIM N$(24) * 7

En Pascal le programme se présentera de la même manière mais on doit être très soigneux quant au type de variables.

* Les montants, exprimés en centimes pourront être des entiers longs (un entier ordinaire ne permet pas de dépasser +32767), alors que les dix-millièmes seront des entiers ordinaires. On effectuera le produit m * t du montant m par le nombre t de dix-millièmes (prévoir des entiers à 14 ou 15 chiffres) et on tronquera en ajoutant 5000 et en divisant par 10000. Il s'agit de la division entière div qui fournit la partie entière du quotient. On ne manipulera ainsi que des nombres entiers, ce qui est une garantie absolue.

La conversion en francs se fera ensuite en plaçant le point décimal dans le montant transformé en chaîne de caractères par str.

```
110 INPUT "NATURE "; N$ : IF N$<>"" THEN INPUT "MONTANT "; M,
    "RUBRIQUE "; R : N$(I) = N$ : M(I) = M : R(I) = R : I =I+1: GOTO 110
```

le calcul de la part de chaque dépense se fait dans les boucles emboitées

```
200 FOR E=0 TO 5 : PRINT "ETAGE "; E : T=0
210 FOR R=1 TO 5 : PRINT "RUBRIQUE " ; R
220 FOR J=0 TO I-1 : IF R(J)<> R THEN 270
230 PRINT N$(J); "   " ; M(J)/100
240 X=M(J) * T(E,R)/10000
250 P=INT(X+A(R)+0.5) : A(R)=A(R) + X - P : T=T+P
260 PRINT "PART "; P/100
270 NEXT J : PRINT
280 NEXT R : PRINT "TOTAL "; T : PRINT
290 NEXT E
```

Noter la ligne 250 qui ajoute l'erreur d'arrondi courante avant arrondi et actualise l'erreur d'arrondi.

On aurait pu se contenter de deux boucles en laissant les dépenses en désordre.

Si on introduit les montants en francs, il faut écrire `M(I)=M* 100` à la ligne 110.

DIXIEME

SEMAINE

5

LES BONS COMPTES : suite

● UN PROBLEME DE MISE A JOUR

Le possesseur d'un compte bancaire veut connaître avec une assez bonne précision la situation de son compte pour lequel la banque ne fournit que des relevés espacés . Il s'astreint seulement à :

- inscrire chaque chèque tiré, virement, retrait automatique ainsi que chaque notification de crédit (salaire...),

- inscrire, dès qu'il en a connaissance, avec la date d'effet, les prélèvements (emprunts, impôts, factures d'électricité ou de téléphone...) au moins pour l'année en cours; l'inscription se fait dans un ordre sans rapport avec les dates d'effet.

Il veut disposer à tout moment, en introduisant la date du jour, du solde de son compte, ainsi que d'un moyen de vérifier la concordance avec les relevés de sa banque.

Pour que le système soit réellement utilisable il ne faut pas que chaque inscription nécessite le branchement d'un téléviseur, le chargement d'un programme et d'un fichier sur cassette ... On a ici un exemple de tâche domestique que les ordinateurs familiaux sont bien souvent incapables de remplir. Comme on ne va pas jusqu'à supposer que chacun dispose d'un ordinateur semi-professionnel toujours "prêt à l'emploi", on s'orientera vers un ordinateur de poche à mémoire permanente, choisi bon marché surtout si cette application doit occuper la totalité de la mémoire et ne laisser plus qu'une simple calculatrice (ce qui n'est déjà pas si mal).

● ORGANISATION DES DONNEES

Les données comporteront deux groupes :

- d'une part les montants des opérations récentes n'ayant pas encore figuré sur un relevé de la banque, sans indication de date,

Pour les applications pratiques, les calculatrices SHARP disposent du mode DEF. Un programme dont la première ligne commence par une lettre parmi les 18 situées dans le quart bas et gauche du clavier (SPC compris), tel que

```
10 "A" ...
```

peut être lancé sans RUN ni GOTO, simplement par l'appui sur la touche [DEF] suivi de la lettre à laquelle il a été fait allusion, c'est à dire A dans notre exemple.

On peut même en profiter pour faire enregistrer dans une variable le contenu de l'écran, grâce à l'instruction AREAD ; par exemple

```
10 "A" AREAD X
```

donne à la variable X la valeur (numérique) affichée à l'écran lorsque le programme est lancé par [DEF] [A] .

Le fait de n'avoir jamais à frapper la touche [ENTER] garantit contre le risque d'effacer des tableaux en mémoire par un RUN intempestif.

- d'autre part un échéancier des opérations à venir, comportant à la fois les montants et les dates.

Supposons que l'on estime à 13 et à 25 le nombre maximum des données appartenant respectivement au premier et au second groupe. On prévoira donc des tableaux R de dimension 12 pour les opérations récentes et V, D de dimension 24 pour celles à venir.

Si l'on veut utiliser le même traitement pour les débits et les crédits, il faut les introduire avec des signes différents, logiquement les crédits devant être précédés de + et les débits de -. Cependant les débits étant plus nombreux, il vaudrait mieux ne pas leur imposer de signe.

On peut mettre la date dans une seule variable numérique. Si on dispose par exemple de variables numériques réelles on peut leur donner la valeur

$$\text{n° du mois} + \frac{\text{jour du mois}}{100} \quad ;$$

de cette manière le 15 février se présentera sous la forme

2.15 .

L'avantage de mettre le mois d'abord est que l'ordre sur les dates est respecté. Bien sûr rien n'empêche d'introduire les dates d'une manière plus naturelle.

● LA MISE A JOUR

Le type de travail que le programme devra faire sera, sur un tableau, de supprimer certaines données et d'en ajouter d'autres. Prenons le cas d'un tableau A de variables numériques de dimension N, tableau dans lequel les N+1 possibilités ne sont pas toutes utilisées. Supposons que le soient celles de rang compris entre 0 et I. Ajouter de nouvelles données est facile; il suffit d'augmenter chaque fois d'une unité la valeur de I et d'affecter la valeur à A(I). En revanche si on veut supprimer des données qui ne soient pas les dernières, il va falloir déplacer toutes celles qui sont à la suite.

Supposons que la valeur 0 ne soit jamais une donnée (c'est le cas pour notre problème : un débit ou un crédit de montant 0 sont sans intérêt). On peut facilement supprimer une donnée en lui attribuant la valeur 0. Cependant si on ajoute toujours les nouvelles données à la suite de la dernière, la valeur de I croîtra indéfiniment et on ne pourra s'en tirer qu'avec un tableau de dimension

Comme faire précéder les crédits d'un signe - serait étrange, et risquerait de provoquer des erreurs, on pourra utiliser la petite astuce suivante sur les calculatrices SHARP : on demandera d'introduire les crédits en terminant par la lettre C, sachant que la variable C aura été fixée à -1. Le BASIC de ces calculatrices autorise dans certains cas le produit sans signe * ; il multipliera la valeur introduite par -1.

En Pascal on dispose de la structure d'enregistrement, nommée record, pour une donnée associant comme dans le cas présent un nombre et une date, la date étant elle-même constituée d'un nombre et d'une chaîne de caractères (le nom du mois).

Par exemple

```
échéance : record montant: long integer;
                  date: record jour:...
                               mois: ...
                        end
           end
```

très grande, gaspillant beaucoup de place en mémoire pour un petit nombre de vraies données. Une solution consiste à ajouter les données nouvelles dans les places laissées libres.

Ainsi , pour la suppression de rang I on écrira seulement

A (I) = 0 ,

alors que pour l'addition d'une donnée nouvelle on pourra utiliser les lignes de programme suivantes

```
I=0
IF A(I)<>0 THEN I=I+1 : GOTO
A(I)=
```

ou

```
I=-1
I=I+1
IF A(I)<> 0 THEN
A(I)=
```

On peut améliorer le procédé en ne repartant pas chaque fois de 0 mais de la dernière valeur de I utilisée; il faudra alors revenir au début lorsqu'on aura atteint le dernier rang possible du tableau, soit 13 dans notre exemple. On écrira donc

```
IF I=13 THEN I=0
IF A(I)<>0 THEN I=I+1 : GOTO
A(I)=
```

ou

```
IF A(I)=0 THEN
I=I+1
IF I=13 THEN I=0
IF A(I)<>0 THEN
A(I)=
```

Ce programme boucle indéfiniment lorsque le tableau est rempli . On peut s'en prémunir en conservant la valeur antérieure de I dans une variable J et en arrêtant le programme lorsque l'on retrouve I=J. Cela conduit ainsi à écrire

```
J=I
IF I=13 THEN I=0
IF A(I)<>0 THEN I=I+1:
    IF I<>J THEN
IF A(I)<>0 PRINT "COMPLET":
    END
A(I)=
```

ou

```
J=I: IF A(I)=0 THEN
I=I+1
IF I=13 THEN I=0
IF I=J PRINT "COMPLET":END
IF A(I)<> 0 THEN
A(I)=
```

On écrira simplement

```
i:=0;
while a[i]<>0 do i:=i+1;
a[i]:=
```

ou

```
i:=-1
repeat i:=i+1 until a[i]=0;
a[i]:=
```

La version améliorée serait

```
while a[i]<>0 do begin
  i:=i+1;
  if i=13 then i:=0
end;
a[i]:=
```

ou

```
if a[i]<>0 then
begin
  repeat
    i:=i+1;
    if i=13 then i:=0
  until a[i]=0
end;
a[i]:=
```

Pour se prémunir contre un tableau complet, on contrôlera la dernière boucle par la condition

until (i=j) or (a[i]=0)

Il faudra ensuite regarder ce qui a été cause de l'arrêt de la boucle.

N.B.: Une manière qui peut paraître plus simple est de parcourir le tableau par une boucle du type

```
FOR I=0 TO 12
IF A(I) = 0 THEN A(I) =... : END
NEXT I
PRINT "COMPLET"
END
```

Il y a un inconvénient : si après avoir trouvé la donnée et fait ce qui était prévu on veut passer à autre chose, on provoquera une sortie en cours de boucle; la répétition de telles sorties est génératrice d'erreurs. Pour éviter ce danger il faudrait dans notre exemple faire I = 12, brancher le programme sur NEXT I et interroger ensuite pour savoir de quelle manière on est sorti de la boucle.

● LE PROGRAMME

Ce ne sera pas exactement un programme mais plusieurs petits programmes qui pourront être appelés séparément.

Ils seront au nombre de quatre, et appelés respectivement pour
- inscrire les opérations récentes, après [DEF] [A]
- effectuer un contrôle à partir des opérations figurant sur un relevé bancaire, après [DEF] [S]
- inscrire les échéances, après [DEF] [Z]
- actualiser le compte en fonction de la dernière date introduite, après [DEF] [X].

Une variable S conservera le "solde" du compte tel qu'il devra normalement apparaître au bas du dernier relevé dont les opérations auront été introduites pour contrôle.

Une variable U conservera le montant réellement "utilisable", compte-tenu des opérations récentes et des échéances passées; ce montant sera affiché après l'inscription d'une opération récente ou l'actualisation.

Lorsqu'on opère dans l'une quelconque des petits programmes il suffit de presser [ENTER] pour le relancer; on appuiera sur [CL] avant d'en demander un autre.

Il faudrait encore apporter quelques améliorations pratiques, faciliter par exemple la correction en cas d'inscription erronée...

L'un des avantages d'un langage structuré comme le langage PASCAL est d'interdire ce genre de sortie. On devrait dans l'exemple donné utiliser une variable booléenne sortie et écrire

```
sortie := false;
for i:= 1 to 12 do ...

    if (sortie = false) and (a[i]= 0) then
    begin
        a[i] := ...;
        sortie := true
    end;
```

Ce serait bien sûr maladroit.

```
100   "A" INPUT "MONT  "; M
110   J=I: IF R(I)=0 THEN 160
120   I=I+1
130   IF I=13 LET I=0
140   IF I=J PRINT "COMPLET": END
150   IF R(I)<>0 THEN 120
160   R(I)=M
170   U=U-M: PRINT "UTIL  "; U: GOTO 100

200   "S" INPUT "MONT  "; M
210   J=I: IF R(I)=M THEN 260
220   J=J-1
230   IF J=-1 LET J=12
240   IF J=1 PRINT "NON TROUVE": END
250   IF R(J)<>M THEN 220
260   I=J: R(J)=0
270   S=S-M: PRINT "SOLDE  "; S: GOTO 200

300   "Z" INPUT "ECHE  "; E
310   INPUT "MONT  "; M
320   L=K: IF V(K)=0 THEN 370
330   K=K+1
340   IF K=25 LET K=0
350   IF K=L PRINT "COMPLET": END
360   IF V(K) THEN 330
370   V(K)=M: D(K)=E
380   PRINT "INSCRIT": GOTO 300

400   "X" INPUT "DATE  "; E
410   FOR L=0 TO 24
420   IF (D(L)<=E)*V(L) LET U=U-V(L): V(L)=0: K=L
430   NEXT L
440   PRINT "UTIL "; U: END

980   END
990   CLEAR: DIM R(12): DIM V(24): DIM D(24): C=-1: END
```

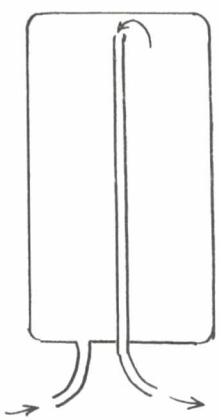
On fera RUN 990 à la première utilisation avant d'avoir inscrit quoi que ce soit.

ONZIEME

SEMAINE

SIMULATION D'UN ECHANGE DE CHALEUR

● UTILISATION D'UN CHAUFFE-EAU A ACCUMULATION

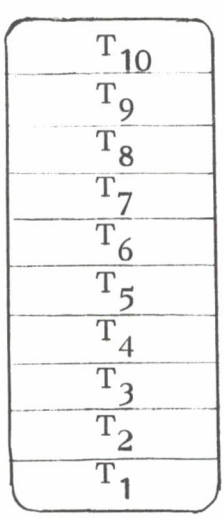


On considère un chauffe-eau constitué d'un ballon cylindrique plus haut que large, placé en position vertivale.

L'eau qu'il contient a été chauffée pendant la nuit à une température T_c. Pendant la journée on tire de l'eau par le haut, cette eau étant remplacée par de l'eau froide à une température T_f arrivant par le bas.

On supposera pour simplifier le ballon parfaitement calorifugé, que les déplacements d'eau se font par couches horizontales sans remous, et enfin que la température à un moment donné est la même pour l'eau d'une même couche horizontale.

On s'intéresse donc uniquement aux échanges de chaleur entre les différentes couches horizontales entre deux utilisations du chauffe-eau.



On shématise la situation à un moment donné en découpant le cylindre en un assez grand nombre de tranches (on en prendra 10 par exemple), chacune étant à une température particulière.

Programmons

La répartition des températures sera représentée par un tableau T ; dans le cas de 10 tranches on le dimensionnera par

```
10  DIM T(10)
```

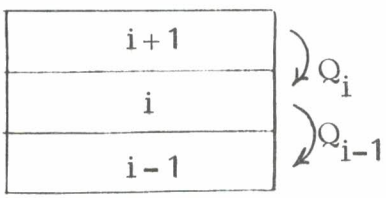
en réservant une tranche fictive T(0) dont l'usage apparaîtra plus loin. L'intervalle Δt sera représenté par une variable U.

Les échanges de chaleur se produiront entre une tranche et ses deux voisines, sauf aux extrémités. Au bout d'un intervalle de temps Δt , une quantité de chaleur Q_i passe de la tranche de rang $i+1$ dans celle de rang i ; elle est proportionnelle à la différence de température $T_{i+1} - T_i$, autrement dit

$$Q_i = K(T_{i+1} - T_i) ,$$

avec un coefficient de proportionnalité K . De même la quantité Q_{i-1} de chaleur qui passe de la tranche de rang i à celle de rang $i-1$ est donnée par

$$Q_{i-1} = K(T_i - T_{i-1})$$



La tranche de rang i reçoit donc pendant le temps Δt

$$Q_i - Q_{i-1} = K(T_{i+1} + T_{i-1} - 2T_i) .$$

La nouvelle température T_i' de la tranche de rang i est modifiée de façon proportionnelle à cet apport de chaleur, avec un coefficient diviseur C , ce qui donne lieu à

$$T_i' = T_i + \frac{Q_i - Q_{i-1}}{C}$$

$$= T_i\left(1 - \frac{2K}{C}\right) + \frac{K}{C} T_{i+1} + \frac{K}{C} T_{i-1} .$$

Aux extrémités on aura de même

$$T_{10} = T_{10}\left(1 - \frac{K}{C}\right) + \frac{K}{C} T_9$$

$$T_1' = T_1\left(1 - \frac{K}{C}\right) + \frac{K}{C} T_2$$

- INTERPRETATION BARYCENTRIQUE

Posons $\frac{K}{C} = k$. La formule générale de modification de la température s'écrit

Programmons

Le chauffage à la température C pendant la nuit du ballon sera représenté dans la boucle suivante qui sert aussi à initialiser la durée D

```
50  D = 0 : FOR I=0 TO 10 : T(I) = C : NEXT
```

Le fait de tirer l'équivalent d'une tranche d'eau chaude et de la remplacer par de l'eau froide à la température F se fait en décalant les tranches vers le haut, chassant la tranche la plus élevée et ajoutant une tranche froide en bas . On donne à T(I) la valeur de T(I-1), mais en commençant par la tranche du haut, pour ne pas détruire des valeurs appelées à servir, ce qui est fait par la boucle

```
70  T(0) = F : FOR I=10 TO 1 STEP -1 : T(I) = T(I-1) : NEXT I ,
```

dans laquelle STEP -1 indique que I décroit de 10 à 1.

$$T'_{i+1} = (1-2k)\,T_i + kT_{i+1} + kT_{i-1} .$$

Ainsi T'_{i+1} apparait comme une combinaison linéaire des nombres T_i, T_{i+1}, T_{i-1}, la somme des coefficients étant

$$1 - 2k + k + k = 1 .$$

C'est ce qu'on appelle une moyenne ou un barycentre. Noter en particulier que si $T_{i+1} = T_{i-1} = T_i$ alors $T'_i = T_i$. On a $k>0$ parce que $K>0$, la chaleur étant cédée par la tranche la plus chaude à la tranche la plus froide.

Soyons plus précis. Le coefficient K est d'autant plus grand que la surface de contact S et l'intervalle de temps Δt sont plus grands et que l'épaisseur Δx d'une couche est plus petite. On a

$$K = \frac{S.\Delta t}{\Delta x} K_o ,$$

où K_o est la conductivité thermique. De même le coefficient C est d'autant plus grand que la masse $m = \rho S \Delta x$ d'une tranche (où ρ désigne ici la masse spécifique) l'est. Ainsi

$$C = \rho S \Delta x \; C_o ,$$

où C_o est la chaleur massique. Donc

$$k = \frac{K}{C} = \frac{\Delta t}{(\Delta x)^2} \cdot \frac{K_o}{\rho C} .$$

On peut adapter k en jouant sur l'épaisseur Δx ou l'intervalle de temps Δt, supposés bien entendu tous les deux petits pour justifier les raisonnements.

Une valeur de k à la fois commode pour les calculs et pour équilibrer Δx et Δt est simplement

$$k = \frac{1}{4} .$$

● EXPERIENCES

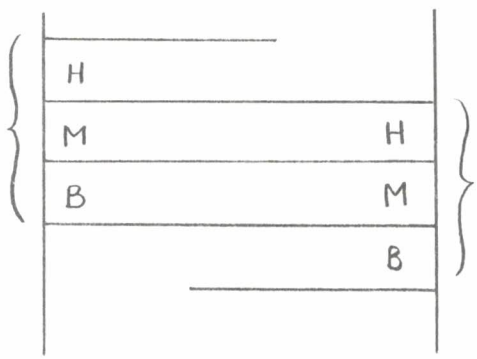
Après une nuit, la température de tout le ballon est T_c. On tire l'équivalent d'un certain nombre de tranches qui sont remplacées par de l'eau froide à la température T_f et on étudie l'évolution de la répartition des températures. On choisira par exemple $T_c = 65°$, $T_f = 15°$ (mais aussi 25° et 5° pour voir l'influence de la saison). La durée sera évaluée dans une unité de temps que l'on pourrait

On se place dans le cas où $k = 1/4$.

Pour modifier la valeur de T(I) à l'aide de T(I+1) et T(I-1) suivant les formules obtenues, on doit prendre garde de ne pas détruire une valeur qui aurait encore à servir. Par exemple, si on écrit

$$T(I) = T(I)/2 + T(I+1)/4 + T(I-1)/4 \ ,$$

pour une certaine valeur de I, on ne pourra plus utiliser la formule pour la valeur I+1 ou la valeur I-1, car pour ces valeurs, on aura besoin de T(I) qui aura été modifié. Pour chaque valeur de I on notera dans des variables H, M, B les températures des tranches de rang I+1 (haute), I (moyenne) et I-1(basse).



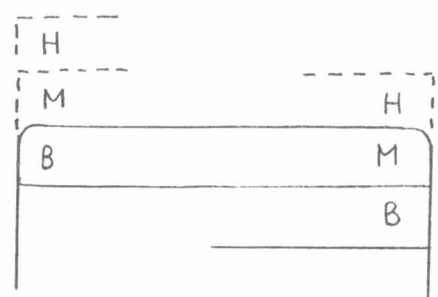
Aux extrémités, on ajoutera une tranche fictive à la même température que la plus haute ou la plus basse pour ne pas avoir à utiliser une formule spéciale.

Si on modifie les températures en descendant, on passera d'une tranche à la suivante en diminuant I et en écrivant

```
130  H = M : M = B : B = T(I-1) : T(I)= M/2 + H/4 + B/4
```

L'ancienne valeur de T(I) reste dans la variable M.

Cette ligne sera évidemment prise dans une boucle




```
120  FOR I=10 TO 1 STEP -1
       "
140  NEXT I
```

initialisée par

```
100  M=T(10) : B = M
```

et suivie de la répétition de la tranche du bas

```
150  T(0) = T(1)
```

Enfin on fait afficher, en 110 après initialisation, la température de l'eau, on décompte en 160 la durée par D= D+ U et on reprend en 170 le cycle d'échange par GOTO 100.

Avec une calculatrice SHARP on utilisera le mode DEF en complétant le début de certaines lignes comme

```
50  "A"...
70  "S"...
100 "D"...
```

préciser en étalonnant par comparaison avec un chauffe-eau réel dont on souhaite prévoir le fonctionnement. On tracera la variation de la température de l'eau prête à sortir en fonction du temps.

Réponses et solutions

Le programme complet pour une calculatrice SHARP sera

```
10  DIMT(10) : U=1
20  INPUT "C="; C, "F="; F
30  END
50  "A" D=0: FOR I=0 T0 10: T(I) =C : NEXT I
60  END
70  "S" T(0)=F : FOR I=10 TO 1 STEP -1 : T(I) = T(I-1) : NEXT I
80  END
100 "D" M=T(10) : B=M
110 PAUSE D; USING "####.##"; M: USING
120 FOR I=10  TO 1 STEP -1
130 H=M : M=B: B=T(I-1) : T(I)= M/2+H/4 + B/4
140 NEXT I
150 T(0) = T(1)
160 D=D +U
170 G OTO 100
```

Après RUN pour dimensionner, on fera [DEF] [A] pour une nuit, [DEF] [S] autant de fois que l'on voudra retirer l'équivalent d'une tranche d'eau, et [DEF] [D] pour lancer l'échange de chaleur qui sera arrêté par [BRK] .

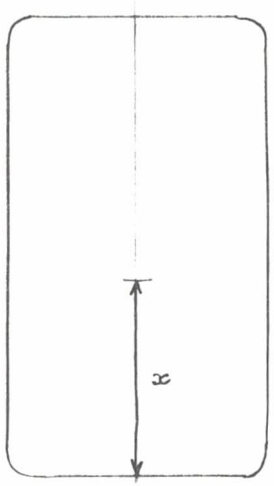
DOUZIEME

SEMAINE

SIMULATION D'UN ECHANGE DE CHALEUR : suite

● PASSAGE AU CONTINU

Si, au lieu de découper le ballon en un nombre fini de tranches horizontales pour lesquelles la température est supposée uniforme, nous considérons que cette température, à un instant donné, varie continûment en fonction de la hauteur x mesurée à partir du bas, nous sommes amenés à décrire l'évolution du système par une fonction



$$T(t, x)$$

de deux variables, exprimant la température à l'instant t du point d'altitude x .

La relation faisant passer de T_i à T_i' s'écrira, si le prime indique que l'on passe de t à $t+\Delta t$, si x correspond à i, $x+\Delta x$ à $i+1$ et $x-\Delta x$ à $i-1$, sous la forme

$$T(t+\Delta t, x) - T(t, x) = -2k\,T(t, x) + k\,T(t, x+\Delta x) + k\,T(t, x-\Delta x) .$$

Négligeons pour la discussion mathématique le coefficient $\frac{K_0}{\rho C}$ qui peut toujours être pris égal à un par un choix convenable d'unités et retenons simplement

$$k = \frac{\Delta t}{(\Delta x)^2} .$$

La relation précédente s'écrit encore

$$\frac{T(t+\Delta t, x) - T(t, x)}{\Delta t} = \frac{T(t, x+\Delta x) - 2T(t, x) + T(t, x-\Delta x)}{(\Delta x)^2} .$$

● DERIVEES PARTIELLES

Considérons uniquement les points du ballon situés à une altitude x fixée. La température ne dépend alors plus du temps t . On reconnait au premier membre de la relation ci-dessus un taux de variation qui, lorsqu'on remplace Δt par un infiniment petit dt, fournit une dérivée par rapport à la variable t. Une dérivée par rapport à l'une des variables lorsque l'autre est fixée s'appelle une dérivée partielle . Dans ce cas, on le notera

$$\frac{\partial T}{\partial t} ,$$

avec le symbole ∂ au lieu du d droit, en indiquant la variable par rapport à laquelle on dérive , qui est ici t .

Le second membre est un peu plus délicat à interpréter. Notons tout de suite que tous les termes concernent la même valeur t du temps. On peut considérer cette fois que le temps t a été fixé, et qu'il n'y a plus que la variable x . Si on écrit le second membre sous la forme

$$\frac{1}{\Delta x} \left\{ \frac{T(t, x+\Delta x) - T(t, x)}{\Delta x} - \frac{T(t, x+(-\Delta x)) - T(t, x)}{-\Delta x} \right\}$$

on voit apparaître deux taux d'accroissement, qui tous les deux à la limite fourniront une dérivée partielle

$$\frac{\partial T}{\partial x}$$

au point (t, x). A première vue ce membre se comporte comme une forme indéterminée $\frac{0}{0}$. Allons donc un peu plus loin en prenant un développement limité fourni par la formule de Taylor, toujours pour t fixé, la seule variable étant x . On a

$$T(t, x+\Delta x) = T(t, x) + \Delta x \frac{\partial T}{\partial x} + \frac{(\Delta x)^2}{2} \frac{\partial^2 T}{\partial x^2} + \ldots$$

Ici apparaît la dérivée seconde par rapport à la seule variable x, qui est une dérivée seconde partielle

$$\frac{\partial^2 T}{\partial x^2} .$$

De même

$$T(t, x-\Delta x) = T(t, x) - \Delta x . \frac{\partial T}{\partial x} + \frac{(\Delta x)^2}{2} . \frac{\partial^2 T}{\partial x^2} + \ldots$$

En combinant, tout se simplifie et il ne reste plus que

$$\frac{\partial^2 T}{\partial x^2} + \dots ,$$

où les ... sont là pour désigner des termes d'ordre supérieur, disparaissant à la limite.

On aboutit ainsi à la relation

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} .$$

C'est ce qu'on appelle une équation aux dérivées partielles; dans le cas présent c'est l'équation de la chaleur.

● CONCLUSION

Les équations aux dérivées partielles jouent un rôle capital dans la modélisation des problèmes de la physique. L'apparition de puissants moyens de calcul et le développement de méthodes numériques performantes ontpermis de réaliser la simulation de phénomènes physiques complexes. Cependant il s'agit là de mathématiques difficiles et il n'est pas question ici de donner une idée des méthodes. On pourra s'enorgueillir d'avoir, au début du chapitre, résolu de manière approchée une équation aux dérivées partielles.

Retenons simplement que pour une fonction de deux variables, que nous noterons $f(x, y)$ pour utiliser des noms de variables plus courants, en mathématiques, on peut considérer les dérivées partielles

$$\frac{\partial f}{\partial x} , \quad \frac{\partial f}{\partial y}$$

en chaque point (x, y). Pour chacune de ces fonctions on peut prendre des dérivées partielles

$$\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial x}\right) = \frac{\partial^2 f}{\partial x^2}$$

$$\frac{\partial}{\partial y}\left(\frac{\partial f}{\partial y}\right) = \frac{\partial^2 f}{\partial y^2}$$

mais aussi

$$\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial y}\right) = \frac{\partial^2 f}{\partial x \partial y}$$

$$\frac{\partial}{\partial y}\left(\frac{\partial f}{\partial x}\right) = \frac{\partial^2 f}{\partial y \partial x} \ ;$$

on démontre que les deux dernières $\frac{\partial^2 f}{\partial x \partial y}$ et $\frac{\partial^2 f}{\partial y \partial x}$ sont identiques.

Ces choses savantes ayant été dites, vous êtes dispensés de programmation cette semaine !