

Guide programmeur — générateur de spectres RMN synthétiques

basé sur le fonctionnement de spectrum_generator_random.R

1. Vue d'ensemble du projet

Ce script génère des spectres RMN 1D synthétiques (mélanges de multiplets Voigt), crée des labels (classification et régression) pour entraîner des modèles (notre CNN 1D en l'occurrence), et propose deux variantes : des spectres composés à partir d'une base de donnée peak forest (**BdDReference_CPMG**) ou bien des simulations de coupes TOCSY denses, puis génère ensuite des datasets mixtes composés de ces deux variantes.

Les composants principaux :

- définitions utilitaires (axe ppm, triangle de Pascal pour la création de multiplets, profil Voigt approché pour coller à la forme des pics réels),
- génération de multiplets (**simulate_multiplet**),
- génération d'un spectre aléatoire à partir d'une base de métabolites issue de la base de données Peak Forest (**generate_random_spectrum_with_peaks**),
- deux types de génération d'un dataset d'entraînement (la deuxième est la plus optimale) (**generate_spectrum_labels_full**, **generate_spectrum_labels_full_mixed**),
- fonction dédiée au type coupe TOCSY dense (**generate_dense_tocsy_1D_cut**),
- affichage interactif avec plotly (grid de spectres).

2. Variables globales importantes

ppm

```
ppm <- seq(14, 0, length.out = 2048)
```

- **But** : axe des abscisses en ppm standard ($14 \Rightarrow 0$), par défaut 2048 points.
- **Remarques** : utilisé par **simulate_multiplet()** et fonctions qui s'appuient sur la grille globale.

3. Fonctions utilitaires

3.1 pascal_triangle

- **But** : retourne les coefficients binomiaux pour construire les intensités relatives d'un multiplet classique (ex. doublet, triplet \Rightarrow distribution d'intensités).
- **Entrée** : **n (entier)** — ordre du multiplet ($0 \Rightarrow$ singulet, $1 \Rightarrow$ doublet, etc).
- **Sortie** : vecteur de $n+1$ coefficients (numeric), somme = 2^n .
- **Cas limites** : si $n < 0 \Rightarrow$ on peut renvoyer vecteur vide ou erreur. S'assurer que n est entier ≥ 0 .

3.2 voigt_profile

- **But** : Approximativement créer un profil mixte Lorentzien + Gaussien (approximation simple d'un Voigt). Normalise le pic.
- **Entrées** :
 - **x (numeric vector)** : axe (ppm),
 - **center (numeric)** : position centrale ppm,
 - **fwhm_g (numeric)** : full-width at half-maximum gaussien (en ppm),
 - **fwhm_I (numeric)** : FWHM lorentzien (en ppm).
- **Sortie** : vecteur numérique de même longueur que x, valeurs normalisées (max = 1).

4. Génération de multiplets

simulate_multiplet

- **But** : générer le signal d'UN multiplet (ensemble de sous-pics pour constituer un gros pic global) répartis autour d'une position **center** en ppm, avec intensités relatives données par le triangle de Pascal. Fait également la somme de plusieurs **voigt_profile**.

- **Entrées :**
 - **center (numeric)** : position en ppm du multiplet,
 - **n_couplings (int)** : nombre de *couplages* ⇒ correspond à l'ordre du multiplet (*n_couplings*=0 singlet, =1 doublet, etc.),
 - **J (numeric)** : constante de couplage en Hz (par défaut 7 Hz ici mais peut être modifié s'il faut),
 - **n_protons (numeric)** : nombre d'équivalents en protons ⇒ échelle d'intensité,
 - **fwhm_g, fwhm_l (numeric)** : FWHM gaussien et lorentzien en ppm.
- **Sortie** : vecteur signal de longueur **length(ppm)** correspondant au profil total du multiplet.

5. Accès à la base de données de métabolites

get_metabolite_peaks

```
get_metabolite_peaks <- function(metabolite_name) {
  df <- BdDReference_CPMG[[metabolite_name]]
  if (is.null(df) || nrow(df) == 0) return(NULL) # cas où le métabolite n'existe pas ou est vide

  peaks <- lapply(seq_len(nrow(df)), function(i) {
    list(
      ppm = df$ppm[i],
      nH = df$nH[i],
      couplings = as.numeric(df$couplage[i]) # si besoin, convertir en nombre ou laisser
      en caractère
    )
  })
  return(peaks)
}
```

- **But** : extraire la liste des signaux (ppm, nH, couplage) pour un métabolite particulier depuis **BdDReference_CPMG**.
- **Entrées** :

- **metabolite_name** (string)
- **Sortie :**
 - **peaks** : liste de listes, chaque élément contient ppm, nH, couplings.
 - Retourne NULL si métabolite absent ou vide.

6. Génération d'un spectre aléatoire à partir d'une BdD

generate_random_spectrum_with_peaks

But : composer un mélange aléatoire de **n_compounds** (nombre aléatoire entre 20 et 34, le choix de ces bornes peut bien évidemment être modifié) extraits de la base BdD, simuler les multiplets correspondants, additionner, ajouter du bruit, renvoyer spectre et table des pics réels.

Entrées :

- **BdD (list of data.frames)** : base de référence (par défaut **BdDReference_CPMG**, mais peut être substitué par C13 pour entraîner sur des extractions lignes/colonnes de spectres HSQC par exemple)
- **min_compounds, max_compounds (ints)** : nombre aléatoire de composés
- **noise_sd** : écart-type du bruit gaussien ajouté
- **seed** : graine pour reproductibilité.

Sortie : liste avec

- **spectrum** : tibble(ppm = ppm, intensity = spectrum_total) — vecteur normalisé (% max(abs(...))),
- **true_peaks** : data.frame listant pour chaque signal : compound_id, compound_type, ppm, nH, couplage, n_couplings, intensity.

Exemple d'appel :

```

res <- generate_random_spectrum_with_peaks(BdDReference_CPMG,
min_compounds=10, max_compounds=15,
noise_sd=0.005, seed=42)
plot(res$spectrum$ppm, res$spectrum$intensity, type="l")

```

7. Affichage multiple (plotly)

```

spectres <- lapply(1:4, function(i) {
  generate_random_spectrum_with_peaks(..., seed = 40 + i)
})
plots <- lapply(spectres, function(res) {
  p <- plot_ly(res$spectrum, x = ~ppm, y = ~intensity, type = "scatter", mode = "lines", ...)
  for (ppm_peak in res$true_peaks$ppm) {
    p <- p %>% add_segments(...)
  }
  p
})
subplot(plots, nrows = 2, ...)

```

- **But :** créer un panneau 2x2 de spectres interactifs avec segments verticaux aux positions réelles des pics.

8. Génération d'un dataset complet (labels)

generate_spectrum_labels_full

```

generate_spectrum_labels_full <- function(n_spectra = 1000,
n_points = 2048,
intensity_threshold = 0.2,
center_margin = 0.2,
seed = 123) { ... }

```

- **But :** générer **n_spectra** spectres et labels utiles pour l'entraînement d'un réseau :
 - **X :** matrice [**n_spectra**, **n_points**] des spectres interpolés,
 - **y_class :** matrice [**n_spectra**, **n_points**] labels (0 = background, 1 = pic centré+fort, 2 = pic non-centre/moins fort),

- **y_reg** : array [n_spectra, n_points, 3] avec (ppm_val, intensity, fwhh_est) pour positions de pics (sinon 0).
- **Entrées** :
 - **n_spectra (int)** : nombre de spectres à générer,
 - **n_points** : longueur de la grille de sortie,
 - **intensity_threshold** : seuil pour distinguer fort/faible (pas forcément super utile, possibilité de supprimer cette variable si vous le jugez nécessaire),
 - **center_margin** : fraction centrale de la fenêtre considérée comme "centré",
 - **seed** : graine.
- **Sortie** : liste X, y_class, y_reg.
- **Propriétés des labels** :
 - y_class a 0/1/2 selon logique centered && strong => 1, else 2 pour peaks, 0 pour background.
 - y_reg[idx,] stocke la position ppm, amplitude et estimation FWHH.

9. Simulation TOCSY dense — **generate_dense_tocsy_1D_cut**

```
generate_dense_tocsy_1D_cut <- function(n_points = 2048,
                                         n_multiplets = 100,
                                         max_multiplet_order = 6,
                                         line_broadening = 2.5,
                                         noise_level = 0.000001,
                                         base_freq = 4.7,
                                         ppm_range = c(0,14),
                                         coupling_range = c(4,14),
                                         voigt_mix = 0.5,
                                         seed = NULL,
                                         plot = TRUE) { ... }
```

- **But** : créer une coupe 1D « dense » ressemblant à une coupe TOCSY — beaucoup de multiplets aléatoires répartis sur la gamme ppm donnée.

- **Entrées** : paramètres pour nombre de multiplets, **ordre max**, **bruit**, **gamme ppm**, **plage couplages (Hz)**, **mélange gauss/lorentz (voigt_mix)**, etc.
- **Sortie** : liste spectrum (tibble ppm, intensity) et true_peaks (tibble ppm, amplitude).
- **Comportement** :
 - pour chaque multiplet : choisi center_ppm aléatoire, order (1..max), convertit J en ppm (/400), construit shifts et intensités
 - tire intensity, sigma, gamma de façon aléatoire, ajoute le profil via voigt_approx
 - ajoute du bruit (gaussien)

10. Dataset mixte —

generate_spectrum_labels_full_mixed

- **But** : mixer les deux générateurs (BdD-based random spectra & TOCSY dense) pour créer un dataset hétérogène.
- **Entrées** : identiques aux autres générateurs + mix_ratio probabilité d'utiliser la simulation TOCSY.
- **Sortie** : identique X, y_class, y_reg.
- **Comportement** : pour chaque spectre, choisit aléatoirement la méthode de génération selon mix_ratio, puis construit les labels comme dans generate_spectrum_labels_full.
- **Remarques pratiques** : permet d'entraîner un modèle robuste à différents styles de spectre.