Les débuts d'un clustering musical

Mémoire de Master 2 en Mathématiques Fondamentales
UPMC - Télécom ParisTech - IRCAM

Kilian Herraez
Sous la tutèle de
Carlos Agon, Moreno Andreatta, Jamal Atif et Isabelle Bloch

Mai 2017

Préambule

Ce mémoire traite d'un sujet toujours ouvert à l'heure actuelle et de nature explicitement exploratoire. Pour cette raison, il est impossible de présenter ce rendu sous une forme conventionnelle. En effet, malgré le fait que ce document vienne clôturer un master de mathématiques fondamentales, le sujet lui-même et son traitement comportent une caractéristique informatique très importante. On ne vise pas ici à étudier en détail un objet mathématique abstrait, mais à utiliser des outils mathématiques pour étudier les compositions musicales.
Ce document prendra donc une forme hybride, à la frontière entre un article de recherche, un rapport de stage et un mémoire. La philosophie est ici de fournir un article de musicologie computationnelle traditionel, mais aussi de détailler plus précisément l'implémentation des idées ainsi que les difficultés rencontrées lors de leurs mises en œuvre ; le sujet étant en cours d'exploration, baliser la voie pour les prochaines personnes qui s'y pencheront nous semble nécessaire.
Ce sujet nécessitant des connaissances poussées en mathématiques, il s'addresse avant tout à des mathématiciens. Cependant, l'expérience montre que les mathématiciens sont trop peu versés dans l'informatique pour pouvoir être à l'aise avec les outils que cette discipline fournit et s'en trouvent limités et ralentis dans leurs démarches. On présentera donc ici les codes importants utilisés au cours de l'étude ainsi que des conseils d'installation et d'utilisation des logiciels. L'utilisation d'une distribution de Linux est fortement conseillée, en ce que ce système d'exploitation facilite grandement l'installation des ressources.

Résumé

On s'intéresse ici à l'analyse intercompositionnelle ; cet article montre une voie possible vers des algorithmes de classification du style. Après avoir rappelé et formalisé des notions de musicologie basiques, nous introduirons le *Tonnetz*, un complexe simplicial particulièrement pertinent pour structurer les accords présents dans une composition, et ce, d'une manière très visuelle. Afin de parvenir à l'analyse de la composition, on cherchera ensuite à déformer le Tonnetz par le biais de la composition étudiée. Nous nous bornerons ici à déformer le Tonnetz isotrope bidimensionnel en un Tonnetz anisotrope plongé en trois dimensions. Nous discuterons alors des *filtrations* de cet espace en vue d'en calculer les *homologies persistantes* associées. Nous rappelerons alors les notions d'homologie persistante nécessaires avant de les apliquer à notre Tonnetz. Il en ressortira un *diagramme de persistence*, dont l'espace total est muni d'un distance, la *bottleneck distance*, qui nous permettera de séparer les compositions, en vue d'un *clustering* musical par *dendrogramme*. Des précisions concernant la mise en place informatique de ces idées seront données tout au long du texte et les codes sources importants seront donnés en annexes.

Table des matières

Chapitre 1

Notions et formalisations de musicologie

Pour permettre au plus grand nombre de bien comprendre tous les aspects du sujet, nous nous devons de traiter des divers concepts d'acoustique et de musicologie que nous utiliserons par la suite. La formation des gammes, les modes, la consonnance et la dissonnance sont des outils fondamentaux qu'on ne peut passer sous silence. Les notions susceptibles de jouer un rôle crucial seront ensuite formalisées. Le lecteur déjà à l'aise avec ces notions peut dès maintenant passer à la lecture du deuxième chapitre.

1.1 Un peu d'acoustique...

1.1.1 Terminologie

Le son est un phénomène acoustique et donc essentiellement physique. C'est une onde longitudinale réprésentée par une courbe périodique ou semi-périodique. On différencie plusieurs caractéristiques dans une onde :

L'amplitude qui mesure le volume sonore associé à cette onde. C'est la distance séparant la position d'équilibre et le point de déformation maximale de l'air.

La fréquence qui mesure le nombre de motifs qui se répète en une seconde. Elle caractérise le *pitch* d'un son, c'est-à-dire sa *hauteur*, entre grave et aigu. On considère que l'oreille humaine permet d'entendre les sons dont la fréquence est située entre 20 et 20000 Hertz. Plus bas, on trouve les infrasons et les ultrasons occupent l'autre bord du spectre.

Le motif qui témoigne de la complexité d'une onde. Celui-ci peut être sinusoïdal, carré, triangulaire, en dent de scie ou toute combinaison de ces formes. C'est cette caractéristique qui nous permet de différencier un violon d'un piano : ce que l'on apelle *timbre* est lié à la forme du signal. Les propriétés vibratoires

d'un instrument de musique caractérisent son timbre via la distribution des harmoniques liée à chaque note : la superposition de plusieurs signaux de fréquences différentes crée la forme de l'onde finale.

Comme évoqué plus tôt, le son est une onde périodique ou semi-périodique. Dans le premier cas, il s'agit d'un son qui ne perd pas en amplitude en fonction du temps. Par exemple, un orgue, une thérémine ou un synthétiseur produiront un son constant dans le temps. A l'opposé, les sons d'une guitare, d'un piano ou d'une caisse claire s'estompent et se modifient légèrement après leur production. L'intervalle de temps compris entre la formation de la note et le moment ou celle-ci devient inaudible est appelé *sustain* et est une caractéristique de la facture de l'instrument.

Avant de passer à la série des harmoniques et aux notions de consonnance et de dissonnance, nous devons faire la distinction, bien que très subjective, entre bruit et son. Un *son* est une onde périodique ou semi-périodique ayant un motif élémentaire relativement simple. Un *bruit* est soit une onde périodique ou semi-périodique ayant un motif très complexe, ne permettant plus à l'oreille de le reconnaître, soit une onde ne montrant aucune forme de périodicité. Une flûte produit un son alors qu'une porte qui claque émet un bruit, tandis que les percussions rythmiques sont à la frontière de ces deux notions.

1.1.2 La série harmonique

Les cordes et les corps des instruments à vent vibrent de manière plus complexe que ce que l'on pourrait croire. Dans le cas idéalisé d'une corde parfaite à laquelle on donnerait une forme sinusoïdole exacte avant de la relacher, on entendrait un son pur, composé d'un motif parfaitement sinusoïdal apparaissant à une fréquence donnée. Mais la réalité est toute autre, et il suffit de regarder un guitariste jouer pour comprendre qu'on est loin de la sinusoïde parfaite au moment de la création du son. Au lieu de vibrer d'un seul tenant à une fréquence donnée, la corde vibre à différentes fréquences par portions :

- Fondamentale : toute la longueur de la corde vibre à la fréquence la plus basse.
- Deuxième harmonique : la moitié de la corde vibre à *l'octave*, c'est à dire deux fois plus vite.
- Troisième harmonique : le tiers de la corde vibre une *quinte juste* au dessus de l'harmonique précédente, c'est à dire dans un rapport de 3 pour 2.
- Quatrième harmonique : le quart de la corde vibre *deux octaves* plus haut que la fondamentale.
- ...
- n-ième harmonique : $\frac{1}{n}$ de la corde vibre à n fois la fréquence de la fondamentale.

Les intensités des harmoniques dépendent grandement de l'instrument. Par exemple, la troisième harmonique d'une contrebasse et d'un ukulele jouant la même fondamentale n'auront pas les mêmes amplitudes, ce qui créera alors des motifs élémentaires très distincts, nous permettant de faire la différence entre un solo de Charles Mingus et de Cliff Edwards. La modélisation de la vibration d'une corde est donc liée à la *série harmonique* mathématique. Pour une corde de la longueur π, le point d'abscisse x est décalé algébriquement de la position d'équilibre en fonction de la somme suivante :

$$\sum_{k=1}^{\infty} A_k \sin(kx),$$

où A_k est l'amplitude de la k-ième harmonique. La suite des A_k est propre à chaque instrument individuel mais aussi à la manière de production du son. En effet, sur tout instrument à cordes il existe des *harmoniques*, qui sont des sons dont les motifs ne sont constitués que de certaines hamoniques. Bien que la terminologie soit kabbalistique car redondante, l'idée qui la sous-tend est simple : posez le doigt exactement au milieu de la corde d'une guitare, au dessus de la douzième frette, sans la plaquer contre le manche et pincez la corde. Un son totalement différent du son habituel de l'octave en ressortira. En forçant ainsi l'apparition d'un nœud au milieu de la corde, on permet uniquement l'apparition des harmoniques paires. En faisant de même au-dessus de la septième frette, au tiers de la corde, seules les hamoniques multiples de trois sonneront. En combinant les deux, il est alors possible de n'entendre que les multiples de six. On a là une méthode ingénieuse pour tester la facture d'un instrument à cordes et sa sonorité. Bien entendu, plus l'harmonique est haute et moins on l'entendra : passé une certain nombre, plus aucune harmonique ne s'entend. Cependant, le test des harmoniques sur une guitare permet de se rendre compte qu'il est possible d'entendre au moins jusqu'à la vingtième harmonique sur un instrument de facture moyenne. La guitare *slide* est un type de guitare optimisé pour le jeu au *bottleneck*, un cylindre de verre ou de métal avec lequel on touche les cordes sans les appuyer contre le manche, à ne pas confondre avec la *bottleneck distance* mentionnée plus haut. Ce procédé permet de ne faire ressortir que certaines harmoniques ; Derek Trucks en est un virtuose.

1.1.3 Consonance et dissonance

Ces notions relèvent de différents domaines et ont varié au fil du temps et des mouvements musicaux et artistiques. Nous traiterons rapidement ces aspects avant de nous arrêter plus longuement sur leurs quantification mathématique.

Aspects historiques et mouvements musicaux

Les règles de consonances tendent à devenir de plus en plus flexibles au cours du temps. Pendant longtemps, l'intervalle de triton fut banni des compositions car considéré diabolique. Certains genres particuliers de compositions sont toujours soumis à des règles strictes, comme la fugue et le contrepoint, règles explicitées dans le *Gradus ad Parnassum* de Johann Joseph Fux, parmis lesquelles ont peut citer l'interdiction des quintes et octaves parallèles ou celle du redoublement de la sensible. Mais ces règles classiques disparaissent dans la musique moderne. Le *blues* met à l'honneur l'accord de septième de dominante, dont l'élément caractéristique est le triton, le *jazz* découd la trame de la tonalité et du concept même de gamme, tandis que la musique *sérielle* et *atonale* détruit tout système préconçu de musicalité. Entre la rigueur classique et les poudroiements de la musique atonale, ou pourra citer les œuvres d'Olivier Messiaen ou de Claude Debussy.

Aspect physico-combinatoire

On pourrait résumer cette section en disant que la consonance n'est rien d'autre qu'une mesure de notre tolérance à l'égard de sons ne possédant que peu d'harmoniques en commun. Comme la tolérance est quelque chose qui évolue et se travaille, cela explique pourquoi les règles d'écriture du passé ne sont plus les mêmes que celles d'aujourd'hui, et aussi pourquoi l'éducation joue tant dans l'appréciation de la musique. Ceci explique aussi pourquoi différents peuples ont différentes musiques : on distingue une musique traditionnelle andalouse d'une musique traditionnelle japonaise en une fraction de seconde. Au-delà des instruments utilisés, les japonais ont tendance à composer avec des gammes pentatoniques et leurs modes, tandis que les espagnols ont privilégié les gammes heptatoniques comme les différentes gammes mineures et leurs modes. C'est pourquoi jouer seulement sur les touches noires d'un piano évoque indubitablement l'Asie.

Les aspects mélodiques et harmoniques de la consonance se confondant, nous expliquerons les choses pour deux notes jouées simultanément. Intéressons-nous donc aux ondes correspondant à deux notes de fréquences fondamentales 110 et 220 Hertz et notons dans la table 1.1 le début de leurs séries harmoniques.

Harmonique	Fréquence en Hz	Fréquence en Hz
1	110	220
2	220	440
3	330	660
4	440	880
5	550	1100
6	660	1320
7	770	1540
8	880	1760
9	990	1980
10	1100	2200

TABLE 1.1 – Séries harmoniques pour des fondamentales écartées d'une octave

On se rend compte que toutes les harmoniques de la troisième colonne se retrouvent dans celles de la première, et ce une fois sur deux. Autrement dit, ces deux fondamentales ont la moitié des harmoniques en commun, et sonneront très consonantes (il s'agit d'une note et de son octave). Faisons de même en partant d'une quinte dans la table 1.2.

Le nombre d'harmoniques en commun est plus faible dans ce cas, elle n'en partagent qu'un quart, mais reste cependant relativement grand, ce qui explique la position centrale de la quinte dans la musique tonale. Enfin, comparons les séries harmoniques de deux notes formant un intervalle de seconde mineure, c'est-à-dire dans un rapport de 10 pour 9, dans la table 1.3.

Ces deux notes ne partagent qu'un dixième de leurs harmoniques, les rendant relativement dissonantes.

Harmonique	Fréquence en Hz	Fréquence en Hz
1	330	220
2	660	440
3	990	660
4	1320	880
5	1650	1100
6	1980	1320
7	2310	1540
8	2640	1760
9	2970	1980
10	3300	2200

TABLE 1.2 – Séries harmoniques pour des fondamentales écartées d'une quinte

Harmonique	Fréquence en Hz	Fréquence en Hz
1	1000	900
2	2000	1800
3	3000	2700
4	4000	3600
5	5000	4500
6	6000	5400
7	7000	6300
8	8000	7200
9	9000	8100
10	10000	9000

TABLE 1.3 – Séries harmoniques pour des fondamentales écartées d'une seconde mineure

1.2 Tempérament et formation de la gamme chromatique

Ces deux notions de sonance ont formé la fondation de notre système de musique occidental actuel : la consonnance de l'octave et de la quinte a longtemps dicté le *tempérament* utilisé.

1.2.1 Quintes superposées modulo l'octave : avant la gamme tempérée

Encore à ce jour, l'intervalle d'octave est considéré comme si parfait que les deux notes qui le composent cet accord portent le même nom. On les considère alors comme étant les mêmes. Cependant, jouer des musiques uniquement constituée d'unissons et d'octaves d'une certaine note ne présente que peu d'intérêt musical. La tendance a donc été de développer la présence des quintes dans les compositions. En partant d'une note de fréquence 440 Hz donnée, listons-en dans la première colonne de la table 1.4 les quinte successives.

Pour tout n, il existe un unique k vérifiant l'équation

Suite de quintes (arrondies)	Modulo l'octave	Résultats ordonnés croissants
440	440	440
660	660	470
990	495	495
1485	743	529
2228	557	557
3341	835	595
5012	626	626
7518	470	660
11277	705	705
16915	529	743
25373	793	793
38059	595	835

TABLE 1.4 – La gamme par quintes

$$440\frac{3^n}{2^k} \in [440, 880[.$$

En divisant alors la première colonne par le k correspondant à chaque valeur, on obtient la seconde colonne, qui représente la suite des quintes ramenée sur une octave. On les range dans l'ordre croissant dans la dernière colonne. On forme ainsi la gamme longtemps utilisée durant le Moyen-Age.

Le fait d'avoir 12 degrés différents n'est ici pas anodin. Trop peu de degrés et la musicalité s'en trouve atrophiée, mais un trop grand nombre de notes rend le jeu de l'instrument impossible en pratique, demandant au musicien une précision micrométrique.

On repère toutefois un problème de taille avec cette gamme : l'octave ne tombe pas juste. En fait, quelque soit le nombre de degrés choisi, il est impossible de retomber exactement sur un multiple de la fréquence fondamentale. Cela vient du fait que l'équation :

$$\frac{3^n}{2^m} = 1$$

n'a pas de solution pour $(n, m) \neq (0, 0)$ ce qui est trivial par unicité de la décomposition en nombres premiers. Cependant, on peut trouver des résultats approchés, comme le couple $(5, 8)$ qui donne environ $0,95$ et qui permet de simuler l'octave dans la gamme qui nous intéresse. On peut bien entendu trouver de bien meilleures approximations, mais cela implique d'augmenter le nombre de degrés considérablement. Ce problème, d'origine purement arithmétique a des conséquence désastreuses en musique : il est impossible de créer un tempérament à partir de quintes superposée à moins de n'accepter la *quinte du loup*, due à cette différence entre cinq quintes et huit octaves. On peut d'ailleurs généraliser ce problème : il est impossible de former une gamme en superposant des ratios non-entiers de fréquences. Par exemple, il est impossible de superposer des secondes mineures modulo la quinte car l'équation

$$(\frac{10}{9})^n(\frac{2}{3})^m = \frac{2^{n+m}.5^n}{3^{2n+m}}$$

n'a pas de solution non triviale.

1.2.2 La gamme tempérée

Pour pallier à ce problème du à l'utilisation exclusive du corps des rationnels $\mathbb{Q}$ et non de sa clotûre algébrique $\mathbb{R}$, on ne doit plus chercher à superposer des fractions de la fondamentales. Pour assurer que l'octave est juste, on décide maintenant de découper celle-ci en 12 parties égales pour les fréquences. Comme l'oreille humaine fonctionne plutôt sur un plan multiplicatif qu'additif, on ne peut créer des degrés tous espacés de la valeur $\frac{2}{12}$: on a besoin d'utiliser le nombre $\sqrt[12]{2}$. Par définition,

$$(\sqrt[12]{2})^{12} = 2,$$

donc 12 degrés superposés dont les ratios successifs valent $\sqrt[12]{2}$ formeront finalement une octave parfaite. On appelle la gamme ainsi formée la *gamme chromatique tempérée* et elle fut inventée par le musicien Johann Sebastian Bach. C'est le tempérament le plus utilisé dans la musique occidentale : nous y sommes tellement habitués que certains tempérament différents, comme les tempéraments microtonaux, sonnent très dissonants à nos oreilles. On règle ainsi le problème de l'octave juste, en répartissant l'erreur de la quinte du loup précédente sur tous les autres degrés. On a donc une gamme où seule l'octave est juste, et chacun des autres degrés est légèrment décalé. Ceci est explicité dans la table 1.5.

Gamme par quintes	Gamme tempérée	Différence algébrique
440	440	0
470	466	4
495	494	1
529	523	6
557	554	3
595	587	8
626	622	4
660	659	1
705	698	7
743	740	3
793	784	9
835	830	5

TABLE 1.5 – Comparaison des deux tempéraments

On note toutefois que la quinte juste et la quinte approximée ne sont décalées que d'environ un Hertz, ce qui a permis un passage historique en douceur entre les deux tempéraments. Pour toute la suite de cette étude, nous présupposerons nous trouver dans la gamme tempérée : l'étude et l'analyse musiques classique ou comtemporaines ne peuvent se faire autrement que dans ce contexte ; Il n'aurait pas été de même pour l'étude de la musique médiévale.

Avant de passer à la section suivante, nous donnons une liste du nom des notes et de leurs intervalles associées en fonction de leur degré dans la table 1.6.

Note	Nom	Solmisation
0	Tonique	do
1	Seconde mineure	di - ra
2	Seconde majeure	re
3	Tierce mineure	ri - me
4	Tierce majeure	mi
5	Quarte juste	fa
6	Triton	fi - se
7	Quinte juste	sol
8	Sixte mineure	si - le
9	Sixte majeure	la
10	Septième mineure	li - te
11	Septième majeure	ti

TABLE 1.6 – Liste des notes de la gamme tempérée

1.3 Formation des gammes importantes et abrégé d'harmonie tonale

Nous travaillons maintenant dans une gamme tempérée à 12 degrés, que l'on appele gamme chromatique et que l'on assimile à $\mathbb{Z}/12\mathbb{Z} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$, où le degré 0 correspond à la *tonique* de la gamme. Cependant, une vaste majorité de musique utilisent en fait un sous-ensemble de cette gamme chromatique. Après avoir explicité cela, nous donnerons les principes fondamentaux d'harmonie qui permetteront de bien comprendre l'interêt du Tonnetz, objet que nous utiliserons plus tard.

1.3.1 L'ensemble des gammes : $2^{\frac{\mathbb{Z}}{12\mathbb{Z}}}$

La théorie combinatoire de l'ensemble des parties de $\mathbb{Z}/12\mathbb{Z}$ est complexe. Beaucoup d'études concernant des dénombrement de ces gammes à action près existent, et le lecteur intéressé pourra lire [5]. On y traite les actions des groupes de fonctions affines, de renversement. et de permutations. On souhaite surtout ici familiariser le lecteur avec certaines gammes d'importance toute particulière. Afin de faciliter les notations, nous imposons quelques conditions sur nos gammes.

Définition (Gamme et note)**.** Une gamme est un sous ensemble ordonné croissant de $\mathbb{Z}/12\mathbb{Z}$ commençant par 0 et une note est un élément d'une gamme.

Définition (Vecteur intervallique d'une gamme)**.** Etant donnée une gamme $G \in \mathbb{Z}/12\mathbb{Z}$ de cardinal $N = \text{Card}(G)$, on numérote ses notes dans l'ordre croissant via la suite $(G_0, G_1, \ldots, G_{N-1})$. Le vecteur intervallique correspondant à cette gamme est alors le N-uplet $(G_1 - G_0, G_2 - G_1, \ldots, G_{N-2} - G_{N-1}, G_{N-1} - G_0) =$

$(I_0, \ldots, I_{N-1})$ qui liste les écarts entre les notes successives de cette gamme. Sa somme vaut toujours 12.

Définition (Mode d'une gamme)**.** Etant donnée $G = (G_0, G_1, \ldots, G_{N-1})$, I son vecteur intervallique, un entier $k < N$ et en notant σ la permutation cyclique sur N éléments, le k-ième mode de la gamme G est l'unique gamme commençant dont le vecteur intervallique est $\sigma^{k-1}(I)$.

Exemple (Gamme naturelle majeure)**.** L'ensemble $(0, 2, 4, 5, 7, 9, 11)$ est appelé la gamme naturelle majeure. Son vecteur intervallique est $(2, 2, 1, 2, 2, 2, 1)$, et on lui distingue 7 modes distincts :

— Ionien $(0, 2, 4, 5, 7, 9, 11)$
— Dorien $(0, 2, 3, 5, 7, 9, 10)$
— Phrygien $(0, 1, 3, 5, 7, 8, 10)$
— Lydien $(0, 2, 4, 6, 7, 9, 11)$
— Mixolydien $(0, 2, 4, 5, 7, 9, 10)$
— Eolien $(0, 2, 3, 5, 7, 8, 10)$
— Locrien $(0, 1, 3, 5, 6, 8, 10)$

La gamme naturelle majeure ainsi qui son mode éolien, appelé gamme naturelle mineure, sont les piliers de la musique occidentale classique et contemporaine. Chantée *do, ré, mi, fa, sol, la, si*, elle revêt une importance capitale pour la suite de notre étude. Cependant, il existe des kyrielles d'autres gammes utilisées en pratique, en voici une liste non-exhaustive :

— Gamme harmonique mineure
— Gamme mélodique mineure
— Gamme par tons
— Gamme pentatonique de blues
— Gamme andalouse
— Gamme hongroise
— Gamme japonaise
— ...

1.3.2 Bases d'harmonie tonale de la gamme majeure

L'harmonie est la recherche de la beauté musicale des notes jouées *simultanément*, tandis que la mélodie concerne les notes jouées *successivement*. Ces deux notions sont fortement mêlées l'une à l'autre, et la mélodie elle-même très liée au *rythme*, qui est l'espace temporel dans lequel la composition progresse. Pour bien comprendre l'objet qu'est le Tonnetz, nous devons nous intéresser à l'harmonie de la gamme majeure.

Définition (k-accord)**.** Dans une gamme G de N notes et pour $k \leqslant N$, un k-accord ou accord de k sons est un sous-ensemble de cardinal k de G.

Définition (Accord majeur)**.** Un accord majeur est un 3-accord dont le vecteur intervallique est l'une des trois permutations cycliques du triplet $(4, 3, 5)$. Autrement dit, c'est un accord contenant une tierce mineure au-dessus d'une tierce majeure.

Définition (Accord mineur)**.** Un accord mineur est un 3-accord dont le vecteur intervallique est l'une des trois permutations cycliques du triplet $(3, 4, 5)$.

Autrement dit, c'est un accord contenant une tierce majeure au-dessus d'une tierce mineure.

Définition (Accord diminué)**.** Un accord diminué est un 3-accord dont le vecteur intervallique est l'une des trois permutations cycliques du triplet $(3, 3, 6)$. Autrement dit, c'est un accord contenant une tierce mineure au-dessus d'une tierce mineure. Comme tout accord diminué contient deux intervalles de triton, c'est une famille d'accords particulièrement dissonants.

Définition (Accord augmenté)**.** Un accord augmenté est un 3-accord dont le vecteur intervallique est l'une des trois permutations cycliques du triplet $(4, 4, 4)$. Autrement dit, c'est un accord contenant une tierce majeure au-dessus d'une tierce majeure. Tout accord diminué contenant deux intervalles de triton, c'est un accord particulièrement dissonant.

Exemple. $(0, 4, 7)$ est appelé *accord de do majeur* et $(7, 11, 4)$ porte le nom d'*accord de sol majeur*.

$(0, 3, 7)$ est l'*accord de do mineur* et $(9, 0, 4)$ est celui de *mi mineur*

$(0, 3, 6)$ est l'accord de *do diminué*, et $(11, 2, 5)$ celui de *si diminué*

$(0, 4, 8)$ est l'accord de *do augmenté*. Contrairerement au troix autres espèces d'accord évoquées, il n'apparait pas naturellement dans la gamme majeure de do.

Notons que les accords $(4, 7, 11)$ et $(7, 11, 4)$ sont formés des mêmes notes, mais sont distincts à l'oreille : en effet, la permutation appliquée au vecteur intervallique de référence qui forme l'accord définit ce que l'on appelle le renversement de cet accord, mais nous n'irons pas plus loin sur ce sujet. Les accords majeurs, mineurs et diminués forment la base de notre musique. Prenons la gamme majeure de do. A partir de chaque note, formons un accord de trois sons comprenant cette note, celle placée deux positions après, et celle placée deux positions après la précédente. On forme alors la *gamme d'accord* associée avec la gamme naturelle majeure. Pour nommer les accords qui la compososent : do majeur, ré mineur, mi mineur, fa majeur, sol majeur, la mineur et si diminué. L'alternance de différentes espèces d'accords témoigne fortment de la gamme de notes sous-jacente. Alors qu'une autre mode de cette gamme engendrerait cette même liste à permutation cyclique près, et qu'une translation de cette gamme engendrera exactement la même liste, toute autre gamme donnera lieu à une nouvelle combinaison des espèces d'accords, certains pouvant ne pas être caractérisables par les dénominations "majeur", "mineur", "diminué", "augmenté". Le tableau 1.7 en montre un exemple.

1.4 Vision géométrique, symétries et opérations

1.4.1 Forme de la gamme majeure et des accords

La vision géométrique des gammes et des accords nous permet de gagner en intuition. Sur l'image 1.1, représentant la gamme de do majeur tournant dans le sens indirect en partant du sommet le plus à droite, on discerne clairement une symétrie axiale, dont l'axe relie le 2ème au 8ème degré. Cette symétrie est appelée *inversion* de la gamme, et est un processus utilisé en musique serielle.

Degré	Naturelle majeure	Naturelle mineure	Mélodique mineure (0, 2, 3, 5, 7, 9, 11)
1	Majeur	Mineur	Mineur
2	Mineur	Diminué	Mineur
3	Mineur	Majeur	Augmenté
4	Majeur	Mineur	Majeur
5	Majeur	Mineur	Majeur
6	Mineur	Majeur	Diminué
7	Diminué	Majeur	Diminué

TABLE 1.7 – Liste des espèces d'accords pour différentes gammes

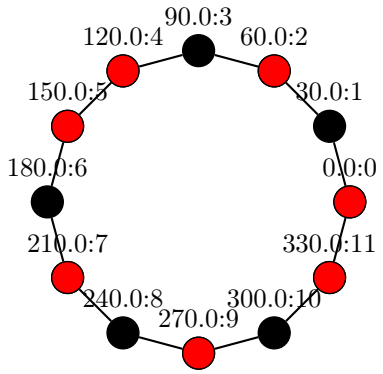

FIGURE 1.1 – La gamme de do majeur

Regardons maintenant les accords de do de chaque espèce dans la figure 1.2. Les deux premiers accords ne possèdent aucune symétrie. Cependant, il est possible de passer de l'une à l'autre par une symétrie axiale : nous en parlerons dans la prochaine section. On se rend aussi compte que l'accord diminué possède une symétrie non-triviale, alors que l'accord augmenté en a cinq. Une règle empirique de musique veut que moins l'accord a de symétries, plus il est consonant.

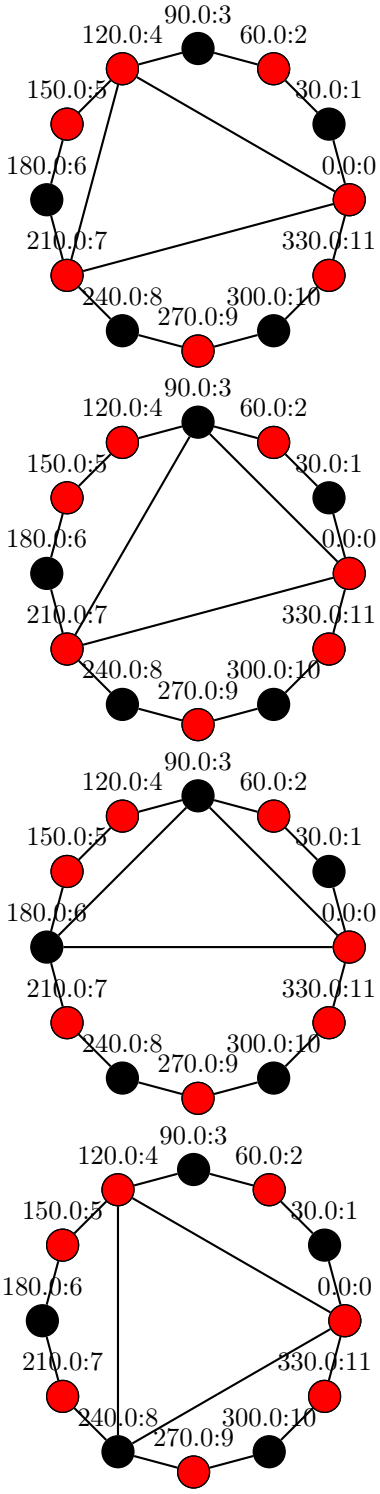

FIGURE 1.2 – Do majeur, mineur, diminué, augmenté

1.4.2 Opérations sur les accords

On a vu précedemment qu'une symétrie axiale permet de transformer un accord majeur en mineur et vice versa, mais ce n'est pas la seule possible. Nous allons maintenant décrire les trois principales.

La symétrie axiale dont l'axe passe par les milieux des segments $[3, 4]$ et $[9, 10]$ permet de transformer un accord majeur en mineur, sans changer le nom de la note, comme le montre la figure 1.3. On note qu'il est impossible de réaliser cette transformation en restant dans la gamme majeure : on aura pour cela besoin d'une gamme possédant une tierce majeur et une tierce mineure.

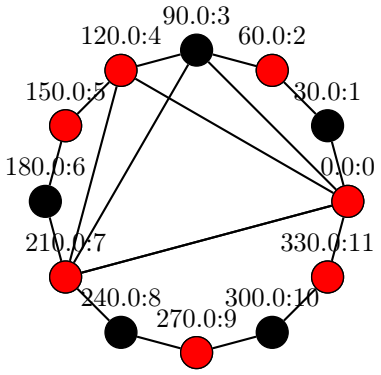

FIGURE 1.3 – De do majeur à do mineur et vice versa

La symétrie axiale dont l'axe passe par les sommets 2 et 8 permet de transformer un accord majeur en son relatif mineur, comme le montre la figure 1.4. Cette transformation se fait à l'intérieur de la gamme.

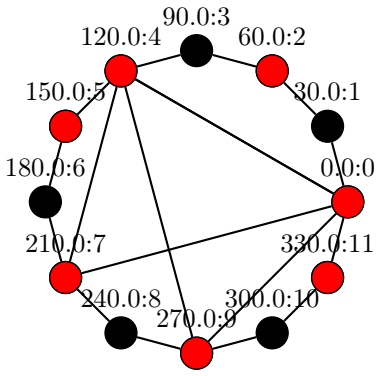

FIGURE 1.4 – De do majeur à la mineur et vice-versa

La symétrie axiale dont l'axe passe par le milieu des segments $[5, 6]$ et $[0, 11]$ permet de transformer un accord majeur en son relatif sensible mineur, comme le montre la figure 1.5. Cette transformation reste aussi dans la gamme.

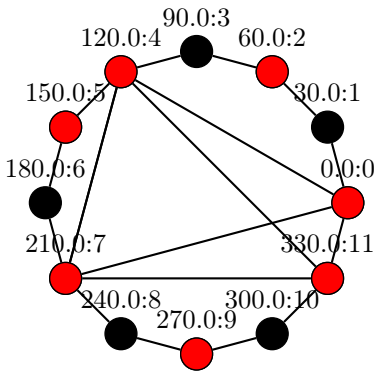

FIGURE 1.5 – De do majeur à mi mineur

Toutes ces opérations vont nous permettre de bien comprendre l'agencement du Tonnetz, dont nous traitons maintenant.

Chapitre 2

Le Tonnetz : déformation et analyse

Le Tonnetz est un complexe simplicial plongé en trois dimensions qui représente les k-accords dans un système intervallique donné.

2.1 Tonnetz bidimensionnel

Une manière simple de comprendre le Tonnetz bidimensionnel est de le voir comme un pavage du plan par des triangles equilatéraux. Les sommets de ces triangles correspondent aux notes de la gammes, les arrêtes aux intervalles et les triangles aux accords de trois sons. On peut facilement envisager de rajouter simplexes de dimension supérieures pour représenter les accords de k-sons. Des études concernant le 12-simplexe complet sont en cours dans l'équipe de Moreno Andreatta à Strasbourg ; nous nous bornerons ici au Tonnetz bidimensionnel visualisable réprésentant au mieux les 3-accords. L'utilisation d'autres pavages est aussi envisageable : les pavages semi-réguliers semblent pouvoir encoder proprement des systèmes de sons microtonaux ou même s'adapter parfaitement à certains genre musicaux ayant des codes stricts, comme le blues.

2.1.1 Définition intervallique

Les Tonnetz doivent vérifier la condition de boucle : si l'on part d'un sommet du graphe, représenté par le nombre k, et que l'on parcourt un boucle fermée de longueur N le long des arrêtes A_i ayant chacune la valeur intervallique I_i, nous devons retomber à l'arrivée au même nombre k. Autrement dit, dans $\mathbb{Z}/12\mathbb{Z}$,

$$\sum_{k=1}^{N} I_i = 0$$

En fait, il est aisé de constater que cela revient en fait à dire que la somme des valeurs intervaliques des côtés d'un triangle du graphe doit valoir 12. Il y a donc autant de Tonnetz que de partitions de 12. Nous allons utiliser par la suite la partition

$$12 = 3 + 4 + 5$$

Celle-ci est particulièrement adaptée à la musique classique occidentale, car elle code parfaitement les accords mineurs, majeurs, diminués et augmentés présentés plus tôt. En effet, sur un triangle, on a trois axes de déplacement, un pour chaque arrête. Dans notre cas, on a l'axe des tierces mineures (3), l'axe des tierces majeures (4) et le dernier est déjà prédit par la condition de boucle, c'est l'axe des quintes (5).

2.1.2 Visualisation du Tonnetz

Différents outils permettent la visualisation du Tonnetz. On citera particulièrement *Hexachord* développé par Louis Bigot dans le cadre de sa thèse [2], qui permet l'affichage dynamique du Tonnetz en fonction d'une entrée MIDI et le calcul du système intervallique le plus pertinent pour une composition. Nous visualiserons le Tonnetz, ou tout du moins une partie, grâce au logiciel de calcul formel *SAGE*, disponible en ligne à cette adresse (https ://cocalc.com/). Le compilateur SAGE est particulièrement flexible et la grande quantité de librairies préexistantes rendent le langage particulièrement attractif. De plus, ce logiciel donne aussi la possibilité d'exporter beaucoup de données dans des formats différents, ce qui permet d'utiliser des méthodes propres à SAGE avant d'exporter les résultats pour les traiter avec un autre programme. L'image 2.1 représente une portion du Tonnetz générée avec SAGE, dont le code est donné à l'annexe B. En utilisant un code rigoureux, il est possible d'exporter cet objet 3D en un fichier compréhensible par le logiciel de visualisation *Paraview*. Les extensions .x3d et .stl sont des bons candidats pour importer le Tonnetz dans Paraview. Le plugin TTK, *Topology ToolKit* qui vient se greffer sur Paraview est alors prêt à être utilisé pour des calcul de diagrammes de persistance.

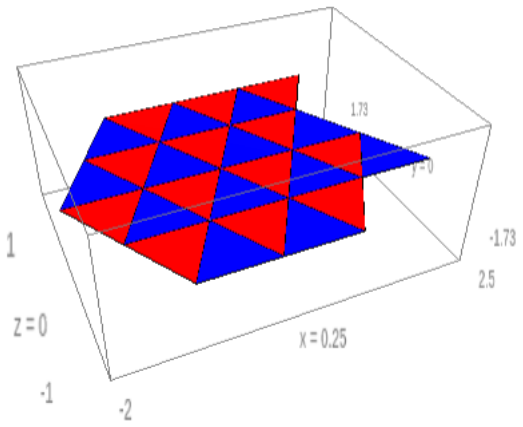

FIGURE 2.1 – Tonnetz bidimensionnel isotrope.

On note que le Tonnetz est en réalité topologiquement équivalent à un tore : comme quatre tierces mineures et trois tierces majeures valent une octave, cet

objet qui semble initialement plat est en fait un tore. Cela est présenté dans [6].

2.2 Tonnetz anisotrope

Une fois que l'on a créé ce Tonnetz plan, nous souhaitons maintenant le déformer en fonction de la composition étudiée. On obtiendra alors l'empreinte du morceau sur cet espace musical, comme suggéré dans [1].

2.2.1 Fonction de déformation

On doit maintenant définir une manière pertinente de déformer le Tonnetz. Il existe plusieures manières de le faire, en voilà des exemples :

— Rapprochement des voisins musicaux
— Création de nouveaux simplexes
— Elévation des simplexes

Dans notre étude, nous avons décidé d'utiliser la dernière méthode, celle de l'élévation des simplexes. Celle-ci consiste à associer un nombre à chaque simplexe. Nous ne le ferons que pour les notes, c'est-à-dire les sommets. On fait ainsi entrer le Tonnetz dans la troisième dimension. Pour associer sa hauteur à une note x du Tonnetz, on peut utiliser l'une des deux fonctions suivantes :

$$Z(x) = \frac{Card(Pitch^{-1}(x))}{Card(N)},$$

qui compte le nombre de notes du fichier qui sont à une hauteur donnée et que l'on normalise par le nombre total de notes, ou

$$Z(x) = \frac{\sum\limits_{n \in Pitch^{-1}(x)} Length(n)}{\sum\limits_{n \in N} Length(n)},$$

qui compte la durée totale des notes qui sont à une hauteur donnée et que l'on normalise par la durée totale des notes. Nous utiliserons la première solution, pour sa facilité d'implémentation. Dans ces deux définitions, on a utilisé deux fonctions qui émergent du format MIDI :

$$\begin{array}{rcccc} Pitch & : & N & \to & \mathbb{Z}/12\mathbb{Z} \\ & & n & \mapsto & Pitch(n) \end{array}$$

$$\begin{array}{rcccc} Length & : & N & \to & \mathbb{R} \\ & & n & \mapsto & Off(n) - On(n) \end{array}$$

Les fonctions On et Off sont des commandes natives du format MIDI et N est ici l'ensemble des notes du fichier. Pour plus de détail sur le sujet, la référence [4] explique brièvement le fonctionnement de ce système. On peut maintenant associer à chaque sommet du Tonnetz une hauteur caractéristique de la composition étudiée. Le décompte à la main des notes étant fastidieux et peu sûr, l'automatisation de ce processus est cruciale : la macro Visual Basic présentée en annexe A permet de compter les notes présentent dans un fichier MIDI. Ce fichier doit au préalable avoir été traduit en un fichier d'instructions intelligibles au format CSV, *comma-separated values* interpretable par le tableur

Microsoft Excel. Il est possible de l'adapter pour les équivalents libres Open Office et autres. La figure 2.2 montre une visualisation du résultat en utilisant pour composition de déformation le *Prélude en do majeur* de Bach, BWV 846.

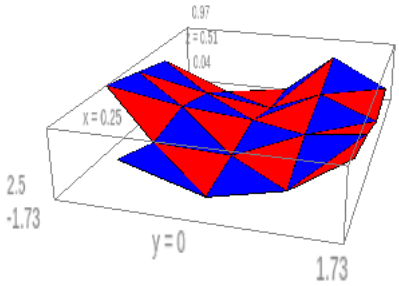

FIGURE 2.2 – Tonnetz anisostrope déformé par la composition BWV 846.

2.3 Homologie persistante et bottleneck distance

Après avoir déformé ainsi le Tonnetz, nous cherchons à étudier sa forme. L'outil que nous utilisons pour cela est l'homologie persistante. Nous présentons ici uniquement les notions dont nous auront besoin : le lecteur intéressé pourra se renseigner plus en détail grâce au document [3].

2.3.1 Cadre théorique

L'homologie persistante peut se résumer comme étant l'application de l'homologie traditionnelle à différents niveaux de zoom sur la variété étudiée. Ce zoom est formalisé par une filtration de la variété.

Définition (Filtration)**.** Une filtration d'un espace X est une suite

$$\varnothing = S_0 \subset S_1 \subset \cdots \subset S_n = X$$

Chacun de ces S_i correspond à un niveau différent de précision sur notre variété. En se remémorant la figure 2.2, on note que nous avons déformé le Tonnetz en hauteur : il serait donc judicieux de définir notre filtration d'étude comme une suite d'ensembles de sous-niveau de la fonction de projection sur la dernière coordonée. En d'autres termes, nous allons découper notre Tonnetz par des plans horizontaux de plus en plus hauts et étudier ce qui se situe sous chacun de ces plans par homologie simpliciale. Le Tonnetz étant un objet purement simplicial, le calcul effectif de l'homologie persistante pourra être automatisé. Toutes ces homologies vont être calculées dans $\mathbb{Z}/2\mathbb{Z}$. Voilà la marche à suivre : notons f_i la fréquence d'apparition de la i-ème note de notre gamme tempérée. Rangeons la liste des f_i par ordre croissant, et notons cette nouvelle liste F_i, que nous appelerons liste des *valeurs de persistance* par la suite. Nous définissons notre filtration par :

$$S_i = p_z^{-1}(]-\infty, F_i])$$

où p_z est la fonction dernière coordonée sur le Tonnetz à valeur dans $\mathbb{R}$.

Nous allons ensuite trouver les groupes d'homologie à coefficient dans $\mathbb{Z}/2\mathbb{Z}$ que nous notons $H_i(S_k)$. Notre Tonnetz étant essentiellement torique, ceux-ci n'auront d'intérêt que pour $i = 0$ (composantes connexes), $i = 1$ (trous) et $i = 2$ (vides). L'implémentation subséquente représentera le Tonnetz par un tore simplicial, et non un plan simplicial. La filtration par les S_k permet alors de faire apparaitre des notions propres à l'homologie persistante : la *naissance* et la *mort* d'un groupe d'homologie.

Définition (Naissance et mort d'une classe d'homologie)**.** Pour $k < m$, notons

$$p^i_{k,m} : H_i(S_k) \to H_i(S_m)$$

la flèche image de l'inclusion $S_k \subset S_m$ par le foncteur H_i. Soit $\alpha \in H_i(S_k)$ une classe d'homologie.

On dit que α est née à S_k si elle n'est pas l'image d'une classe d'homologie de $H_i(S_{k-1})$ par la fonction $p^i_{k-1,k}$.

Dans ce cas, on dit que la classe α meurt à S_m si l'image de $p_{k-1,m-1}$ ne contient pas l'image de α par $p_{k,m-1}$, mais que cette image de α est contenue dans d'image de $p_{k-1,m}$.

Intuitivement, pour $k = 0$, cela correspond au fait qu'une nouvelle composante connexe apparaisse à une certaine étape de la filtration et qu'elle se résorbe à une étape ultérieure. Cela permet de définir le *temps de vie* d'une composante connexe, d'un trou ou de tout autre artefact topologique de dimension supérieure. Le temps de vie peut en général être défini comme étant la différence entre l'indice de filtration où le groupe d'homologie disparait et celui où il apparait. Une manière plus pertinente de le faire dans notre cas pourrait être de considérer la différence des valeurs de persistance F_i plutôt que les indices de filtration. Il est important de comprendre que dans les faits, certains groupes d'homologie peuvent exister du début jusqu'à la fin de la filtration : la convention est que ces groupes particuliers meurent en $+\infty$. A chaque groupe d'homologie sont donc associés deux nombres, le premier correspond à sa naissance, le second à sa mort. On obtient alors un nuage de point de $\mathbb{R}^2$ situé au-dessus de la première bissectrice. L'ensemble de ces points auxquel on adjoint la bissectrice est appelé *diagramme de persistance de degré i*. Un exemple créé par TTK est donné dans la figure 2.3.

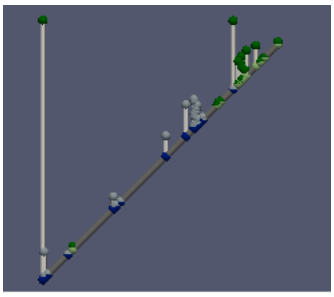

FIGURE 2.3 – Un exemple de diagramme de persistance généré par TTK.

Il existe une distance sur l'espace des diagrammes de persistance : la *bottleneck distance*.

Définition (Bottleneck distance en degré i)**.** Soient $Diag_1$ et $Diag_2$ deux diagrammes de persistance de même degré (ils émanent tous deux du foncteur H_i). On définit alors

$$d_B^i(Diag_1, Diag_2) = \inf_{\phi} \sup_{x \in Diag_1} \|x - \phi(x)\|,$$

où la norme considérée est la norme infinie standard de $\mathbb{R}^2$, et ϕ une bijection entre $Diag_1$ et $Diag_2$, ce qui est rendu possible par le fait que la première bissectrice appartient au diagramme.

Cette mesure est en fait une mesure de transport minimal entre les deux nuages de points. Comme nous nous intéréssons à l'homologie en degrés 0, 1 et 2, on considère alors la distance totale définie par

$$d_B(Diag_1, Diag_2) = d_B^0(Diag_1, Diag_2) + d_B^1(Diag_1, Diag_2) + d_B^2(Diag_1, Diag_2)$$

C'est cette distance qui va nous permettre de séparer nos compositions. Mais pour cela, nous devons automatiser tous les calculs nécessaires.

2.3.2 Cadre computationnel

L'automatisation computationnelle est rendue possible par le fait que notre objet d'étude est un complexe simplicial. Les simplexes sont aisés à représenter et à traiter par l'outil informatique : il s'agit d'une liste d'ensembles, les simplexes, qui doit vérifier la condition de faces : tout sous-ensemble d'un ensemble appartient aussi à la liste. Cela simplifie grandement les calculs, car le calcul des groupes d'homologie dans $\mathbb{Z}/2\mathbb{Z}$ revient dans ce cas à des réductions de matrices rectangulaires contenant uniquement des 0 et des 1 : les matrices de bords du complexe. En bref, si le simplexe i est un bord du simplexe j, le coefficient (i, j) de la matrice de bord vaut 1, ou 0 dans le cas contraire. Le calcul des nombres de Betti revient alors à des calculs de rangs et de kernels sur ces matrices. Pour plus de détails concernant la mise en place pratique de ces idées, nous renvoyons le lecteur vers la source [3].

Il existe quelques outils permettant de calculer l'homologie persistante d'un complexe. Nous en citerons trois :

— RedHom
— GUDHI
— MatLab et javaplex

Les deux premiers sont des librairies C++, tandis que le dernier est un environnement de programmation pouvant traiter les complexes et leurs homologies à l'aide du package *javaplex*. Comme nous n'avons pas testé RedHom, nous n'en parlerons pas plus longuement. GUDHI est un outil très puissant permettant de réaliser un large spectre de calculs différents. Son format, un librairie C++, lui permet très aisément de s'intégrer dans d'autres programmes et d'automatiser une longue chaîne de calculs. Cela le rend particulièrement fonctionnel et adapté à notre problématique. Cependant, par sa nature, GUDHI est difficile d'installation et l'utilisation d'une distribution de Linux est fortement recommandée pour ce faire. De plus, elle nécessite l'installation d'autres librairies indépendantes, rendant la démarche très labyrinthique pour les néophytes. On trouvera à l'annexe C le code C++ utilisant GUDHI qui permet de calculer le diagramme de persistance du Tonnetz associé à une composition. Cependant, un problème de patch de cette librairie fait disfonctionner les méthodes de traitement de bottleneck distance sur notre machine. Nous nous tournerons donc vers MatLab pour cette partie du calcul. MatLab est un environnement et un langage de programmation bien connu des scientifiques. Son installation est beaucoup plus aisée que celle de GUDHI, mais le logiciel n'est pas un *freeware*. Cependant, les étudiants de l'UPMC ont accès à des licences spéciales et gratuites. Ce software étant relativement répandu dans la communauté, beaucoup de packages différents et de documentation est disponible pour son utilisation. La syntaxe est flexible et facilement compréhensible ; de plus, ce logiciel tout-en-un permet réellement de s'immerger dans la programmation. L'annexe D présente un algorithme écrit en MatLab qui permet de calculer les homologies persistantes ainsi que les bottleneck distances associées à deux compositions.

2.4 Application de l'analyse à des compositions

Nous souhaitons maintenant uiliser cette méthode pour séparer de véritables compositions.

2.4.1 Compositions à séparer

Nous avons décidé de tester notre algorithme en prenant des musiques venant de deux genres très distincts harmoniquement parlant : la musique classique de J. S. Bach et celle de plusieurs groupes de punk rock. Les compositions de Bach sont complexes et certaines modulent au cours du temps, alors que le punk est un genre très répétitif fortement ancré dans une tonalité particulière, contenant rarement plus de trois accords distincts. Nous listons dans la table 2.1 les musiques que nous allons analyser.

Classique	Punk
Badinerie en si mineur	American Idiot
Fugue en la mineur	Anarchy In The UK
Prélude en do majeur	Beat On The Brat
Fugue en do majeur	Belsen Was A Gas
Prélude	Blitzkrieg Bop
Allemande	I Fought The Law
Courante	I Wanna Be Sedated
Sarabande	London Calling
Menuet	Should I Stay Or Should I Go
Gigue	Welcome To Paradise

TABLE 2.1 – Listes des compositions à séparer

Les six dernières musiques de Bach sont issues de son premier concerto pour violoncelle solo, et les trois précédentes du *Clavier bien tempéré*. Les musiques punk rock viennent des groupes Green Day, Sex Pistols, The Ramones et The Clash. Nous allons lister les distances entre chaque paire de musiques dans une grande matrice symétrique de 20 lignes par 20 colonnes. Voilà l'ordre des compositions :

— 1 : Badinerie en si mineur
— 2 : Prélude en do majeur
— 3 : Fugue en la mineur
— 4 : Fugue en do majeur
— 5 : Prélude
— 6 : Allemande
— 7 : Courante
— 8 : Sarabande
— 9 : Menuet
— 10 : Gigue
— 11 : American Idiot
— 12 : Anarchy In The UK
— 13 : Beat On The Brat
— 14 : Belsen Was A Gas
— 15 : Blitzkrieg Bop
— 16 : I Fought The Law
— 17 : I Wanna Sedated
— 18 : London Calling
— 19 : Should I Stay Or Should I Go
— 20 : Welcome To Paradise

Nous avons choisi de mettre les musiques du même style ensemble pour une meilleure lisibilité des résultats à venir.

2.4.2 Résultats de l'analyse

La table 2.2 est la matrice des distances obtenue grâce à une version améliorée de l'algorithme donné en annexe D appliquée aux musiques présentées à la section précédente.

0	9	21	18	11	15	14	18	14	8	30	22	68	37	65.5	54	31	31	36	10
9	0	18	15	11	12	9	15	13	9	33	15	70	36	67.5	57	29.5	24	39	8
21	18	0	9	15	12	13	10	11	15	49.5	33	82	52	79.5	69.5	42	42	54.5	21
18	15	9	0	12	7	8	2	10	12	47.5	30	80	50	77.5	67.5	39	39	52.5	18
11	11	15	12	0	11.5	10.5	12	10	8.5	36	26	72	41	69.5	59.5	35	35	42	14
15	12	12	7	11.5	0	3	7	12	9	45	27	78	48	75.5	65.5	36	36	50.5	16
14	9	13	8	10.5	3	0	8	10	8	42	24	76	45	73.5	63.5	33	33	48	15
18	15	10	2	12	7	8	0	10	12	47.5	30	80	50	77.5	67.5	39	39	52.5	18
14	13	11	10	10	12	10	10	0	12	42	28	76	45	73.5	63.5	37.5	37	48	16
8	9	15	12	8.5	9	8	12	12	0	36	24	72	39	69.5	59.5	33	33	42	12
30	33	49.5	47.5	36	45	42	47.5	42	36	0	26	39	25	36	27	19.5	18	8.5	30
22	15	33	30	26	27	24	30	28	24	26	0	57	21	54	42	23.5	19	24	12
68	70	82	80	72	78	76	80	76	72	39	57	0	36	7	16	48	48	33	68
37	36	52	50	41	48	45	50	45	39	25	21	36	0	33	21	27.5	25	20.5	33
65.5	67.5	79.5	77.5	69.5	75.5	73.5	77.5	73.5	69.5	36	54	7	33	0	18	45	45	30	65.5
54	57	69.5	67.5	59.5	65.5	63.5	67.5	63.5	59.5	27	42	16	21	18	0	35.5	33	19	54
31	29.5	42	39	35	36	33	39	37.5	33	19.5	23.5	48	27.5	45	35.5	0	16.5	21	27.5
31	24	42	39	35	36	33	39	37	33	18	19	48	25	45	33	16.5	0	21.5	23
36	39	54.5	52.5	42	50.5	48	52.5	48	42	8.5	24	33	20.5	30	19	21	21.5	0	36
10	8	21	18	14	16	15	18	16	12	30	12	68	33	65.5	54	27.5	23	36	0

TABLE 2.2 – Matrice des distances

Il est possible d'analyser ces données dès maintenant. Le premier bloc en haut à gauche est le bloc des comparaisons des musiques classiques entre elles. Celles-ci sont relativement proches, la distance moyenne se trouvant autour de 10. Une grande partie des musiques de Bach étudiées formant une suite pour violoncelle, il est confortant de voir cette proximité. Le dernier bloc en bas à droite forme les comparaisons des musiques punk avec elles-mêmes. Les valeurs sont ici plus grandes mais aussi plus disparates. Cela s'explique par le fait que nos musiques d'études proviennent de différents groupes. Enfin, les deux blocs symétriques restant sont les distances qui séparent les musiques classiques des musiques punk. Les valeurs sont encore plus dispersées, mais on note très clairement une moyenne plus élevée, aux alentours de 50. Pour rendre ces résultats plus clairs, nous transformons cette matrice en dendrogramme grâce à MatLab. L'image 2.4 illustre les résultats obtenus.

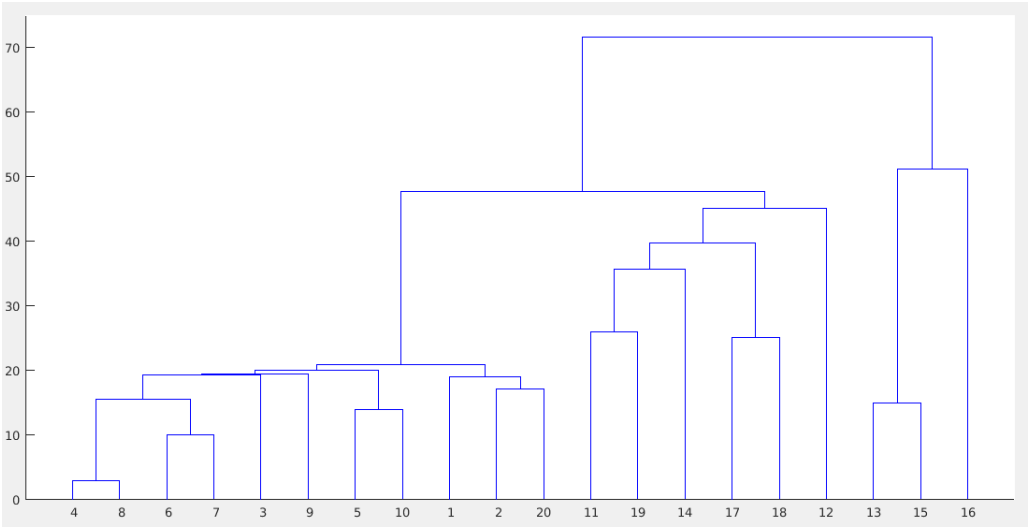

FIGURE 2.4 – Dendrogramme des distances

L'arbre distingue nettement trois groupes de compositions. Le premier est constitué de toutes les musiques classiques et de *Welcome To Paradise* de Green Day. Ce morceau est harmoniquement plus recherché que les autres morceaux punk, ce qui l'a distingué dans ce groupe. La deuxième catégorie contient le reste des morceaux punk à l'exception de *Beat On The Brat*, *Blitzkrieg Bop* et de *I Fought The Law*, qui forment à part le troisième groupe. En effet, ces trois morceaux sont particulièrement pauvres harmoniquement parlant, plusieurs notes n'apparaissent jamais dans ces musiques tandis qu'une hauteur particulière se dresse largement au-dessus des autres ; à titre d'exemple, *Beat On The Brat* est composée à 44 pour cent de la note si.

Chapitre 3

Conclusion et horizons

Les résultats présentés à la section précédente sont encourageants. Les musiques classiques et punk ont été séparées quasiment sans faute, et l'algorithme a même pu distinguer les musiques particulièrement pauvres des autres.

Tout le processus de calcul peut s'automatiser pour être réalisé à grande vitesse : le calcul du dendrogramme à partir des listes de fréquences prend environ 13 secondes sur notre machine, à l'aide d'un hardware et d'un code source très peu optimisé. L'analyse fréquentielle des morceaux reste encore trop lente à l'heure actuelle : l'utilisation de macros Visual Basic est à éviter pour ce genre de calculs à la chaîne. La transmission des données depuis Excel vers MatLab a été faite à la main, ce qui a pris un temps considérable ; il reste néanmoins possible d'automatiser ce transfert de données.

L'approche de l'algorithme peut encore être modifiée pour mieux coller à la réalité. Considérer non pas les fréquences d'apparition des notes mais leur temps de jeu est particulièrement judicieux. Représenter les accords par des sommets du Tonnetz auxquels sont attachés des scalaires au même titre que les notes est aussi très pertinent. Enfin, l'aspect dynamique de la pièce est à prendre en compte : dans une composition, une note n'existe jamais seule, elle fait partie intégrante d'un groupement de notes formé de ses voisines et qui constitue une phrase musicale. Prendre compte de cet élément est important pour la généralisation des idées présentées ici. Sans ces différents aspects, il semble difficile de séparer des styles harmoniquement proches, comme peuvent l'être la *musique atonale* ou *sérielle* et le *free jazz* ou le *modal jazz*.

Nous avons travaillé avec des fichiers MIDI tout au long de cette étude, ce qui annihile les notions de timbres et de micro variations. A lui seul, le choix des instruments aide beaucoup pour classer les morceaux par style, et cela n'est pas retranscrit dans cette étude. Une grosse machinerie calculatoire ayant pour fondation l'analyse et le traitement du signal serait à mettre en place pour traiter des enregistrements sonores ou des prestations live. Les résultats seraient alors beaucoup plus riches de détails.

Le traitement automatique du style par les techniques présentées ici semblent encore loin de nous. La conjonction de ces méthodes avec des processus de *machine learning* sont une piste qu'il nous semble nécessaire de creuser.

Appendices

Annexe A

Code VBA pour l'analyse fréquentielle d'un fichier MIDI

Le code qui suit présuppose la conversion du fichier MIDI à analyser en fichier CSV. Nous avons utilisé le logiciel GNMIDI pour cela. Ce software crée la liste des commandes dans un format intelligible pour l'humain. La macrocommande que l'on présente ici est un contre-exemple parfait de code bien écrit : il réalise douze fois la même chose en spaghetti, ne déclare pas toutes ses variables et réalise quantité d'opérations inutiles. Il a pris cette forme à force de modifications *ad hoc* successives.

```
Sub Analyse()

'---------------------
ActiveSheet.Range("D3:G15").Clear
'---------------------

Worksheets(1).Activate

Dim Command As String
Dim Channel As String

For Octave = 0 To 10

If ActiveSheet.UsedRange.Find("d" & Octave) Is Nothing Then
GoTo Skipd
End If

FirstRow = ActiveSheet.UsedRange.Find("d" & Octave).Row
ActiveSheet.UsedRange.Find("d" & Octave).Select
Index = 1

While Index <> 0
```

```
Lookd:
ActiveSheet.UsedRange.FindNext(After:=ActiveCell).Select

If ActiveCell.Row = FirstRow Then
Index = 0
Range("E5").Value = Range("E5").Value
GoTo Skipd
End If

'On vérifie que l'on est pas sur la chaîne dédiée à la batterie.
Command = ActiveCell.Value
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Channel = Mid(Command, 1, 2)

If Channel = 10 Then
GoTo Lookd
End If

Range("E5").Value = Range("E5").Value + 1

Wend


Skipd:
Next

'On prend en compte que chaque note apparait en On et en Off
Range("E5").Value = Range("E5").Value / 2

Range("E5").Select


'-----------------------------------------------------------------

For Octave = 0 To 10

If ActiveSheet.UsedRange.Find("c" & Octave) Is Nothing Then
GoTo Skipc
End If

FirstRow = ActiveSheet.UsedRange.Find("c" & Octave).Row
ActiveSheet.UsedRange.Find("c" & Octave).Select
Index = 1

While Index <> 0
```

```
Lookc:
ActiveSheet.UsedRange.FindNext(After:=ActiveCell).Select


If ActiveCell.Row = FirstRow Then
Index = 0
Range("E3").Value = Range("E3").Value
GoTo Skipc
End If

Command = ActiveCell.Value
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Channel = Mid(Command, 1, 2)

If Channel = 10 Then
GoTo Lookc
End If

Range("E3").Value = Range("E3").Value + 1

Wend


Skipc:
Next

Range("E3").Value = Range("E3").Value / 2

Range("E3").Select


'-----------------------------------------------------------------

For Octave = 0 To 10

If ActiveSheet.UsedRange.Find("c#" & Octave) Is Nothing Then
GoTo Skipcs
End If

FirstRow = ActiveSheet.UsedRange.Find("c#" & Octave).Row
ActiveSheet.UsedRange.Find("c#" & Octave).Select
Index = 1

While Index <> 0
```

```
Lookcs:
ActiveSheet.UsedRange.FindNext(After:=ActiveCell).Select


If ActiveCell.Row = FirstRow Then
Index = 0
Range("E4").Value = Range("E4").Value
GoTo Skipcs
End If

Command = ActiveCell.Value
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Channel = Mid(Command, 1, 2)

If Channel = 10 Then
GoTo Lookcs
End If

Range("E4").Value = Range("E4").Value + 1

Wend


Skipcs:
Next

Range("E4").Value = Range("E4").Value / 2

Range("E4").Select


'--------------------------------------------------------------------

For Octave = 0 To 10

If ActiveSheet.UsedRange.Find("d#" & Octave) Is Nothing Then
GoTo Skipds
End If

FirstRow = ActiveSheet.UsedRange.Find("d#" & Octave).Row
ActiveSheet.UsedRange.Find("d#" & Octave).Select
Index = 1

While Index <> 0


Lookds:
```

```
ActiveSheet.UsedRange.FindNext(After:=ActiveCell).Select


If ActiveCell.Row = FirstRow Then
Index = 0
Range("E6").Value = Range("E6").Value
GoTo Skipds
End If

Command = ActiveCell.Value
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Channel = Mid(Command, 1, 2)

If Channel = 10 Then
GoTo Lookds
End If

Range("E6").Value = Range("E6").Value + 1

Wend


Skipds:
Next

Range("E6").Value = Range("E6").Value / 2

Range("E6").Select


'-----------------------------------------------------------------

For Octave = 0 To 10

If ActiveSheet.UsedRange.Find("e" & Octave) Is Nothing Then
GoTo Skipe
End If

FirstRow = ActiveSheet.UsedRange.Find("e" & Octave).Row
ActiveSheet.UsedRange.Find("e" & Octave).Select
Index = 1


While Index <> 0


Looke:
```

```
ActiveSheet.UsedRange.FindNext(After:=ActiveCell).Select


If ActiveCell.Row = FirstRow Then
Index = 0


'On prend en compte un petit artefact à la première ligne.
 If ActiveCell.Row = 1 Then
 GoTo MiniSkip
 End If
Range("E7").Value = Range("E7").Value
GoTo Skipe
End If

If ActiveCell.Row = 1 Then
GoTo Looke
End If

Command = ActiveCell.Value
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Channel = Mid(Command, 1, 2)

If Channel = 10 Then
GoTo Looke
End If

Range("E7").Value = Range("E7").Value + 1

MiniSkip:
Wend


Skipe:
Next

Range("E7").Value = Range("E7").Value / 2

Range("E7").Select


'-----------------------------------------------------------------

For Octave = 0 To 10

If ActiveSheet.UsedRange.Find("f" & Octave) Is Nothing Then
GoTo Skipf
```

```
End If

FirstRow = ActiveSheet.UsedRange.Find("f" & Octave).Row
ActiveSheet.UsedRange.Find("f" & Octave).Select
Index = 1

While Index <> 0


Lookf:
ActiveSheet.UsedRange.FindNext(After:=ActiveCell).Select


If ActiveCell.Row = FirstRow Then
Index = 0
Range("E8").Value = Range("E8").Value
GoTo Skipf
End If

Command = ActiveCell.Value
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Channel = Mid(Command, 1, 2)

If Channel = 10 Then
GoTo Lookf
End If

Range("E8").Value = Range("E8").Value + 1

Wend


Skipf:
Next

Range("E8").Value = Range("E8").Value / 2

Range("E8").Select


'-----------------------------------------------------------------------

For Octave = 0 To 10

If ActiveSheet.UsedRange.Find("f#" & Octave) Is Nothing Then
GoTo Skipfs
End If
```

```
FirstRow = ActiveSheet.UsedRange.Find("f#" & Octave).Row
ActiveSheet.UsedRange.Find("f#" & Octave).Select
Index = 1

While Index <> 0


Lookfs:
ActiveSheet.UsedRange.FindNext(After:=ActiveCell).Select


If ActiveCell.Row = FirstRow Then
Index = 0
Range("E9").Value = Range("E9").Value
GoTo Skipfs
End If

Command = ActiveCell.Value
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Channel = Mid(Command, 1, 2)

If Channel = 10 Then
GoTo Lookfs
End If

Range("E9").Value = Range("E9").Value + 1

Wend


Skipfs:
Next

Range("E9").Value = Range("E9").Value / 2

Range("E9").Select


'-----------------------------------------------------------------

For Octave = 0 To 10

If ActiveSheet.UsedRange.Find("g" & Octave) Is Nothing Then
GoTo Skipg
End If
```

```
FirstRow = ActiveSheet.UsedRange.Find("g" & Octave).Row
ActiveSheet.UsedRange.Find("g" & Octave).Select
Index = 1

While Index <> 0


Lookg:
ActiveSheet.UsedRange.FindNext(After:=ActiveCell).Select


If ActiveCell.Row = FirstRow Then
Index = 0
Range("E10").Value = Range("E10").Value
GoTo Skipg
End If

Command = ActiveCell.Value
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Channel = Mid(Command, 1, 2)

If Channel = 10 Then
GoTo Lookg
End If

Range("E10").Value = Range("E10").Value + 1

Wend


Skipg:
Next

Range("E10").Value = Range("E10").Value / 2

Range("E10").Select


'-----------------------------------------------------------------

For Octave = 0 To 10

If ActiveSheet.UsedRange.Find("g#" & Octave) Is Nothing Then
GoTo Skipgs
End If

FirstRow = ActiveSheet.UsedRange.Find("g#" & Octave).Row
```

```
ActiveSheet.UsedRange.Find("g#" & Octave).Select
Index = 1

While Index <> 0


Lookgs:
ActiveSheet.UsedRange.FindNext(After:=ActiveCell).Select


If ActiveCell.Row = FirstRow Then
Index = 0
GoTo Skipgs
End If

Command = ActiveCell.Value
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Channel = Mid(Command, 1, 2)

If Channel = 10 Then
GoTo Lookgs
End If

Range("E11").Value = Range("E11").Value + 1

Wend


Skipgs:
Next

Range("E11").Value = Range("E11").Value / 2

Range("E11").Select


'---------------------------------------------------------------------

For Octave = 0 To 10

If ActiveSheet.UsedRange.Find("a" & Octave) Is Nothing Then
GoTo Skipa
End If

FirstRow = ActiveSheet.UsedRange.Find("a" & Octave).Row
ActiveSheet.UsedRange.Find("a" & Octave).Select
Index = 1
```

```
While Index <> 0


Looka:
ActiveSheet.UsedRange.FindNext(After:=ActiveCell).Select


If ActiveCell.Row = FirstRow Then
Index = 0
GoTo Skipa
End If

Command = ActiveCell.Value
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Channel = Mid(Command, 1, 2)

If Channel = 10 Then
GoTo Looka
End If

Range("E12").Value = Range("E12").Value + 1

Wend


Skipa:
Next

Range("E12").Value = Range("E12").Value / 2

Range("E12").Select


'-----------------------------------------------------------------

For Octave = 0 To 10

If ActiveSheet.UsedRange.Find("a#" & Octave) Is Nothing Then
GoTo Skipas
End If

FirstRow = ActiveSheet.UsedRange.Find("a#" & Octave).Row
ActiveSheet.UsedRange.Find("a#" & Octave).Select
Index = 1

While Index <> 0
```

```
Lookas:
ActiveSheet.UsedRange.FindNext(After:=ActiveCell).Select


If ActiveCell.Row = FirstRow Then
Index = 0
GoTo Skipas
End If

Command = ActiveCell.Value
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Channel = Mid(Command, 1, 2)

If Channel = 10 Then
GoTo Lookas
End If

Range("E13").Value = Range("E13").Value + 1

Wend


Skipas:
Next

Range("E13").Value = Range("E13").Value / 2

Range("E13").Select


'-----------------------------------------------------------------

For Octave = 0 To 10

If ActiveSheet.UsedRange.Find("b" & Octave) Is Nothing Then
GoTo Skipb
End If

FirstRow = ActiveSheet.UsedRange.Find("b" & Octave).Row
ActiveSheet.UsedRange.Find("b" & Octave).Select
Index = 1

While Index <> 0
```

```
Lookb:
ActiveSheet.UsedRange.FindNext(After:=ActiveCell).Select


If ActiveCell.Row = FirstRow Then
Index = 0
GoTo Skipb
End If

Command = ActiveCell.Value
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Command = Right(Command, Len(Command) - InStr(Command, ","))
Channel = Mid(Command, 1, 2)

If Channel = 10 Then
GoTo Lookb
End If

Range("E14").Value = Range("E14").Value + 1

Wend


Skipb:
Next

Range("E14").Value = Range("E14").Value / 2

Range("E14").Select


'-----------------------------------------------------------------------
'Mise en forme des résultats
'-----------------------------------------------------------------------

ActiveSheet.Range("D3").Value = "C"
ActiveSheet.Range("D4").Value = "C#"
ActiveSheet.Range("D5").Value = "D"
ActiveSheet.Range("D6").Value = "D#"
ActiveSheet.Range("D7").Value = "E"
ActiveSheet.Range("D8").Value = "F"
ActiveSheet.Range("D9").Value = "F#"
ActiveSheet.Range("D10").Value = "G"
ActiveSheet.Range("D11").Value = "G#"
ActiveSheet.Range("D12").Value = "A"
ActiveSheet.Range("D13").Value = "A#"
ActiveSheet.Range("D14").Value = "B"
ActiveSheet.Range("D15").Value = "Total"
```

```
ActiveSheet.Range("E15").Value = Excel.WorksheetFunction.Sum(Range("E3:E14"))

For Indx = 3 To 15
ActiveSheet.Range("G" & Indx).Formula =_
ActiveSheet.Range("E" & Indx).Value * 100 / ActiveSheet.Range("E15").Value
Next

'-----------------------------------------

End Sub
```

Annexe B

Code SAGE pour la visualisation du Tonnetz

Les douze premières affectations correspondent à la hauteur de chacune des notes du Tonnetz : ce code donnera un Tonnetz isotrope, illustré par l'image 2.1. Cependant, en utilisant des données issues de compositions on obtiendra des valeurs et donc un Tonnetz différent.

```
c=0
db=0
d=0
eb=0
e=0
f=0
gb=0
g=0
ab=0
a=0
bb=0
b=0

C=(0, 0, c)
Db = (-1-cos(pi/3), sin(pi/3), db)
D=(2, 0, d)
Eb=(cos(pi/3), sin(pi/3), eb)
E=(cos(pi/3), -sin(pi/3), e)
F=(-1, 0, f)
Gb=(2*cos(pi/3), 2*sin(pi/3), gb)
G=(1, 0, g)
Ab=(-cos(pi/3), sin(pi/3), e)
A=(-cos(pi/3), -sin(pi/3), a)
Bb=(-2, 0, bb)
B=(1+cos(pi/3), -sin(pi/3), b)

Bb2=(1+cos(pi/3), sin(pi/3), bb)
B2=(0, 2*sin(pi/3), b)
```

```
E2=(-2*cos(pi/3), 2*sin(pi/3), e)
D2=(-1-cos(pi/3), -sin(pi/3), d )
Gb2=(-2*cos(pi/3), -2*sin(pi/3), gb)
Db2=(0, -2*sin(pi/3), db)
Ab2=(2*cos(pi/3), -2*sin(pi/3), ab)
Db3=(1+2*cos(pi/3), 2*sin(pi/3), db)
Gb3=(2+cos(pi/3), -sin(pi/3), gb)

CM=polygon([C, G, E], rgbcolor=(127, 0, 0))
Cm=polygon([C, G, Eb])
GM=polygon([G, D, B], rgbcolor=(127, 0, 0))
Gm=polygon([G, D, Bb2])
FM=polygon([F, C, A], rgbcolor=(127, 0, 0))
Fm=polygon([F, C, Ab])
BbM=polygon([Bb, F, D2], rgbcolor=(127, 0, 0))
Bbm=polygon([Bb, F, Db])
DM=polygon([D2, Gb2, A], rgbcolor=(127, 0, 0))
Dm=polygon([D2, F, A])
AM=polygon([A, Db2, E], rgbcolor=(127, 0, 0))
Am=polygon([A, C, E])
EM=polygon([E, Ab2, B], rgbcolor=(127, 0, 0))
Em=polygon([E, G, B])
DbM=polygon([Db, Ab, F], rgbcolor=(127, 0, 0))
Dbm=polygon([Db, E2, Ab])
AbM=polygon([Ab, C, Eb], rgbcolor=(127, 0, 0))
Abm=polygon([Ab, B2, Eb])
EbM=polygon([Eb, G, Bb2], rgbcolor=(127, 0, 0))
Ebm=polygon([Eb, Gb, Bb2])
BM =polygon([B2, Gb, Eb], rgbcolor=(127, 0, 0))
Bm =polygon([B, Gb3, D])
GbM=polygon([Gb, Bb2, Db3], rgbcolor=(127, 0, 0))
Gbm = polygon([Gb2, A, Db2])

EM2 = polygon([E2, B2, Ab], rgbcolor=(127, 0, 0))
Dbm2 =polygon([Db2, E, Ab2])

Out = CM +Cm +GM +Gm+FM+Fm+Bbm+BbM+DM+Dm+AM+Am+EM+Em+DbM+Dbm+AbM+Abm+EbM+Ebm+BM
+Gbm+EM2+Dbm2+GbM+Bm
Out += line([Bb, Db, F, Ab, C, Eb, G, Bb2, D, B, G, E, C, A, F, D2, Bb],
 rgbcolor=(0, 0, 0))
Out += line([Db, E2, Ab, B2, Eb, Gb, Bb2], rgbcolor=(0, 0, 0))
Out += line([D2, Gb2, A, Db2, E, Ab2, B], rgbcolor=(0, 0, 0))
Out += line([E2, B2, Gb], rgbcolor=(0, 0, 0))
Out += line([Db, Ab, Eb, Bb2], rgbcolor=(0, 0, 0))
Out += line([Bb, F, C, G, D], rgbcolor=(0, 0, 0))
Out += line([D2, A, E, B], rgbcolor=(0, 0, 0))
Out += line([Gb2, Db2, Ab2], rgbcolor=(0, 0, 0))
Out += line([Gb, Db3, Bb2], rgbcolor=(0, 0, 0))
Out += line([B, Gb3, D], rgbcolor=(0, 0, 0))
Out.show()
```

Annexe C

Code C++ pour le calcul d'homologie persistante avec GUDHI

La compilation de ce programme nécessite l'installation complète de GUDHI. Ce programme prend en entrée les douze valeurs correspondant aux douze notes de $\mathbb{Z}/12\mathbb{Z}$.

```
#include <gudhi/graph_simplicial_complex.h>
#include <gudhi/Simplex_tree.h>
#include <gudhi/Persistent_cohomology.h>
#include <fstream>
#include <iostream>
#include <ctime>
#include <utility>
#include <vector>

// Types definition
using Simplex_tree = Gudhi::Simplex_tree<>;
using Filtration_value = Simplex_tree::Filtration_value;
using Field_Zp = Gudhi::persistent_cohomology::Field_Zp;
using Persistent_cohomology = Gudhi::persistent_cohomology::
Persistent_cohomology<Simplex_tree, Field_Zp >;
using typeVectorVertex = std::vector< Simplex_tree::Vertex_handle >;

void usage(char * const progName) {
  std::cerr << "Usage: " << progName << "
coeff_field_characteristic[integer > 0] min_persistence[float >= -1.0]\n";
  exit(-1);
}

//fonction MAX

float MAX(float u, float v, float w)
```

```
{
return std::max(std::max(u, v), std::max(v, w));
}

//Initialisation du Tonnetz

int C = 0;
int Db = 1;
int D = 2;
int Eb = 3;
int E = 4;
int F = 5;
int Gb = 6;
int G = 7;
int Ab = 8;
int A = 9;
int Bb = 10;
int B = 11;


int main(int argc, char * const argv[]) {
  // program args management
  if (argc != 3) {
    std::cerr << "Error: Number of arguments (" << argc << ") is not correct\n";
    usage(argv[0]);
  }

  int coeff_field_characteristic = 0;
  int returnedScanValue = sscanf(argv[1], "%d", &coeff_field_characteristic);
  if ((returnedScanValue == EOF) || (coeff_field_characteristic <= 0)) {
    std::cerr << "Error: " << argv[1] << " is not correct\n";
    usage(argv[0]);
  }

  Filtration_value min_persistence = 0.0;
  returnedScanValue = sscanf(argv[2], "%lf", &min_persistence);
  if ((returnedScanValue == EOF) || (min_persistence < -1.0)) {
    std::cerr << "Error: " << argv[2] << " is not correct\n";
    usage(argv[0]);
  }

  //Entrée des frequences

float c;
float db;
float d;
float eb;
float e;
float f;
float gb;
```

```
float g;
float ab;
float a;
float bb;
float b;

std::cin >> c;
std::cin >> db;
std::cin >> d;
std::cin >> eb;
std::cin >> e;
std::cin >> f;
std::cin >> gb;
std::cin >> g;
std::cin >> ab;
std::cin >> a;
std::cin >> bb;
std::cin >> b;

  //Creation du complexe simplicial torique

  Simplex_tree st;
  typeVectorVertex SimplexVector;

  st.insert_simplex_and_subfaces({E, B, Ab}, MAX(e, b, ab));
  st.insert_simplex_and_subfaces({Eb, B, Ab}, MAX(eb, b, ab));
  st.insert_simplex_and_subfaces({Eb, B, Gb}, MAX(eb, b, gb));
  st.insert_simplex_and_subfaces({Eb, Bb, Gb}, MAX(eb, bb, gb));
  st.insert_simplex_and_subfaces({Db, Bb, Gb}, MAX(db, bb, gb));

  st.insert_simplex_and_subfaces({Bb, Db, F}, MAX(db, bb, f));
  st.insert_simplex_and_subfaces({F, Db, Ab}, MAX(f, db, ab));
  st.insert_simplex_and_subfaces({F, C, Ab}, MAX(f, ab, c));
  st.insert_simplex_and_subfaces({Eb, C, Ab}, MAX(eb, c, ab));
  st.insert_simplex_and_subfaces({C, Eb, G}, MAX(c, eb, g));
  st.insert_simplex_and_subfaces({G, Eb, Bb}, MAX(g, eb, bb));
  st.insert_simplex_and_subfaces({G, Bb, D}, MAX(g, bb, d));

  st.insert_simplex_and_subfaces({Bb, D, F}, MAX(bb, d, f));
  st.insert_simplex_and_subfaces({D, F, A}, MAX(d, f, a));
  st.insert_simplex_and_subfaces({A, F, C}, MAX(a, f, c));
  st.insert_simplex_and_subfaces({C, E, A}, MAX(c, e, a));
  st.insert_simplex_and_subfaces({C, E, G}, MAX(c, e, g));
  st.insert_simplex_and_subfaces({E, B, G}, MAX(b, e, g));
  st.insert_simplex_and_subfaces({G, D, B}, MAX(d, b, g));
  st.insert_simplex_and_subfaces({B, D, Gb}, MAX(b, d, gb));

  st.insert_simplex_and_subfaces({D, A, Gb}, MAX(d, a, gb));
  st.insert_simplex_and_subfaces({A, D, Gb}, MAX(a, db, gb));
  st.insert_simplex_and_subfaces({Db, E, A}, MAX(db, e, a));
```

```
  st.insert_simplex_and_subfaces({Db, E, Ab}, MAX(db, e, ab));

  st.set_dimension(3);
  st.set_filtration(0.5);

  std::cout << "The complex contains " << st.num_simplices() << " simplices - "
<< st.num_vertices() << " vertices "
      << std::endl;
  std::cout << "   - dimension " << st.dimension() << "   - filtration " <<
st.filtration() << std::endl;
  std::cout << std::endl << std::endl << "Iterator on Simplices in the
 filtration, with [filtration value]:"
      << std::endl;
  std::cout << "*****************************************" << std::endl;
  std::cout << "strict graph G { " << std::endl;

  for (auto f_simplex : st.filtration_simplex_range()) {
    std::cout << "   " << "[" << st.filtration(f_simplex) << "] ";
    for (auto vertex : st.simplex_vertex_range(f_simplex)) {
      std::cout << static_cast<int>(vertex) << " -- ";
    }
    std::cout << ";" << std::endl;
  }

  std::cout << "}" << std::endl;
  std::cout << "*****************************************" << std::endl;

  // Compute the persistence diagram of the complex
  Persistent_cohomology pcoh(st);
  // initializes the coefficient field for homology
  pcoh.init_coefficients(coeff_field_characteristic);

  pcoh.compute_persistent_cohomology(min_persistence);


 // Output the diagram in filediag
  pcoh.output_diagram();

  return 0;
}
```

Annexe D

Code MatLab pour le calcul d'homologie persistante et de bottleneck distance

La compilation de ce programme nécessite d'avoir préalablement lancer le script *load_javaplex* du package *javaplex* de MatLab. Les valeurs des vecteurs des fréquences $v1$ et $v2$ sont à adapter en fonction des compositions à comparer.

```
clc; clear; close all;
import edu.stanford.math.plex4.*;

%defining new triple max function
max = @(x,y,z) (max(max(x, y), max(y, z)));

% get new ExplicitSimplexStreams
streamA = api.Plex4.createExplicitSimplexStream();
streamB = api.Plex4.createExplicitSimplexStream();

%defining frequency vectors
v1=zeros(1, 12);
v2=zeros(1, 12);

v1=[0, 14, 2, 5, 3, 5, 7, 8, 4, 15 ,84, 10];
v2=[45, 16, 18, 98, 45, 0, 2, 45, 62, 74 ,85, 48];

%defining usable names
c1=v1(1);
db1=v1(2);
d1=v1(3);
eb1=v1(4);
e1=v1(5);
f1=v1(6);
gb1=v1(7);
g1=v1(8);
```

```
ab1=v1(9);
a1=v1(10);
bb1=v1(11);
b1=v1(12);


c2=v2(1);
db2=v2(2);
d2=v2(3);
eb2=v2(4);
e2=v2(5);
f2=v2(6);
gb2=v2(7);
g2=v2(8);
ab2=v2(9);
a2=v2(10);
bb2=v2(11);
b2=v2(12);

% construct complex A
streamA.addVertex(1, c1);
streamA.addVertex(2, db1);
streamA.addVertex(3, d1);
streamA.addVertex(4, eb1);
streamA.addVertex(5, e1);
streamA.addVertex(6, f1);
streamA.addVertex(7, gb1);
streamA.addVertex(8, g1);
streamA.addVertex(9, ab1);
streamA.addVertex(10, a1);
streamA.addVertex(11, bb1);
streamA.addVertex(12, b1);

streamA.addElement([1, 4, 8], max(c1, eb1, g1));
streamA.addElement([1, 5, 8], max (c1, e1, g1));
streamA.addElement([2, 5, 9], max (db1, e1, ab1));
streamA.addElement([2, 6, 9], max(db1, f1, ab1));
streamA.addElement([3, 6, 10], max(d1, f1, a1));
streamA.addElement([3, 7, 10], max(d1, gb1, a1));
streamA.addElement([4, 7, 11], max(eb1, gb1, bb1));
streamA.addElement([4, 8, 11], max(eb1, g1, bb1));
streamA.addElement([5, 8, 12], max(e1, g1, b1));
streamA.addElement([5, 9, 12], max(e1, ab1, b1));
streamA.addElement([6, 9, 1], max(f1, ab1, c1));
streamA.addElement([6, 10, 1], max(f1, a1, c1));
streamA.addElement([7, 10, 2], max(gb1, a1, db1));
streamA.addElement([7, 11, 2], max(gb1, bb1, db1));
streamA.addElement([8, 11, 3], max(g1, bb1, d1));
streamA.addElement([8, 12, 3], max(g1, b1, d1));
streamA.addElement([9, 12, 4], max(ab1, b1, eb1));
```

```
streamA.addElement([9, 1, 4], max(ab1, c1, eb1));
streamA.addElement([10, 1, 5], max(a1, c1, e1));
streamA.addElement([10, 2, 5], max(a1, db1, e1));
streamA.addElement([11, 2, 6], max(bb1, db1, f1));
streamA.addElement([11, 3, 6], max(bb1, d1, f1));
streamA.addElement([12, 3, 7], max(b1, d1, gb1));
streamA.addElement([12, 4, 7], max(b1, eb1, gb1));
streamA.finalizeStream();

% construct complex B
streamB.addVertex(1, c2);
streamB.addVertex(2, db2);
streamB.addVertex(3, d2);
streamB.addVertex(4, eb2);
streamB.addVertex(5, e2);
streamB.addVertex(6, f2);
streamB.addVertex(7, gb2);
streamB.addVertex(8, g2);
streamB.addVertex(9, ab2);
streamB.addVertex(10, a2);
streamB.addVertex(11, bb2);
streamB.addVertex(12, b2);

streamB.addElement([1, 4, 8], max(c2, eb2, g2));
streamB.addElement([1, 5, 8], max (c2, e2, g2));
streamB.addElement([2, 5, 9], max (db2, e2, ab2));
streamB.addElement([2, 6, 9], max(db2, f2, ab2));
streamB.addElement([3, 6, 10], max(d2, f2, a2));
streamB.addElement([3, 7, 10], max(d2, gb2, a2));
streamB.addElement([4, 7, 11], max(eb2, gb2, bb2));
streamB.addElement([4, 8, 11], max(eb2, g2, bb2));
streamB.addElement([5, 8, 12], max(e2, g2, b2));
streamB.addElement([5, 9, 12], max(e2, ab2, b2));
streamB.addElement([6, 9, 1], max(f2, ab2, c2));
streamB.addElement([6, 10, 1], max(f2, a2, c2));
streamB.addElement([7, 10, 2], max(gb2, a2, db2));
streamB.addElement([7, 11, 2], max(gb2, bb2, db2));
streamB.addElement([8, 11, 3], max(g2, bb2, d2));
streamB.addElement([8, 12, 3], max(g2, b2, d2));
streamB.addElement([9, 12, 4], max(ab2, b2, eb2));
streamB.addElement([9, 1, 4], max(ab2, c2, eb2));
streamB.addElement([10, 1, 5], max(a2, c2, e2));
streamB.addElement([10, 2, 5], max(a2, db2, e2));
streamB.addElement([11, 2, 6], max(bb2, db2, f2));
streamB.addElement([11, 3, 6], max(bb2, d2, f2));
streamB.addElement([12, 3, 7], max(b2, d2, gb2));
streamB.addElement([12, 4, 7], max(b2, eb2, gb2));
streamB.finalizeStream();

% get persistence algorithm over Z/2Z
```

```
persistence = api.Plex4.getModularSimplicialAlgorithm(3, 2);

% compute and print the intervals
intervalsA = persistence.computeIntervals(streamA)
intervalsB = persistence.computeIntervals(streamB)

% compute the bottleneck distances
intervalsA_dim0=intervalsA.getIntervalsAtDimension(0);
intervalsB_dim0=intervalsB.getIntervalsAtDimension(0);
bottleneck_distance_dim0 = edu.stanford.math.plex4.bottleneck.BottleneckDistance
.computeBottleneckDistance(intervalsA_dim0,intervalsB_dim0)
intervalsA_dim1=intervalsA.getIntervalsAtDimension(1);
intervalsB_dim1=intervalsB.getIntervalsAtDimension(1);
bottleneck_distance_dim1 = edu.stanford.math.plex4.bottleneck.BottleneckDistance
.computeBottleneckDistance(intervalsA_dim1,intervalsB_dim1)
intervalsA_dim2=intervalsA.getIntervalsAtDimension(2);
intervalsB_dim2=intervalsB.getIntervalsAtDimension(2);
bottleneck_distance_dim2 = edu.stanford.math.plex4.bottleneck.BottleneckDistance
.computeBottleneckDistance(intervalsA_dim2,intervalsB_dim2)
Total_bottleneck_distance = bottleneck_distance_dim2+
bottleneck_distance_dim1+bottleneck_distance_dim0
```

Bibliographie

[1] Mattia BERGOMI. *Dynamical and topological tools for (modern) music analysis*. PhD thesis, UPMC, 2015.

[2] Louis BIGOT. *Représentations symboliques musicales et calcul spatial*. PhD thesis, Université Paris-Est LACL/IRCAM, 2013.

[3] Herbert EDELSBRUNNER and John HARER. Persistent homology - a survey. http ://www.maths.ed.ac.uk/∼aar/papers/edelhare.pdf.

[4] Robert GUERIN. *MIDI Power ! : The Comprehensive Guide*. Cengage Learning PTR, 2005.

[5] Franck JEDRZEJEWSKI. *Mathematical Theory of Music*. Delatour France - IRCAM, 2006.

[6] Guerino MAZZOLA and Yun-Kang AHN. *La vérité du beau dans la musique*. Delatour France - IRCAM, 2007.