Rapport de Stage :
Code de validation du DRS NeoNarval

Lucas Herbert

1er août 2018

1 Introduction

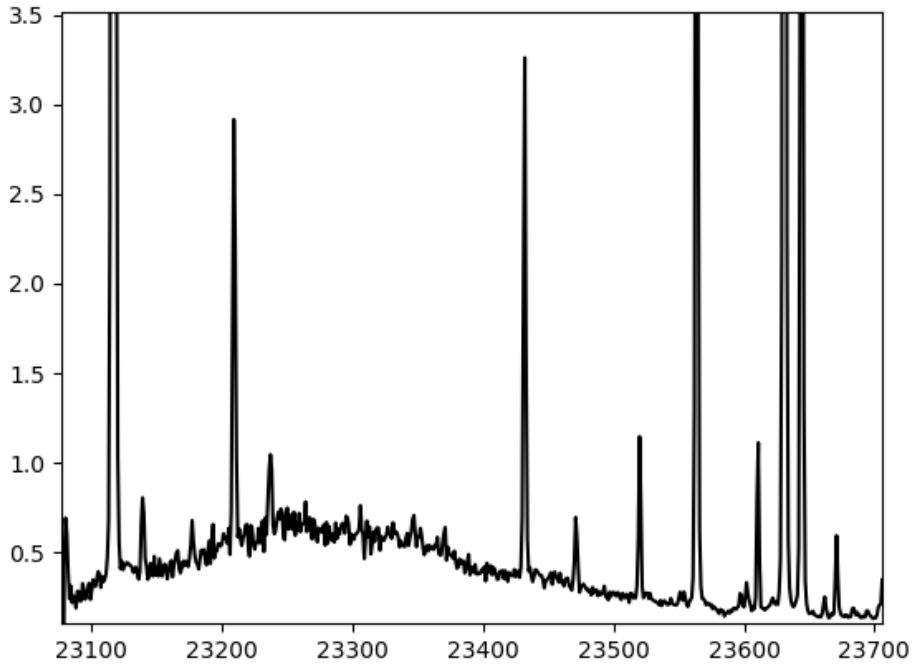
Le but de ce rapport est de présenter et expliquer le code de validation du DRS de NéoNarval. Ce code permet de confirmer ou non la bonne réduction des données par le code du DRS. Pour cela, on s'appuie sur le spectre bien connu du Thorium-Argon. On éclaire le CCD avec une lampe Thorium-Argon, on réduit les données obtenues et on compare le spectre réduit avec un atlas du Thorium-Argon. Le spectre réduit consiste en un fichier .fits dont la première extension contient deux listes : une liste de longueurs d'onde et une liste de valeurs d'intensités associée (un spectre, en somme). Le code doit comparer cela avec l'Atlas du Thorium-Argon qui lui se présente sous forme d'une liste de longueurs d'ondes correspondantes aux pics d'émission du Th-Ar. Nous devons donc pour un spectre réduit de Th-Ar retrouver tous les pics d'émission, calculer leur longueur d'onde précise, et lister ensuite ces longueurs d'onde pour les comparer à celles de l'Atlas. Cela nous permet de calculer la correspondance entre spectre réduit et atlas et ainsi valider ou non le DRS. Le code étant commenté plus précisément, il faut s'y référer pour plus de précisions sur son fonctionnement.

2 Description des différentes parties du code

2.1 Compute_offset.py

Le code nommé "compute_offset.py" est utilisé pour gérer le fond du spectre réduit que l'on veut valider. En effet, le but étant de détecter automatiquement les pics d'émission, les variations d'intensité du fond du spectre peuvent interférer avec l'information utile (à savoir les maxima locaux, etc). En principe, les spectres réduits sont déjà normalisés mais on renormalise quand même pour plus de facilité dans la suite su traitement. Le but pratique est ici de remettre le spectre à plat en supprimant son offset. Ce code implémente la fonction "normalize_offset" qui va prendre un spectre dont le fond n'est pas normalisé en entrée et normaliser le fond pour le rendre le plus régulier possible. Pour cela, on procède en interpolant entre les minima locaux du spectre, ce qui nous permet de remonter à un interpolation du fond que l'on peut ensuite retirer au spectre

FIGURE 1 – Spectre avant la normalisation



lui même. Le résultat est un spectre dont le fond est plat, avec un offset proche de zéro (voir figures 1 et 2). Il faut cependant noter que des problèmes peuvent apparaître aux bord du spectre.

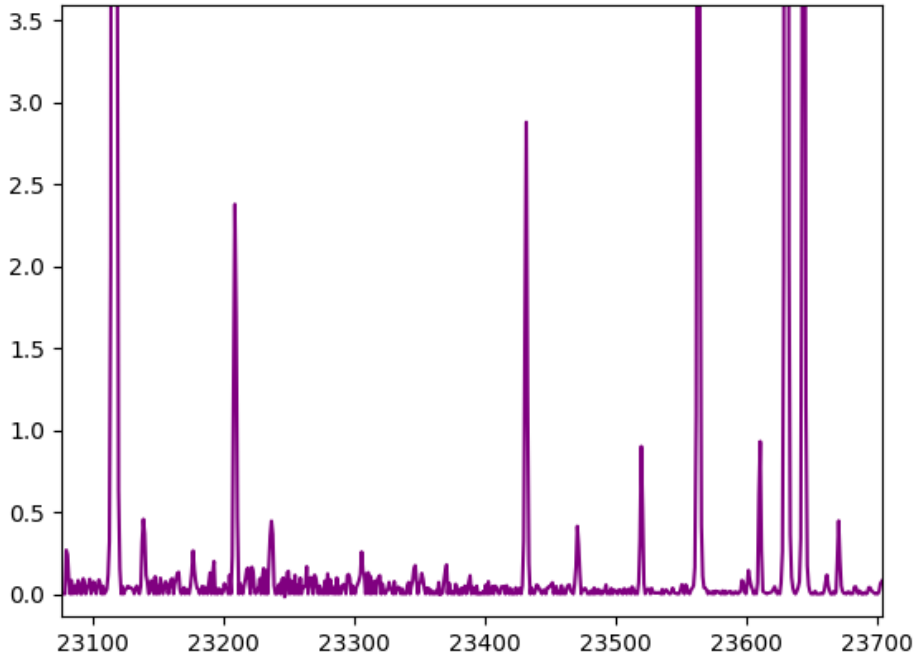
2.2 Compute_order.py

Ce code doit ouvrir le fichier .fits du spectre réduit et retrouver les différents ordres d'interférence au sein de ce spectre pour pouvoir les traiter un par un.

2.2.1 Search_orders

Pour cela, la fonction search_orders va rechercher dans le spectre global chaque ordre, retournant deux listes : la liste des longueurs d'onde de l'ordre sélectionné ainsi que la liste des intensités associées (Attention, les ordres sont ici numérotés dans l'odre croissant des longueurs d'ondes ! L'ordre 0 correspond donc à autour de 4000 Angstroms !).

FIGURE 2 – Spectre après la normalisation



2.2.2 Compute_order

La fonction compute_order va utiliser compute_offset pour normaliser l'ordre souhaité et retourner un spectre normalisé. Les entrées et sorties sont détaillées dans le code.

2.2.3 Plot_order

La fonction plot_order sert elle à tracer graphiquement l'odre souhaité en utilisant le module matplotlib.pyplot de Python.

2.3 Compute_spikes.py

Compute_spikes est un code permettant la détection des pics après la normalisation. Son but est, en partant d'un spectre normalisé, de retrouver les données utiles pour calculer la position précise de chaque pic dans la liste des intensités d'un ordre. Cette position sera repérée grâce à un indice précis et une longueur d'onde associée précise. Pour cela, on commence par retrouver tous les maxima du spectre entré en input. On les détecte mathématiquement (croissance puis décroissance locale des intensités du spectre) et on ajoute un filtre pour ne détecter que les pics suffisament hauts par rapport au fond du spectre (pour éviter de considérer que le bruit puisse générer des pics dans notre liste de pics). Ce critère de sélection des pics est un minimum de détection des maxima locaux (donc des pics) en dessous duquel on ne les considère pas comme des maxima (on ne les ajoute pas à notre liste de maxima) qui est dans le code un paramètre de la fonction appelé max_detection. Il reviendra souvent dans les paramètres des fonctions car on peut jouer sur ce paramètre pour discriminer une partie de l'information des spectres selon qu'on la trouve utile ou non.

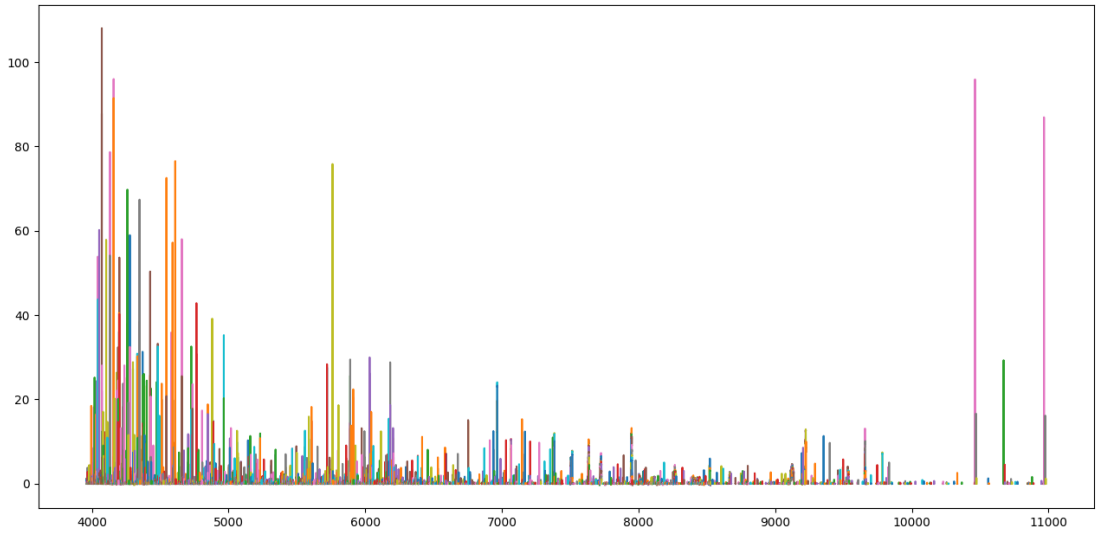
2.3.1 Find_spikes_data

C'est la fonction find_spikes_data qui va effectuer la liste des pics. Pour cela, elle prend le spectre normalisé en entrée et renvoie la liste des données associées à chaque pic : un pic est une liste d'intensités croissantes puis décroissant, on renvoie ainsi pour chaque pic les longueur d'ondes et indices sur lesquels il s'étend (voir la figure 3 pour une illustration). Une fois les données de chaque pic récupérées, on va pouvoir déterminer la position précise du centre de chaque pic par fit gaussien de ce pic (le centre correspondant à celui de la gaussienne), ainsi que sa largeur, etc.

2.3.2 Fit_spikes_order

C'est la fonction fit_spikes_order qui va fitter sur chaque pic trouvé par find_spikes_data une gaussienne dont l'espérance sera le centre du pic et l'écart-type sa largeur. Pour cela, elle va utiliser la fonction gaussian_fit qui est détaillée en dessous. Si chaque spectre peut être repéré par la même liste d'indices, de [0] à [longeur du spectre - 1], les longueur d'ondes calibrées associées au spectre peuvent changer (ce qui changera la valeur

FIGURE 3 – Détection des pics du spectre par find_spikes_data : chaque couleur représente les données d'un pic, elles seront utilisées ensuite pour déterminer la position précise de chaque pic par fit gaussien (axe X : λ en Angstroms, axe Y : intensité). Remarquez qu'il y a du vide entre les pics car on ne trace ici que les pics.



de chaque longeur d'onde centrale pour chaque pic). Pour gérer ce cas particulier, il y a un passage de la fonction qui est déjà clairement expliqué et détaillé dans le code. Cette fonction renvoie donc pour chaque pic de l'ordre considéré les données de son fit gaussien (centre en longueur d'ondes, centre en indices, largeur en longueur d'onde et en indices, etc).

Après utilisation de ce code, nous avons établi la liste des pics du spectre réduit donné en entrée, et nous pouvons donc tout comparer avec l'Atlas du Th-Ar.

2.4 Gaussian_fit.py

Cette routine a pour but de trouver les paramètres de la gaussienne qui colle le mieux à un série de données. Elle prend en entrée deux listes : une liste de valeurs (axe Y) et une liste d'abscisses associées (axe X). Elle trouve la gaussienne qui est la plus proche de ces données et en retourne les paramètres (amplitude, espérance, largeur) ainsi que les données résumant la qualité du fit. Pour cela on utilise le module lmfit de Python et les données du fit correspondent au rapport renvoyé par lmfit lors d'un fit. Il s'agit de l'erreur associée au fit. On utilise cette fonction sur chaque pic pour lui fitter une gaussienne et déterminer ainsi les paramètres précis de chaque pic (son vrai centre, sa largeur, etc). La figure 4 illustre le fit gaussien d'un pic.

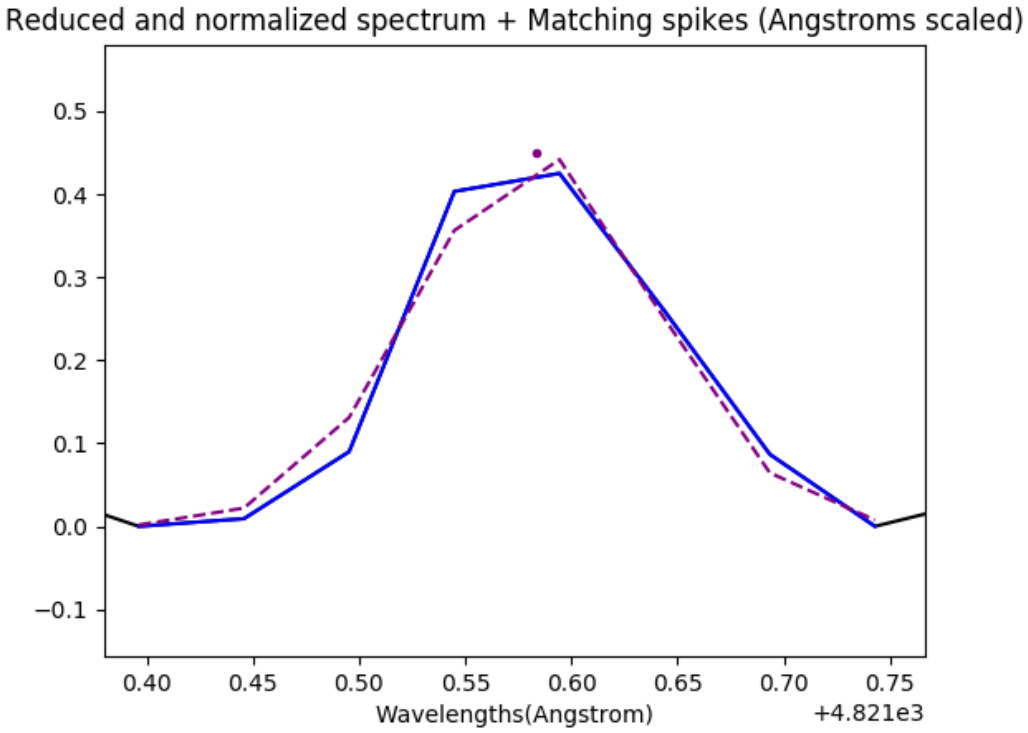
2.5 Matching.py

Le matching est la comparaison entre la liste des pics extraite du spectre réduit par les fonction décrites ci-dessus et celle de l'atlas qui constitue notre référence pour le Thorium-Argon. Pour chaque pic extrait du spectre réduit, il s'agit de retrouver le pic le plus proche dans l'atlas et, s'il est suffisament proche, de le considérer comme étant le même pic. On peut ensuite comparer pour chaque paire de pic matchés (un du spectre réduit et son équivalent de l'atlas) : la distance entre les deux correspondra à notre précision de localisation de ce pic. On peut ainsi remonter à la précision de l'instrument en montrant pour chaque pic détecté l'écart entre l'instrument et la référence Thorium-Argon. On notera l'importance de deux paramètres présent dans quasiment toutes les inputs : precision, qui correspond à la distance maximale séparant deux pics pour les considérer comme matchés, et max_detection qui correspond au minimum d'intensité pour considérer qu'un pic est un pic et qu'il faut le détecter (en dessous on fera comme si c'était du bruit).

2.5.1 Lambda_matching

La fonction lambda_matching va comparer deux listes et établir pour chaque valeur de la première un matching avec la plus proche valeur dans la deuxième. Dans notre cas elle donnera les matchings entre les pics détectés sur le spectre réduit et ceux de l'atlas Th-Ar, et les statistiques (pourcentage de pics matchant avec telle précision en Angstroms). L'idée est tout d'abord, pour chaque valeur de la première liste, de trouver la valeur la plus proche de celle-ci dans la seconde, puis dans un deuxième temps de

FIGURE 4 – Exemple de fit gaussien : en noir, le spectre original, en bleu le pic détécté par find_spikes_data, et en violet pointillé la gaussienne fittée par gaussian_fit, avec le point violet correspondant au centre du pic (ses coordonnées étant le centre calculé en abscisse et l'amplitude de la gaussienne fittée en ordonnée)



comparer leur écart à un critère fixé représentant la "limite de matching". Ce critère est fixé par choix dans l'algorithme en paramètre (voir code) et il dit que si la différence entre les deux valeurs ainsi déterminées est inférieure à une valeur fixée, alors ces deux valeurs sont considérées comme matchées. Cela implique pour nous, où les valeurs sont les positions des pics, que deux pics correspondent au même pic en réalité (le décalage étant dû aux imprécisions de l'appareil et du code de réduction, etc). Pour le détail, voir le code qui est commenté explicitement.

2.5.2 Order_matching

La deuxième fonction, order_matching, va utiliser lambda_matching pour établir le matching sur cet ordre. Elle va au fur et à mesure afficher les statistiques du matching et trier les pics à matcher selon leur largeur et la qualité de leur fit (les mauvais fits sont exclus, le critère utilisé est le chi-square donné dans le rapport de lmfit). Elle va calculer l'équivalent du décalage entre un pic et son pic matché en vitesse radiale (conversion des Angstroms vers le m/s). C'est en effet plus intéressant de connaître les shifts entre deux pics matchés en m/s car cela ne dépend par de la longueur d'onde alors que le décalage en Angstroms évolue selon la longueur d'onde considérée. Pour se faire, elle utilisera quelques fonctions comme read_chi_square (utilisée pour déterminer la qualité du fit en lisant le rapport de lmfit), etc, tout cela étant très détaillé dans les commentaires du code. Les résultats sont ensuite affichés graphiquement grâce à Matplotlib.pyplot (figure 5). Quelques petites précisions (encore une fois tout est dans le code) : si la distance entre deux longueurs d'ondes (une du spectre réduit et l'autre de l'atlas) parmis celles matchées est suffisament petite (critère encore plus précis que le simple critère de matching), on va considérer que le pic en question est tellement bien repéré par le DRS qu'il n'y a aucun doute que le matching est légitime, on va donc réduire cette erreur à zéro en remplaçant la longueur d'onde DRS par celle de l'atlas. Le but est de corriger localement un écart qui pouvait déjà être considéré comme négligeable. Cela aura une importance toute particulière dans les codes de calibration_methods décris dans le rapport sur la calibration.

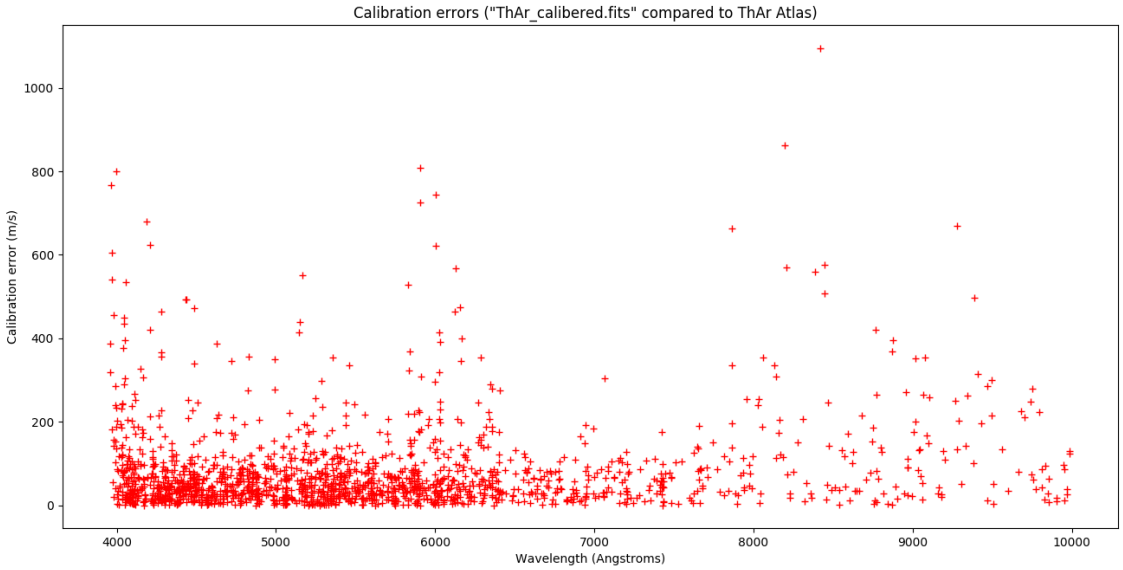
2.5.3 Order_matching_pickle

Cette fonction est utile au fonctionnement des code de calibration_methods. Elle enregistre les résultats du matching pour l'ordre considéré. Pour cela elle prend les sorties de la fonction order_matching et les met dans un dictionnaire qu'elle enregistre ensuite au format pickle. Les commentaires du code détaillent bien son fonctionnement donc nous n'allons pas plus loin sur cette fonction.

2.5.4 Autres fonctions

Enfin, des fonctions commes order_matching_pickle vont enregistrer les données du matching pour chaque ordre pour pouvoir retenir pour chaque pic la longueur d'onde

FIGURE 5 – Distribution des erreurs de calibration en m/s, en utilisant l'algorithme de matching (erreur du fichier .fits Th_calibered réduit avec le DRS comparé à l'atlas ThAr)



théorique qu'il devrait avoir (à savoir celle de son pic matché dans l'atlas) et ainsi recalibrer les longeur d'ondes pour la prochaine réduction. (la calibration sera détaillée dans un second rapport mais on l'évoque ici car la validation permet aussi de quantifier les erreurs de la réduction : en utilisant les résultats ainsi calculés, on peut améliorer la réduction en agissant sur la calibration en longueurs d'ondes).

Il restes quelques fonctions dans ce code qui sont détaillées en commentaire du code et qui permettent de tracer la distribution des erreurs de calibration ou de la résolution spectrale en fonction de l'ordre, de la longueur d'onde, etc. Les figures 6 et 7 illustrent leurs résultats.

3 Fonctionnement pratique des codes de validation

Maintenant que toutes les routines de validation du DRS ont été présentées, vous pouvez retrouver le détail de leur fonctionnement et des explications sur la reflexion qui nous a amené à les écrire comme cela dans les commentaires du code. Mais il reste un dernier point à aborder : comment faire tourner les codes ? Il faut ouvrir Pyzo dans le bon dossier, à savoir le dossier Codes_Lucas_Herbert où se situent les dossiers validation_methods, calibration_methods et extraction_methods. Chacun de ces dossiers contient les codes propres à sa mission ainsi que des fichiers utiles au fonctionnement de ses propres codes. Pour les codes de validation, leurs fichiers utiles se retrouvent dans le

FIGURE 6 – Distribution de la résolution spetrale ($\lambda/\Delta\lambda$) du spectromètre à échelle, la barre noire est à 65000 (résolution attendue)

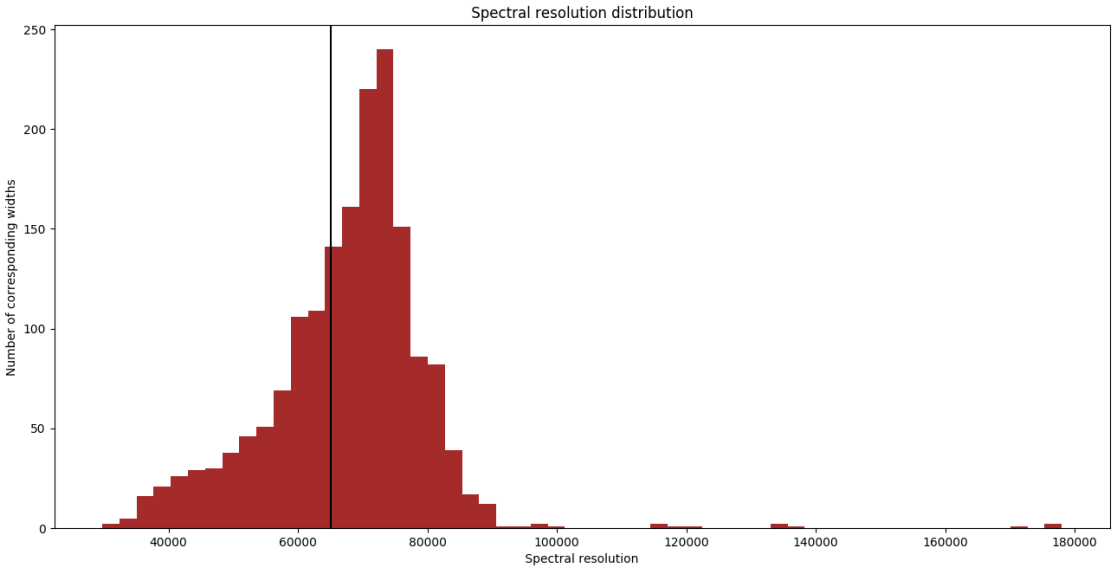
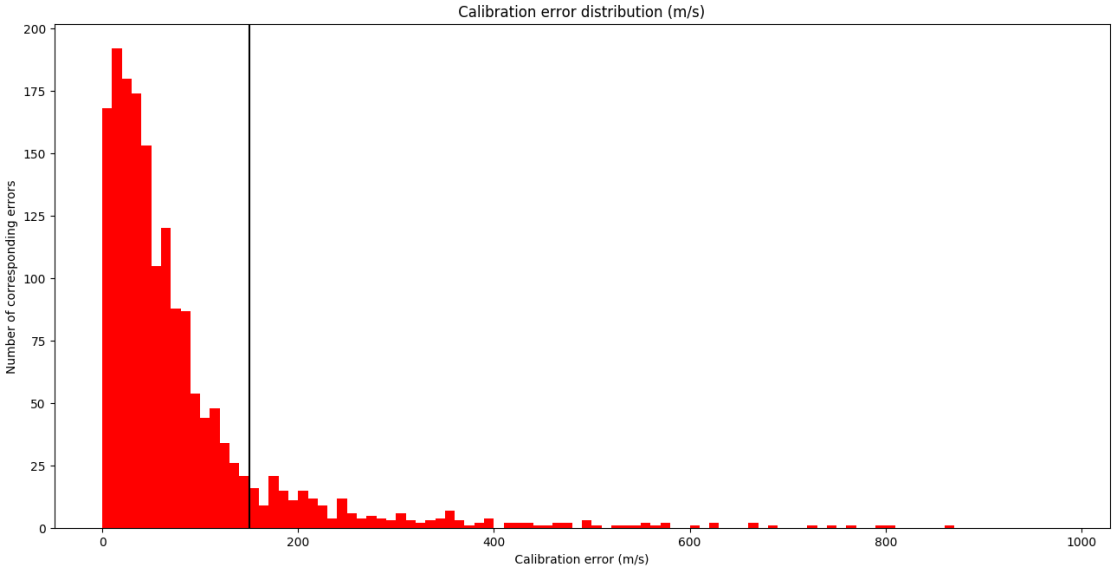


FIGURE 7 – Distribution statistique des erreurs de calibration en m/s, la barre noire est à 150 m/s (performance visée pour Narval)



dossier Validation_files qui se situe dans validation_methods. Dans le code de matching, on enregistre aussi des fichiers dans les dossiers de calibration mais il faudra se référer aux commentaires des codes ainsi qu'au rapport sur la calibration pour comprendre l'intérêts de ces fichiers.