QUELQUES OPÉRATIONS DE COMPOSITION MUSICALE

DANS LE SYSTEME INFORMATIQUE DE MAGNUS LINDBERG

Marc CHEMILLIER

30 Avril 1993

I. HARMONIE

A - INTERPOLATION.

L'interpolation entre deux accords consiste à calculer un nombre donné d'accords intermédiaires permettant de passer progressivement de l'un à l'autre, selon une courbe déterminée. Entre chaque note de l'accord initial et la note en même position dans l'accord final, on introduit une succession de notes intermédiaires déterminées par la courbe passant par les deux notes de départ et d'arrivée. Différents types de courbes sont possibles: linéaires, sinusoïdales, etc.

Le calcul des notes intermédiaires, reliant une note de l'accord initial et sa note correspondante dans l'accord final, se fait de la manière suivante. Les accords sont représentés par des listes de codes midi (60 pour do3, 61 pour réb3, etc.). Soient a_0 une note de l'accord initial, b_0 la note correspondante dans l'accord final, et d la différence $d = b_0 - a_0$. Pour i allant de 0 à n-1 (n est le nombre total d'accords de la séquence cherchée), chaque note intermédiaire a_i entre a_0 et b_0 s'obtient par la formule :

$$a_i = a_0 + f\left(\frac{i}{n-1}\right) * d$$

où f est la fonction de l'interpolation. Cette fonction est définie entre 0 et 1, avec f(0) = 0 et f(1) = 1. Plusieurs fonctions sont possibles, par exemple :

f(x) = x pour une interpolation linéaire, ou

$$f(x) = -\left(1 + \sin\left(\frac{1}{2} + \frac{3\pi}{2} * x\right)\right)$$

pour une interpolation sinusoïdale. Les résultats sont arrondis à l'entier le plus proche : $E(a_i + 0.5)$ où E est la partie entière.

La fonction effectuant le calcul est *samples*. Son premier argument est une variable à laquelle sera affectée la liste d'accords obtenue. On précise la fonction d'interpolation en quatrième argument : 1 pour une interpolation linéaire, sinus pour une interpolation sinusoïdale. La fonction est définie dans le fichier **Library:History-functions**, qui contient les fonctions principales du système. Le calcul d'interpolation proprement dit est fait par la fonction *interpolation*, qui est définie dans le fichier **Library:Rhythm-functions**, car l'interpolation est

également utilisée pour des calculs de rythmes. La fonction d'arrondi est définie dans **MyLib:Primitives** qui contient la plupart des fonctions utilitaires du système.

Dans l'exemple ci-après, la variable *s1* contient le résultat de l'interpolation sinusoïdale en 10 étapes entre *accord1* et *accord2*.

```
? (setq accord1 '(38 47 49 52 58 60)
        accord2 '(67 69 75 78 80 89))
= (67 69 75 78 80 89)

? (samples s1 accord1 accord2 10 'sinus)
= ((38 47 49 52 58 60) (39 48 50 53 59 61) (41 50 52 55 61 63) (45 53 56 59 64
67) (50 56 60 63 67 72) (55 60 64 67 71 77) (60 64 69 72 75 82) (64 66 72 75
77 86) (66 68 74 77 79 88) (67 69 75 78 80 89))
```

Outre l'interprète Le_lisp permettant d'effectuer des calculs comme celui qui vient d'être décrit, à l'aide des fonctions du système, l'utilisateur dispose également d'un menu comprenant un ensemble très complet de commandes d'affichage des résultats : notation musicale sur portées pour une séquence d'accords, ou diagramme de type "piano-roll".

B - CONTROLE DES CLASSES.

L'inconvénient de l'interpolation pour le calcul de successions harmoniques est l'absence de contrôle concernant la "qualité" harmonique des accords intermédiaires engendrés par le calcul. Une manière de remédier à cela consiste à faire une correction de la liste d'accords obtenue par interpolation. Chaque accord est remplacé par un accord le plus proche possible, mais auquel on impose d'être d'un "type" donné. Le type d'un accord est défini ci-après, et présente une légère parenté avec la notion de chiffrage de l'harmonie tonale.

L'ensemble des accords possibles est divisé en classes d'accords considérés comme très proches. On considère comme proches deux accords qui ne diffèrent que par une transposition. De la même manière, on considère comme proches deux accords constitués des mêmes notes, c'est-à-dire qui donnent les mêmes restes modulo 12 (ceci correspond à la traditionnelle notion de "renversement" de l'harmonie tonale). Dans un cas comme dans l'autre, on voit que l'un des points communs entre les deux accords est le fait qu'ils définissent les mêmes intervalles modulo 12. De la même manière, on peut considérer comme proches deux accords qui sont inverses l'un de l'autre (comme do mi sol et do lab fa par exemple), puisqu'ils définissent également les mêmes intervalles modulo 12.

Le type d'un accord évoqué plus haut regroupera la classe de tous les accords qui sont proches de cet accord, dans le sens que l'on vient d'indiquer. Pour caractériser cette classe, on lui associe un représentant canonique.

1.Formes normale, primaire directe et primaire d'un accord.

Forme normale : Celle-ci est définie de la manière suivante.
a) L'accord est réduit modulo 12.
b) Les notes modulo 12 sont ordonnées à éléments croissants.
c) On considère l'ensemble des permutations circulaires de la liste obtenue, et parmi elles, on sélectionne celles qui ont une largeur minimale (intervalle entre les notes extrêmes).

d) Dans la sélection effectuée en c), on choisit l'élément dont la liste d'intervalles entre notes consécutives est minimale pour l'ordre alphabétique.

Forme primaire directe : On l'obtient en transposant sur la note 0 (do) la forme normale.

Forme primaire : On calcule la forme primaire directe de l'accord, et la forme primaire directe d'une inversion de l'accord, et on choisit celle des deux dont la liste d'intervalles entre notes consécutives est minimale pour l'ordre alphabétique.

Exemple : Considérons l'accord do3 ré3 solb3 la3 si3 ré#4, représenté par la liste de codes midi (60 62 66 69 71 75). La réduction modulo 12, et la mise en ordre croissant donne la liste (0 2 3 6 9 11). On calcule les différentes permutations circulaires. Les deux permutations de largeur minimale 9 sont marquées d'une *.

```
(0  2  3  6  9  11)  largeur=11
  (2  3  6  9  11  0)  largeur=10
    (3  6  9  11  0  2)  largeur=11
      (6  9  11  0  2  3)  largeur=9 *
        (9  11  0  2  3  6)  largeur=9 *
          (11  0  2  3  6  9)  largeur=10
```

Pour chacune des deux permutations de largeur minimale, on calcule la liste des intervalles entre notes consécutives :

(6 9 11 0 2 3) -> (3 2 1 2 1)
(9 11 0 2 3 6) -> (2 1 2 1 3)

donc c'est la deuxième qui est minimale pour l'ordre alphabétique. On en déduit la forme normale, puis la forme primaire directe en transposant sur do:

forme normale = (9 11 0 2 3 6)
forme primaire directe = (0 2 3 5 6 9)

Pour une inversion de l'accord, par exemple (76 80 82 85 89 91), la forme primaire directe obtenue est (0 1 3 4 6 9). Si on compare les deux listes d'intervalles entre notes consécutives, pour les deux formes primaires directes obtenues, on a :

(2 1 2 1 3)
(1 2 1 2 3)

et c'est donc la deuxième qui donne la forme primaire de l'accord :

forme primaire = (0 1 3 4 6 9).

Deux accords

(i) qui ne diffèrent que par une transposition, ou
(ii) qui ont mêmes notes modulo 12,

ont la même forme primaire directe. Deux accords qui ne diffèrent que par l'une des deux relations ci-dessus, ou

(iii) qui sont inverses l'un de l'autre,

ont la même forme primaire. Pour tout accord, la forme primaire représente sa classe pour la relation d'équivalence engendrée par ces trois relations (i), (ii) et (iii).

L'inventaire de toutes les classes possibles, pour des accords de trois sons, quatre sons, etc. a été fait par le théoricien Allen Forte, et celui-ci a établi une nomenclature de ces classes. Par exemple, la classe 6-1 est la première classe de six sons, c'est-à-dire (0 1 2 3 4 5). Dans le programme, la fonction *pc* permet de retrouver une classe :

```
? (pc 6 1)
= (0 1 2 3 4 5)
```

De même, la fonction *pc?* prend en argument une liste d'accords, et donne comme résultat la liste des classes correspondantes.

2.Approximation d'une classe dans une table de classes de référence.

Dans le contrôle des résultats fournis par l'interpolation, on remplace la classe d'un accord par une classe choisie dans une table de classes de référence, et on reconstruit un nouvel accord à partir de cette classe. La classe est choisie de telle sorte que sa distance par rapport à la classe de l'accord soit minimale. La distance entre classes est calculée de la manière suivante : soient $a_1, \ldots a_n$ les n éléments d'une classe, et $b_1, \ldots b_p$ les p éléments d'une deuxième classe. Soit k le minimum de n et p. La distance entre ces deux classes est donnée par la formule :

$$\frac{1}{k} \sqrt{(a_1-b_1)^2 + \ldots + (a_k-b_k)^2}$$

Exemple : (Centre Acanthe, juillet 1992)
Reprenons l'interpolation calculée précédemment. La fonction *chain* prend en argument un nom de variable, une liste d'accords et une table de classes. Elle remplace les accords de la liste par des accords dont la classe est dans la table, et le résultat et mis dans la variable passée en argument.

```
? s1
= ((38 47 49 52 58 60) (39 48 50 53 59 61) (41 50 52 55 61 63) (45 53 56 59 64
67) (50 56 60 63 67 72) (55 60 64 67 71 77) (60 64 69 72 75 82) (64 66 72 75
77 86) (66 68 74 77 79 88) (67 69 75 78 80 89))

? (pc? s1)
= (6-2 6-2 6-2 6-10 5-20b 5-20 5-19 6-2b 6-2b 6-2b)

? (chain c1 s1 (list (pc 6 2)))
= ((38 47 49 52 58 60) (39 48 50 53 59 61) (41 50 52 55 61 63) (44 53 55 58 64
66) (50 54 60 63 64 73) (54 58 64 67 68 77) (59 63 69 72 73 82) (64 66 72 75
77 86) (66 68 74 77 79 88) (67 69 75 78 80 89))

? (pc? c1)
= (6-2 6-2 6-2 6-2 6-2 6-2 6-2 6-2b 6-2b 6-2b)

? (chain c2 s1 (list '(0 1 2 3 4 5)))
= ((38 47 49 51 58 60) (39 48 50 52 59 61) (41 50 52 54 61 63) (44 53 55 57 64
66) (50 53 60 63 64 73) (54 57 64 67 68 77) (59 62 69 72 73 82) (64 66 73 75
77 86) (66 68 75 77 79 88) (67 69 76 78 80 89))

? (pc? c2)
```

= (6-1 6-1 6-1 6-1 6-1 6-1 6-1 6-1 6-1 6-1)

? (chain c3 s1 (list '(0 2 4 6 8 10)))
= ((42 48 52 56 58 62) (43 49 53 57 59 63) (45 51 55 59 61 65) (48 54 58 62 64 68) (52 58 60 66 68 74) (56 62 64 70 72 78) (61 67 69 75 77 83) (62 66 68 72 76 82) (64 68 70 74 78 84) (65 69 71 75 79 85))

? (pc? c3)
= (6-35 6-35 6-35 6-35 6-35 6-35 6-35 6-35 6-35 6-35)

La fonction *chain* est définie dans **Library:Chain**. Elle utilise un ensemble important de fonctions effectuant le remplacement d'un accord par un nouvel accord de classe donnée, qui sera décrit plus loin.

3.Contrôle circulaire.

Une autre manière de contrôler le résultat d'une interpolation par une table de classes consiste à parcourir simultanément la liste d'accords et la table, en reconstruisant chaque accord avec la classe correspondante dans la table, cette dernière étant considérée comme circulaire.

? (cir-chain c1 s1 (list (pc 6 2) (pc 6 10)))
= ((38 47 49 52 58 60) (40 48 51 54 59 62) (41 50 52 55 61 63) (45 53 56 59 64 67) (50 54 60 63 64 73) (55 59 64 68 69 77) (59 63 69 72 73 82) (63 66 71 74 77 85) (66 68 74 77 79 88) (66 69 74 77 80 88))

? (pc? c1)
= (6-2 6-10 6-2 6-10 6-2 6-10 6-2 6-10B 6-2B 6-10B)

C - RECONSTRUCTION D'UN ACCORD.

La reconstruction d'un accord à partir d'une classe donnée nécessite le calcul d'un certain nombre de paramètres de l'accord initial.

1.Calcul des parametres d'un accord.

Variable oct: *liste des octaves.*
On calcule l'octave de chaque note de l'accord, c'est-à-dire le quotient par 12 de son code midi (pour le do3 de code midi 60, l'octave vaut donc 5).

Variable pth: *réduction modulo 12.*
On calcule le reste modulo 12 de chaque note de l'accord.

Variable pth-asc: *mise en ordre croissant des restes modulo 12.*
La liste précédente est ordonnée à éléments croissants.

Variable ord: *liste des positions.*
Pour chaque élément de la liste des restes modulo 12, on calcule sa position (comptée à partir de 0) dans la liste ordonnée. Exemple : pour pth = (0 3 5 6 8 0), on a pth-asc = (0 0 3 5 6 8), donc la liste des positions est ord = (0 2 3 4 5 1).

Variable pth-cnt: *décompte des éléments égaux.*
Pour chaque élément de la liste ordonnée des restes modulo 12, on calcule son nombre d'occurrences consécutives. Dans l'exemple précédent, pth-asc = (0 0 3 5 6 8), donc on obtient pth-cnt = (2 1 1 1 1).

Variable pth-uni: *suppression des répétitions.*
On élimine les répétitions dans la liste ordonnée des restes modulo 12.

Variable pr: *forme primaire.*
Deux cas peuvent se produire selon la valeur d'une variable globale prime-sw. Si celle-ci vaut (), on calcule la forme primaire, et si elle vaut t, on calcule la forme primaire directe.

Variable normal: *forme normale.*
On calcule la forme normale de l'accord.

Variable fl1: *position de la première note de la forme primaire.*
Si la forme primaire est égale à la forme primaire directe, on calcule la position dans *pth-uni* de la note correspondant à la première note de cette forme primaire. Sinon, on calcule la position de la note correspondant à la première note de la forme primaire, dans la liste *pth-uni* inversée.

Variable fl2: *indicateur de la forme directe ou inverse.*
Cet indicateur vaut 0 si la forme primaire est égale à la forme primaire directe, et 1 sinon.

Le calcul de ces différents paramètres est effectué par la fonction *#:chain:set-old*, et on peut afficher leur valeur à l'aide de la fonction *par* sans argument. On a ainsi la session suivante :

```
? (#:chain:set-old '(36 39 41 42 44 48))
= (0 2 3 5 8)

?  (par)
  midi:(36 39 41 42 44 48)
  oct: (3 3 3 3 3 4)
  pth: (0 3 5 6 8 0)
pth-asc:(0 0 3 5 6 8)
  ord:(0 2 3 4 5 1)
-------------------------
pth-cnt:(2 1 1 1 1)
pth-uni:(0 3 5 6 8)
-------------------------
    pr:(0 2 3 5 8)
normal:(0 3 5 6 8)
   fl1:0
   fl2:1
-------------------------
  midi1:()
```

La variable *midi1* est utilisée dans la reconstruction d'un accord (voir la fonction *#:chain:get-new*). Pour reconstituer un accord à partir de sa classe, ou pour le remplacer par un accord voisin mais de classe différente, comme on le verra plus loin, le programme utilise deux variables supplémentaires qui ne sont pas affichées par la fonction *par* : *first* et *ref*.

Variable first : deux cas peuvent se produire.

1) la forme primaire est égale à la forme primaire directe.
Dans ce cas, *first* est la première note de la forme normale. Il suffit de transposer la forme primaire en ajoutant *first* à ses éléments pour retrouver toutes les notes modulo 12 de l'accord.

2) la forme primaire est différente de la forme primaire directe.
Dans ce cas, *first* est la dernière note de la forme normale. C'est en retranchant à *first* les éléments de la forme primaire que l'on peut retrouver toutes les notes modulo 12 de l'accord.

Variable ref :
Dans le cas 1), si on ajoute *first* aux éléments de la forme primaire, on restitue la forme normale. Dans le cas 2), si on retranche de *first* les éléments de la forme primaire, on obtient la liste inverse de la forme normale. Dans les deux cas, on retrouve les notes modulo 12 de l'acord, mais dans un ordre non nécessairement croissant. La variable *ref* est une liste de positions obtenue en prenant pour chaque élément de la liste ordonnée *pth-uni* sa position dans la liste obtenue par l'opération de 1) ou 2). Ceci permet de reonstituer *pth-uni* à partir de la forme normale, ou de l'inverse de la forme normale, selon le cas.

Exemple : Considérons l'accord (60 64 69 75), qui donne *pth-uni* = (0 3 4 9). La forme normale est (9 0 3 4), la forme primaire directe est (0 3 6 7) et la forme primaire est (0 1 4 7). La forme primaire est différente de la forme primaire directe, donc on est dans le cas 2). La liste inverse de la forme normale est (4 3 0 9), donc :

first = 4.

Par rapport à la forme normale inversée, la succession de positions de *pth-uni* est :

ref = (2 1 0 3).

A partir de la classe (0 1 4 7), on reconstitue donc les notes modulo 12 de l'accord en effectuant l'opération décrite en 2), c'est-à-dire 4-0=4, 4-1=3, 4-4=0, 4-7=9 (mod 12), puis on retrouve *pth-uni* = (0 3 4 9) en plaçant les éléments de (4 3 0 9) dans l'ordre défini par *ref*.
Remarque: Au signe près, la succession d'intervalles de la forme primaire +1 +3 +3 +5 se retrouve dans la forme normale inversée (4 3 0 9) : 4 -1 = 3, 3 -3 = 0, 0 -3 = 9 (mod 12), 9 -5 = 4.

2.Suppression des doublures.

Si un accord comporte des doublures (deux notes identiques à l'octave près, c'est-à-dire donnant le même reste modulo 12), celles-ci sont supprimées par la fonction *verify*. Le calcul proprement dit est effectué par la fonction *flatten* définie dans le fichier **Library:Flatten**.

Pour décrire l'algorithme utilisé, on associe à chacune des douze notes modulo 12 son nombre d'occurrences dans l'accord. Soit *cur-max-pos* la liste des notes dont le nombre d'occurrences dans l'accord est maximal, et *cur-min-pos* celle des notes dont le nombre d'occurrences est minimal (donc nul, c'est-à-dire qui sont absentes de l'accord). On parcourt la liste *cur-max-pos* en cherchant la note x la plus proche possible d'un élément de *cur-min-pos*, c'est-à-dire d'une note absente y. La distance entre notes est calculée vers la droite, ou vers la gauche, en considérant que la succession des notes est circulaire : à droite de la note 11 (si), on retrouve 0, 1, etc. et à gauche de la note 0 (do), on retrouve 11, 10 etc. Lorsqu'un tel couple de notes x et y est trouvé, on supprime une occurrence de x, et on introduit à la place une occurrence de y dans l'accord. Le calcul est ensuite recommencé avec les nouvelles listes *cur-max-pos* et *cur-min-pos*, jusqu'à ce que la différence entre les nombres maximal, et minimal, d'occurrences d'une note dans l'accord soit inférieure à 1. Si le nombre de notes absentes de l'accord était initialement

supérieur au nombre de doublures, on obtient à la fin du calcul un accord sans doublure.

Exemple: On considère un accord comportant un do, trois réb, un ré, un mi, et trois fa, représenté par le diagramme ci-dessous. Au départ, on a donc cur-max-pos = (1 5) et cur-min-pos = (3 6 7 8 9 10 11).

```
   *         *
   *         *
*  *  *     *  *
-----------------------------------------------
0  1  2  3  4  5  6  7  8  9  10 11
```

Le parcours de *cur-max-pos* donne x = 5 associé à la note y = 6 de *cur-min-pos*, la distance valant 1. Le diagramme est rééquilibré de la manière suivante:

```
   *
   *         *
*  *  *     *  *  *
-----------------------------------------------
0  1  2  3  4  5  6  7  8  9  10 11
```

Pour le parcours suivant, x = 1 et y = 3, la distance valant 2.

```
   *         *
*  *  *  *  *  *  *
-----------------------------------------------
0  1  2  3  4  5  6  7  8  9  10 11
```

Lors des étapes ultérieures, x = 1 avec y = 11, puis x = 5 avec y = 7.

```
            *
*  *  *  *  *  *  *              *
-----------------------------------------------
0  1  2  3  4  5  6  7  8  9  10 11



*  *  *  *  *  *  *  *  *           *
-----------------------------------------------
0  1  2  3  4  5  6  7  8  9  10 11
```

Une fois le calcul terminé, on obtient une nouvelle liste ordonnée croissante *pth-asc* de notes modulo 12, sans répétition. On reconstruit un nouvel accord, sans doublure, selon la technique décrite plus loin (fonction *#:chain:get-new*), et on recalcule les paramètres de l'accord.

3.Reconstruction d'un accord avec une classe imposée.

Ce calcul est effectué par la fonction *#:chain:get-new*. L'accord traité est supposé avoir été au préalable analysé par la fonction *#:chain:set-old*, afin que ses paramètres soient déterminés. Le remplacement par un nouvel accord est effectué à partir de ces paramètres et de la classe imposée, selon les étapes suivantes :

a) *remplacement de* pth-uni.
A partir de la classe imposée, et des variables *first* et *ref*, on calcule une nouvelle liste ordonnée sans répétition de notes modulo 12, destinée à remplacer *pth-uni*. Le

calcul utilise les opérations 1) ou 2) décrites plus haut, selon que la variable *fl2* vaut 0 ou 1.

b) *reconstitution des doublures et remplacement de* pth-asc (dans le cas où la fonction *verify* n'a pas été utilisée).
La variable *pth-cnt* indique les répétitions de notes qu'il faut insérer dans la liste obtenue en a) pour retrouver des doublures analogues à celles de l'accord initial.

c) *remplacement de* pth.
La variable *ord* donne la liste des positions des éléments de *pth*, ces positions faisant référence à la liste *pth-asc*. En plaçant les éléments de la liste obtenue en b) selon l'ordre défini par *ord*, on fabrique une liste destinée à remplacer *pth*.

d) *approximation des octaves.*
Des octaves sont affectées aux éléments de la nouvelle liste *pth* calculée en c) de telles sorte que les notes obtenues soient aussi proches que possible des notes de l'accord initial.

A titre d'illustration, reprenons l'accord (36 39 41 42 44 48) qui a été analysé précédemment par la fonction *#:chain:set-old*, et dont la classe est (0 2 3 5 8). On a alors la session suivante :

```
? (#:chain:get-new '(0 2 3 5 8))
= (36 39 41 42 44 48)

? (#:chain:get-new (pc 5 23))
= (37 39 41 42 44 49)
```

Dans le premier cas, c'est l'accord initial qui est reconstitué. Dans le second cas, on obtient un accord voisin du précédent, qui conserve certaines de ses caractéristiques, par exemple la doublure entre les deux notes extrêmes 37 et 49.

ORGANISATION DU PROGRAMME :

Les fonctions principales du programme sont dans le fichier **Library:History-functions** (complété par **Library:History-functions2** pour des calculs non évoqués ici). Comme on l'a vu précédemment, c'est là que se trouve la fonction *samples* effectuant l'interpolation d'accords. Le calcul proprement dit est réalisé par la fonction *interpolation* définie dans **Library:Rhythm-functions**. Le fichier **MyLib:Primitives** contient quelques fonctions utilitaires comme *round*.

Le contrôle par des classes est fait par la fonction *chain* elle aussi définie dans **Library:History-functions**. Les principales fonctions utilisées pour ce calcul se trouvent dans **Library:Chain**. Il s'agit de *#:chain:min-inter-chain*, *#:chain:distance*, *#:chain:set-old*, *#:chain:verify*, *#:chain:get-new*.

La fonction *#:chain:verify* appelle la fonction *flatten* définie dans le fichier **Library:Flatten** spécialement consacré à ce calcul.

Les principales fonctions utilisées par *#:chain:set-old* sont:

#:chain:midisplit qui calcule la plupart des paramètres d'un accord,

#:chain:track qui calcule *fl1* et *fl2*, et

#:chain:order qui calcule *ord*.

Quelques fonctions apparaissant dans le programme, comme *asc+*, *asclist*, *round2*, *rotlist*, *countlist*, *unique*, ainsi que *sqr* utilisée par *#:chain:distance*, sont définies dans **MyLib:Primitives**.

La fonction *#:chain:get-new* utilise principalement:

#:chain:back calculant la nouvelle valeur de *pth-asc*, et

#:chain:closest qui affecte aux notes du nouvel accord des octaves aussi proches que possible des notes de l'accord initial.

La fonction *makelist+* est définie dans **MyLib:Primitives**.

Le programme utilise également les fonctions *#:theory:normal*, *#:theory:prime1*, *#:theory:prime* calculant respectivement les formes normale, primaire directe et primaire d'un accord. Ces fonctions sont dans **Library:Set-theory**, ainsi que quelques fonctions utilitaires préfixées par *#:theory:*.

D - FONDAMENTALE VIRTUELLE.

Parmi les opérations portant sur des accords, disponibles dans le programme, figure également le calcul de la fondamentale virtuelle (G. Assayag). Etant donné un accord, on considère que ses notes sont les partiels du spectre harmonique d'une note fondamentale grave, non exprimée, dont il s'agit de calculer une approximation. On cherche donc une fréquence fondamentale f, la moins grave possible, telle que chaque note de l'accord ait comme fréquence (en Herz) un multiple entier de f.

Pour effectuer le calcul, on se donne la liste des fréquences des notes de l'accord, et une valeur d'approximation, exprimée par une puissance de 2: 1 pour l'approximation au demi-ton, 2 pour celle au quart de ton, 4 pour celle au huitième de ton, etc. A partir de la valeur fournie pour l'approximation, on encadre chaque fréquence de l'accord par un intervalle dont le diamètre correspond à l'approximation désirée. Si on note a_i et b_i les deux bornes de l'encadrement, celles-ci sont données par:

$$a_i = \frac{f_i}{2^{1/12*x}}, \text{ et } b_i = f_i * 2^{1/12*x},$$

où f_i est la fréquence d'une note de l'accord, et x la valeur d'approximation. On fixe un encadrement initial de la fondamentale cherchée, en posant arbitrairement $binf_1 = 5.0$, et $bsup_1 = b_1$, c'est-à-dire que l'encadrement initial inclut l'intervalle entourant la première fréquence f_1 de l'accord.

Le principe du calcul consiste à raffiner progressivement l'encadrement de la fondamentale virtuelle f. On veut en effet qu'il existe un entier r_i pour chaque note de l'accord tel que $a_i \leq r_i * f \leq b_i$. Si $binf_i$ et $bsup_i$ sont des bornes vérifiant cette condition pour les i-1 premières notes de l'accord, il suffit de prendre $binf_{i+1} = \max(binf_i, a_i/r_i)$, et $bsup_{i+1} = \min(bsup_i, b_i/r_i)$ pour que cette condition soit aussi

vérifiée pour la note suivante. Pour déterminer les valeurs limites entre lesquelles doit être choisi l'entier r_i, il suffit d'envisager d'abord le cas $f = bsup_i$ et $r_i * f = a_i$ ce qui donne sa limite inférieure $p_i = a_i / bsup_i$, et le cas $f = binf_i$ et $r_i * f = b_i$ ce qui donne sa limite supérieure $q_i = b_i / binf_i$.

L'algorithme peut être schématisé de la manière suivante. A la fin du calcul, les rangs des partiels sont les p_i.

tant que $i \leq n$ et ($p_i \leq q_i$ ou $i>1$) **faire**

si $p_i > q_i$ **alors** i := i-1;
$p_i := p_i + 1$;

sinon $r_i := p_i$;
$binf_{i+1} := max(binf_i, a_i/r_i)$;
$bsup_{i+1} := min(bsup_i, b_i/r_i)$;
i := i+1;
$p_i := ceil(a_i/bsup_i)$;
$q_i := trunc(b_i/binf_i)$;

La fonction *fond-virt* est définie dans le fichier **Library:Fond-virt**. A titre d'exemple, on a la session suivante:

```
? (fond-virt '(110. 220. 440.) 4)
= ((1 2 4) 108.423 . 111.6)

? (setq freq '(261.62 293.66 392. 493.88 554.36 698.46 830.61 932.33))
= (261.62 293.66 392. 493.88 554.36 698.46 830.61 932.33)

? (fond-virt freq 1)
= ((4 5 7 8 9 12 15 16) 58.27047 . 58.73295)

? (fond-virt freq 2)
= ((7 8 11 14 15 19 23 25) 36.31108 . 36.31108)

? (fond-virt freq 4)
= ((8 9 12 15 17 21 25 28) 32.82003 . 33.08404)

? (fond-virt freq 8)
= ((16 18 24 30 34 43 51 57) 16.34434 . 16.36088)
```

E - CREATION D'ACCORDS SYMETRIQUES.

Calcul des inversions d'une liste de notes modulo 12 qui ne donnent pas de notes communes.

Soit *l* une liste de notes modulo 12, l0 la liste rétrogradée et x la première note de l0. On ajoute x aux éléments de l0 et on prend les

opposés des éléments obtenus (-y à la place de y). Les résultats sont ramenés entre 0 et 11 modulo 12.

On note *mirror* la nouvelle liste obtenue, et on énumère les différentes transpositions de *mirror* (de +11 à 0), et on retient celles qui n'ont pas de notes communes avec la liste d'origine *l*.

Création d'accords symétriques.

Soit *acc* un accord, et *l1* l'une des listes de notes modulo 12 calculées précédemment, à partir de la liste *l* de notes modulo 12 de l'accord.

d) *approximation des octaves.*
Soit x la première note de l'accord initial, et y la première note modulo 12 de la nouvelle liste *pth* obtenue en c). On cherche une note équivalente à y modulo 12 qui soit dans [x-6,x+6[. A partir de la note ainsi obtenue, le calcul est ensuite effectué sur les intervalles. Soit i un intervalle entre deux notes consécutives de l'accord initial, et j l'intervalle entre les deux éléments correspondants de la nouvelle liste *pth*. On cherche un intervalle équivalent à j modulo 12 qui soit dans [i-6,i+6[. Ce calcul permet de déterminer l'octave de toutes les notes du nouvel accord.