TP 1 : Filtrage adapté pour la détection de signaux

L'objectif de ce TP est de mettre en œuvre la technique du filtrage adapté en utilisant des outils python (`numpy, scipy, pycbc`) pour détecter des signaux de faible significance. Cette mise en pratique vous permettra d'identifier les avantages et les limites de cette procédure de traitement. Elle vous familiarisera également avec des notions qui seront importantes par la suite notamment sur les propriétés du bruit de mesure.

1 Application à des données simulées

Dans un premier temps, vous allez traiter des données que vous simulerez afin de connaître parfaitement le signal attendu ainsi que les propriétés du bruit. Cette première étape est classique dans le développement de toutes les méthodes d'analyse de données. On veut s'assurer que notre procédure permet bien d'estimer les caractéristiques réelles du signal sans engendrer de biais.

1.1 Signal simple et bruit blanc gaussien

Le premier cas étudié sera celui de la détection de la raie d'émission H_α située à une longueur d'onde de $\lambda_0 = 656.3$ nm dans le référentiel du laboratoire. Supposons que l'on étudie un objet astrophysique de redshift $z = \frac{\lambda_{\text{obs}} - \lambda_0}{\lambda_0}$ inconnu précisément mais compris entre 0 et 1.

- En utilisant la librairie `numpy`, effectuez un tirage aléatoire d'un redshift selon une loi uniforme comprise entre 0 et 1 et déterminez la longueur d'onde observée de cet objet astrophysique.
- Utilisez la librairie `numpy` pour générer un tableau de 1000 valeurs de longueurs d'onde variant de 600 nm à 1400 nm. Quel est le pas séparant deux valeurs de longueurs d'onde dans ce tableau ?
- Explorer la librairie `scipy.stats` pour trouver une méthode permettant de calculer la fonction densité de probabilité (*Posterior Distribution Function (PDF)* en anglais) d'une loi normale centrée sur la longueur d'onde précédemment définie et de largeur égale à 1% de cette valeur centrale sur la plage de longueurs d'onde précédemment définie. Ce tableau de valeurs correspond à votre signal. Utilisez la librairie `matplotlib.pyplot` pour tracer ce signal.
- En utilisant la librairie `numpy`, générez un tableau de 1000 valeurs obtenues par tirage aléatoire gaussien centré sur 0 et de déviation standard 1. Ce tableau de valeurs correspond au bruit de mesure.
- Additionnez les deux tableaux précédemment définis pour générer votre vecteur de données simulées. Utilisez la librairie `matplotlib.pyplot` pour tracer ces données.
- Définissez un modèle de signal recherché en générant une loi normale de déviation standard 8 nm sur une plage de longueurs d'onde de même pas que la plage de mesure mais ne contenant que 60 valeurs.
- En utilisant la méthode de filtrage adapté vue en cours, définissez une fonction permettant de calculer la corrélation croisée entre votre vecteur de données et votre modèle de signal recherché.

- Connaissant la déviation standard du bruit, estimer le signal-sur-bruit (SNR) dans ce nouveau tableau filtré. Utilisez la librairie `matplotlib.pyplot` pour tracer les variations du SNR.

- En supposant qu'une détection est effectuée si SNR > 3, en déduire la position du signal. Calculez le redshift correspondant et comparez le avec le redshift réel que vous avez utilisé pour simuler les données.

- Effectuez quelques tests pour étudier l'impact de l'amplitude du signal simulé, de la déviation standard du modèle utilisé pour le filtrage adapté et de l'amplitude du bruit sur le résultat final.

1.2 Signal complexe et bruit gaussien coloré

Nous allons maintenant considérer le cas d'un signal étendu dans le temps avec une structure plus complexe qu'une simple loi normale. Par ailleurs, le bruit gaussien considéré ne sera plus blanc mais coloré.

- Utilisez la librairie `numpy` pour générer un tableau de 1000 valeurs variant de 0 s à 10 s.

- Définissez une fonction permettant de générer un signal sinusoïdal dont l'amplitude et la fréquence augmentent en fonction du temps sur une plage temporelle d'une seconde avec la même fréquence d'échantillonnage que le tableau précédent. Utilisez la librairie `matplotlib.pyplot` pour tracer ce signal. Choisissez des paramètres de cette fonction permettant d'observer $\sim$5 passages par un maximum local.

- Calculez le pas d'échantillonnage de votre signal en s.

- Connaissant le pas d'échantillonnage, utilisez la librairie `fft.fftfreq` de `numpy` pour générer les tableaux f_d et f_s de fréquences de Fourier correspondant à vos plages temporelles de 10 s et de 1 s. Tracez les valeurs de f_s et remarquez sa structure particulière.

- Calculez la transformée de Fourier de votre signal avec la fonction `fft.fft` de `numpy`. Tracez la partie réelle du résultat en fonction de la fréquence f_s. Notez sa structure particulière. Pourquoi la transformée de Fourier calculée par `fft.fft` a-t-elle cette structure ?

- En utilisant la première moitié des valeurs de f_d (fréquences positives), définissez une fonction permettant de générer la densité spectrale de puissance (PSD) d'un bruit coloré sous la forme d'une loi de puissance : $p = \left(\frac{f_d}{10\ \mathrm{Hz}}\right)^{\gamma} + 1$, avec γ un indice inférieur à -1. Il faudra exclure la première valeur de f_d car elle est égale à 0 et p ne sera donc pas définie. Utilisez la librairie `matplotlib.pyplot` pour tracer une PSD.

- Insérez la valeur 0 au début du tableau de votre PSD et concaténez le tableau miroir pour que la structure de cette PSD corresponde à la structure générée par `fft.fft`.

- Générez un tableau de 1000 valeurs obtenues par tirage aléatoire gaussien centré sur 0 et de déviation standard 1. Ce tableau correspond à du bruit blanc gaussien.

- Calculez la transformée de Fourier de ce tableau. Multipliez le résultat par la racine carrée de votre PSD. Utilisez la fonction adaptée de numpy pour effectuer la transformée de Fourier inverse du résultat. Votre réalisation de bruit gaussien coloré correspondra à la partie réelle du résultat final.

- Ajoutez votre signal à une position tirée aléatoirement dans cette réalisation de bruit coloré pour obtenir un vecteur de données simulées d. Tracez le résultat obtenu pour différentes amplitudes pour vous assurer que vous pouvez voir le signal à partir d'une certaine amplitude. Choisir une amplitude de signal qui ne vous permet pas de distinguer le signal à l'œil pour la suite de l'exercice.

- Utilisez la fonction que vous avez codée pour générer du bruit coloré afin d'estimer la matrice de covariance du bruit (utilisez la définition vue en cours). Inversez cette matrice en utilisant la fonction `numpy` adéquate. En déduire le tableau qui correspondra à votre filtre adapté f_g.

- Utilisez la fonction que vous avez développée dans la section 1.1 afin d'identifier votre signal dans les données d par la technique du filtrage adapté.

- Effectuez quelques tests pour étudier l'impact de la pente γ de la PSD de votre bruit sur le SNR de détection de votre signal.

1.3 Comparaison des outils développés avec des librairies dédiées

Avant d'appliquer la technique du filtrage adapté à un véritable ensemble de données, nous allons comparer les résultats obtenus dans les sections 1.1 et 1.2 avec ceux renvoyés par la fonction `correlate` de la librairie `scipy.signal`.

- Utilisez les données simulées ainsi que vos modèles de recherche des sections 1.1 et 1.2 afin d'appliquer la technique du filtrage adapté en utilisant la fonction `correlate` de la librairie `scipy.signal`. Comparez les résultats obtenus avec ceux issus de votre fonction sur un même graphique. Identifiez-vous des différences ?

- Utilisez la librairie `time` afin de mesurer le temps d'exécution de votre programme et comparez le au temps d'exécution de la fonction `correlate`. Pourquoi identifiez-vous de telles différences ?

- Utilisez la fonction `EmissionsTracker` de la librairie `codecarbon` afin d'estimer l'émission de CO_2 induite par l'usage de votre programme et comparer la à celle obtenue en utilisant la fonction `correlate`. Vous effectuerez une boucle de 1000 itérations d'application du filtrage adapté pour que le résultat de `codecarbon` soit non-nul. Comparer les résultats et notez le lien avec le temps d'exécution des programmes.

2 Application à des données LIGO

Maintenant que le filtrage adapté n'a plus de secrets pour vous, utilisons cette méthode de traitement pour identifier un signal d'onde gravitationnelle dans les données LIGO.

2.1 Accéder aux données LIGO

- Utilisez la fonction `Merger` de `pycbc.catalog` afin d'importer les données brutes contenant l'événement GW150914.

- Sélectionnez la série temporelle (strain) du détecteur Hanford (H1) et tracez cette série avec `matplotlib.pyplot`. L'axe temporel correspond à l'attribut `sample_times` de l'objet `strain` que vous avez défini.

- On remarque que les données brutes sont largement dominées par un bruit à basse fréquence (< 15 Hz). Les ondes gravitationnelles accessibles avec LIGO ont des fréquences comprises entre 50 Hz et 10 kHz. On peut donc simplement appliquer un filtre passe-haut pour supprimer ce bruit de basse fréquence dans les données brutes. Utilisez la fonction `highpass` de `pycbc.filter` pour filtrer les données brutes et stockez les données traitées dans un nouvel objet.

- Utilisez la méthode `psd` associée à ce nouvel objet afin de calculer la PSD du bruit dans les données brutes. Cette méthode utilise la procédure de Welch pour calculer la PSD du bruit. Cette procédure permet de réduire les incertitudes dans l'estimation de la PSD en moyennant les PSD obtenues dans différents intervalles temporels des données brutes. Utilisez des fenêtres temporelles de 4 s pour calculer la PSD du bruit avec cette méthode.

- En utilisant des fenêtres temporelles de 4 s dans la procédure de Welch on fixe la fréquence minimale à laquelle on peut estimer la PSD, *i.e.* $1/4 = 0.25$ Hz. Cette fréquence minimale correspond au pas séparant chaque point dans la PSD. Comme la série temporelle initiale a une durée plus importante, l'intervalle de fréquences théoriquement atteignable est plus faible. On va donc interpoler la PSD précédemment estimée pour adapter le pas de fréquence de la PSD a celui attendu pour nos données brutes. Pour ce faire, utilisez la fonction `interpolate` de la librairie `pycbc.psd`.

- Tracez la PSD obtenue et choisissez l'échelle de représentation adaptée.

2.2 Générer un modèle avec PyCBC

- Utilisez la fonction `get_td_waveform` de la librairie `pycbc.waveform` afin de générer un modèle de l'amplitude des polarisations "+" et "x" d'une onde gravitationnelle induite par une binaire constituée de deux objets compacts d'une masse respective de 50 $M_\odot$. On utilisera la méthode "SEOBNRv4_opt" comme approximation et une fréquence minimale de 20 Hz dans les arguments de cette fonction. Tracer l'amplitude associée à la polarisation "+" de l'onde gravitationnelle obtenue avec `matplotlib.pyplot`.

- Utilisez la méthode `resize` contenue dans l'objet précédent pour adapter la taille du modèle à celle des données brutes.

2.3 Significance de détection par filtrage adapté

- Utilisez la fonction `matched_filter` de la librairie `pycbc.filter` pour estimer le SNR de détection via la technique du filtrage adapté. Précisez bien les arguments *psd* et *low_frequency_cutoff* de cette fonction pour prendre en compte la PSD du bruit ainsi que la fréquence minimale du modèle considéré.

- On identifie toujours des artefacts au début et à la fin de la série temporelle traitée du fait du filtrage par la PSD dans l'espace de Fourier. Retirez les 8 premières et les 8 dernières secondes de la série temporelle du SNR en utilisant la méthode `crop` associée à cet objet. Tracer les variations du SNR. Quelle est la valeur maximale du SNR ?

- Appliquez la fonction `matched_filter` en utilisant d'autres valeurs de masse pour les 2 objets compacts afin d'identifier la masse maximisant le SNR. On supposera que les deux objets ont la même masse. Faites varier la masse de 10 $M_\odot$ à 50 $M_\odot$ par pas de 5 $M_\odot$.

- Explorez la littérature scientifique et trouvez quelle est la masse estimée par la collaboration LIGO pour les deux objets compact de cette binaire. Comparez ces résultats avec les vôtres.