

Projet PAM AutoDJ, Auto-Mashup

Maxime ARBISA, Samuel BELL-BELL, Pierre MAHÉ, Léo THIRIFAYS

5 février 2016



Annexe - Explication du code

Partie analyse

- *function* *accordage* = *f_tuning(filename)* : fonction calculant le détunage "*accordage*" en % à partir d'un morceau "*filename*".
- *function* *F0* = *f_detect_F0(Xk, Nfft, H, Fs, Fmin, Fmax)* : fonction détectant la fréquence fondamentale F0 à partir du produit spectral.
- *script* *AnalyseMusic.m* : script réalisant le calcul de la base harmonique de référence des Beatles (parcours de la base de Beatles et extraction des matrices d'observations chroma pour chaque morceau avec la fonction "*extractChroma*"). Cette 1^{ère} partie renvoie un dictionnaire "*Observations*" ayant pour clés le nom des morceaux, et pour valeurs leurs matrices d'observations chroma. On effectue ensuite le mapping avec les annotations grâce à la fonction "*mapping*". Puis on compare les morceaux de notre bibliothèque musicale pour en extraire leur suite d'accords (avec la fonction "*extractChords*").
- *function* *obs_m* = *extractChroma(data_v, sr_hz, L_n, STEP_n, detune)* : fonction qui, pour chaque morceau étudié "*data_v*", crée une base de chroma correspondant au détunage "*detune*", puis crée la matrice d'observations chroma "*obs_m*" correspondante. Le paramètre "*display*" est mis à 0. 1 permet l'affichage de la base de filtres chroma, et de la matrice de vecteurs d'observations.
- *function* [*Accords_morceaux*, *Accords_mat*, *Proba*] = *mapping(Observations, STEP_sec)* : fonction parcourant le dossier des annotations et les mappant avec les matrices d'observations chroma. La sortie de cette fonction se compose de 3 dictionnaires servant à obtenir différentes bases de référence de travail. Ils se composent de :
 - *Accords_morceaux* : dictionnaire contenant pour chaque morceau un dictionnaire avec les accords du morceau et leurs chromas associés. En pratique, on utilise une seule et unique base. Peut servir pour augmenter la base de référence, car il tient compte de la façon de jouer propre à chaque morceau.
 - *Accords_mat* : Dictionnaire contenant pour chaque accord la matrice des vecteurs d'observations chroma de cet accord. Sera utile pour la partie avec les HMM.
 - *Proba* : Dictionnaire contenant les probabilités de passage d'un accord à l'autre, calculé d'après la base des Beatles (est utile pour la partie HMM).A la fin de cette fonction, on calcule aussi le dictionnaire "*Accords*" contenant pour chacun des 24 accords un seul vecteur chroma moyenné. Il est obtenu à partir de la moyenne sur les colonnes des matrices de chromas de "*Accords_mat*" pour chaque accord.

- *function* `c_chroma_ref = generateChordBase()` : fonction créant une base de référence de chromas obtenue par rotation des chromas de deux accords de Do et de Dom joués par nos soins. C'est une alternative à l'étude avec la base des Beatles. Elle donne cependant de moins bons résultats que la base des Beatles.
- *[list_chords, list_chords_2, list_times] = extractChords(data_v, sr_hz, L_n, STEP_n, detune, Accords, Proba)* : pour chaque morceau, on extrait la matrice d'observations chromas (avec une base chroma recentrée avec le "détunage", puis nous comparons par distance cosinusoidale chaque chroma à la base de chromas de référence "Accords" pour choisir l'accord. L'accord avec la distance la plus proche de 1 est choisi. La fonction renvoie la liste d'accords "list_chords", et la liste des temps correspondant au début et à la fin de chaque accord. On calcule aussi les distances pour chaque accord, et pour chaque chroma du morceau analysé qu'on met dans une matrice. On fait ensuite un filtrage passe-bas pour enlever les trop grandes fluctuations, et on récupère le max sur chaque colonne pour localiser l'accord le plus ressemblant. La suite d'accords filtrés est dans la liste "list_chords_2". Le paramètre "display" permet d'observer les observations chromas pour chaque morceau.
- *function [path_v] = HMM(Accords_mat, obs_m)* : fonction réalisant la détection d'accords pour une matrice d'observations chroma obs_m via les HMM. Renvoie une séquence d'accords (en numérique) à partir de la matrice d'observations chromas du morceau étudié "obs_m".
- *function prob_trans = f_cycle_des_quintes()* : fonction calculant les probabilités de transition entre accords d'après le cycle des quintes.
- *function path_v = viterbi(probInit_v, probObs_m, probTrans_m)* : Algorithme de Viterbi. Renvoie la séquence d'accords la plus probable à partir d'une matrice d'observations chromas. Modèle probabiliste plus précis que la comparaison directe.
- *spectrogramme = f_Q_transform, f_Q_transform2(v_sig, Fe, Q, note_min, note_max, freq_la_ref)* : Calcule le spectrogramme à Q constant. `f_Q_transform` fait la CQT classique alors que `f_Q_transform2` fait la "efficient CQT".
`v_sig` est le signal à analyser, `Fe` la fréquence d'échantillonnage, `Q` le facteur de qualité, `note_min` la note la plus basse de l'analyse, `note_max` la note la plus haute de l'analyse, `freq_la_ref` la note de référence (au cas où le morceau serait désaccordé).

Partie alignement

- *Alignement global, Needleman-Wunch = f_needleman, f_needleman2, f_needleman2_Interval(chaineA, chaineB, m_sim, m_cor, open_gap, ext_gap)* : différentes fonctions effectuant l'algorithme de Needleman-Wunch. Version naïve pour `f_needleman`, avec gap affine pour `f_needleman2` et avec gap affine en utilisant les intervalles pour `f_needleman2_Interval`.

ChaineA et ChaineB les deux suites d'accords à comparer, m_sim la matrice de pénalité pour les matchmismatch. Paramètre requis selon version : m_cor est simplement là pour faire la correspondance entre l'accord et sa position dans la matrice de similarité. open_gap, le coût de l'ouverture et ext_gap le coût d'extension.

- *Alignement local, Smith-Waterman* = *f_smith_waterman f_smith_waterman2 f_smith_waterman2_playlist(chaineA, chaineB, m_sim, m_cor, open_gap, ext_gap, nb_match)* : différentes fonctions effectuant l'algorithme de Smith-Waterman. La version naïve pour la fonction *f_smith_waterman*, avec affine gap pour *f_smith_waterman2*, avec les intervalles des accords pour *f_smith_waterman2_Interval* et avec un sous espace de recherche spécifique pour le programme, *f_smith_waterman2_playlist*. Les paramètres, les deux chaînes à comparer : *chaineA*, *chaineB*. *m_sim*, la matrice de pénalité. Selon les versions, *m_cor* la matrice de correspondance entre les accords et la matrice de similarité, *open_gap* et *ext_gap* les coûts d'ouverture et d'extension et *nb_match* le nombre de chemins voulu.
- *matrice de pénalité* = *f_creer_penalty_et_corres f_creer_penalty_et_corres_dist(c_chroma_ref)* : fonction calculant la matrice de pénalité, *f_creer_penalty_et_corres* avec des valeurs arbitraires fixées en dur dans le programme. *f_creer_penalty_et_corres_dist* elle utilise *c_chroma_ref* des chromas de référence pour calculer la distance entre chaque accords.
- *c_mor, dist* = *f_similarity_base(c_mor, c_chroma_ref)* : calcule la similarité entre tous les morceaux de la structure. *c_mor* est la stucture contenant tous les morceaux, *c_chroma_ref* correspond au chroma de référence pour les 24 accords (plus l'accord 'N' correspondant au silence). La fonction va d'abord appeler *f_creer_penalty_et_corres_dist* pour avoir les matrices puis pour chaque couple de morceaux va appeler les fonctions d'alignement.
- *creation_mix* script permettant de prendre la structure retournée par *analyse_Music*. Puis l'algorithme va calculer les chemins entre chaque morceau, puis va les afficher dans un espace deux dimensions pour faire la playlist. Malheureusement, le script retourne uniquement les fichiers et les temps de cross-fade, l'utilisateur doit donc les rentrer à la main dans la partie synthèse s'il veut avoir le mix final.

Partie synthèse

- *function [y_v, ts_factor] = f_crossfade (x1_v, x2_v, Fs1, t1_start, t1_end, t2)* : fonction effectuant la transition entre les vecteurs audio des deux morceaux "*x1_v*" et "*x2_v*". C'est aussi cette fonction qui effectue la détection du rythme qui mène au timestretch du second morceau.

- *function* $[y_v, tempo_v] = f_rhythm(x_v, Fs)$: fonction effectuant une analyse du rythme sur le signal " x_v " fourni. Elle fournit en sortie un vecteur " y_v " de même taille composé de Diracs synchronisés sur la noire, ainsi qu'un vecteur " $tempo_v$ " décrivant l'évolution du nombre de bpm dans le temps.
- *function* $y_v = f_timestretch(x_v, stretch)$: fonction effectuant l'étirement temporel du signal " x_v " à l'aide d'un vocodeur de phase.
- *function* $Xtilde_m = f_tfct(x_v, Nfft, L_n, R)$: fonction effectuant la transformée de Fourier à court terme du signal " x_v ", nécessaire à l'utilisation de " $f_timestretch$ ", afin d'effectuer la synthèse.
- *function* $x_v = f_itfct(Xtilde_m, L_n, R, w_v)$: fonction effectuant la transformée de Fourier à court terme inverse du signal " $Xtilde_m$ ", nécessaire à l'utilisation de " $f_timestretch$ ", une fois la synthèse terminée.