Pointé automatique du temps d'arrivé des ondes P et S

Rapport terminal de projet informatique
1A EOST

Bastien PITOISET et Valentin CASSAYRE

5 mai 2023

Table des matières

1 Introduction

La libération d'énergie lors d'un séisme génère des ondes qui vont se propager différemment selon leurs caractéristiques. Les ondes de surfaces transmettent la plus grande partie de l'énergie relâchée, mais ne se propagent qu'à la surface de la Terre. Les ondes de volume, plus rapides, se propagent à l'intérieur de la Terre. Des ondes de volume, nous pouvons distinguer les ondes de compression, les ondes P, des ondes de cisaillement, les ondes S. Les ondes P sont les ondes les plus rapides avec une vitesse de propagation de 6km/s près de la surface de la Terre.

L'ensemble de ces champs d'ondes peuvent être mesurées au niveau de capteurs sismiques, par exemple dans une station de surveillance sismique. Ces stations enregistrent en continu les déplacement du sol et ces mesurent permettent de détecter d'éventuels séismes à la suite d'analyses. Et la détermination du temps d'arrivée des différentes ondes est une étape essentiel dans l'analyse de ces signaux. Le temps d'arrivée d'une onde correspond au début de sa phase, période pendant laquelle l'onde est enregistrée. Cette identification est très importante dans divers domaines, il permet par exemple de localiser le foyer ou de récolter des données sur la structure intérieure de la Terre.

L'objet de ce projet est donc de réaliser un programme permettant de détecter un séisme puis de pointer automatiquement le début des phases P et S.

Il existe trois grandes familles d'algorithmes de détection et de pointage automatique [d'après Cuenot (2003)] :

— Les algorithmes de détection par calcul de l'énergie comparent chaque valeur avec la moyenne des valeurs qui la précèdent.
— Les algorithmes basés sur des méthodes autorégressives recherchent les modélisations du bruit et du signal sismique.
— Les algorithmes utilisant des réseaux de neurones artificiels, après une phase d'apprentissage, sont capables de détecter les différentes ondes sismiques.

Pour ce projet nous avons choisi de travailler avec les algorithmes de détection par calcul de l'énergie. Ce sont historiquement ces algorithmes qui ont été utilisés en premier dans les années 80. Ils présentent l'avantage d'être les plus simples et de pouvoir à la fois détecter un potentiel séisme et de pointer le début des phases sismiques.

Ces algorithmes sont basée sur un signal d'entrée qui peut-être corrigé ou modifié et calculent une fonction caractéristique de ce signal. Cette fonction qui caractérise le signal peut ensuite être utilisée pour détecter un potentiel séisme et pour pointer ses différentes phases.

2 Acquisition des données

Nous avons choisi de travailler essentiellement avec le langage Python, un langage haut niveau qui présente de nombreux modules notamment dans le domaine de la sismologie. Nous nous sommes basés sur des enregistrements de séismes locaux autour de Strasbourg par des stations à proximité. Les données sont celles du RéSiF et ont été chargées avec le module Obspy. Un traitement de base a été effectué au préalable consistant en un filtre passe bande entre les fréquences de 2 Hzet 10 Hz. Le signal obtenu est ensuite stocké dans deux arrays Numpy, le temps dans *times* et l'amplitude correspondante dans *data* et le module Obspy n'est plus utilisé dans la suite, l'intérêt étant de présenter ces algorithmes.

3 Fonction caractéristique

La famille d'algorithme étudiée ici consiste en une analyse de l'énergie des champs d'onde incidents et de son évolution au cours du temps. Allen (1978) a introduit le concept de fonction caractéristique, qui caractérise un signal. Elle est obtenue par une ou plusieurs transformations non linéaires du sismogramme et doit augmenter brusquement au moment de l'arrivée d'un champ d'onde sismique.

Elle se base sur un signal qui peut-être modifié ou corrigé. Dans notre cas, nous avons déjà corrigé le signal notamment en le filtrant. La modification du signal s'effectue par des transformations non linéaires. En général les signaux modifiés sont positifs et sont donc appelés enveloppe par abus de langage.

3.1 Modification du signal

3.1.1 Transformées simples

W. Vanderkulk (1965) introduit l'utilisation de la valeur absolue du signal comme enveloppe. Cette approximation permettait de faire des économies de calcul significatives par rapport au carré des valeurs des amplitudes. Mais l'augmentation de la puissance de calculs a permis de retirer cette barrière. Allen (1978) généralise l'utilisation du carré des valeurs des amplitudes pour calculer la fonction caractéristique. Ces deux transformations correspondent aux transformations non linéaires les plus simples permettant d'aboutir à un signal positif, mais ne prennent pas en compte les valeurs voisines et donc les variations.

3.1.2 Enveloppe de Stewart

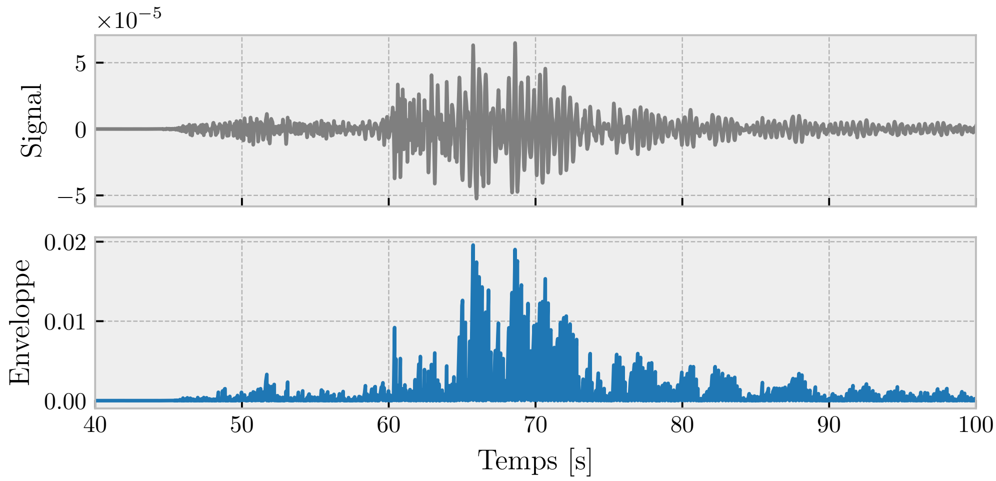

FIGURE 1 – Comparaison de l'enveloppe de Stewart (bleu) au signal d'entrée (gris).

Stewart (1977) a utilisé une enveloppe modifiée mdx basée sur la dérivée des données mettant en évidence les changements de pente. La valeur de mdx est calculée à partir d'une estimation de la dérivée dx en chaque point [d'après Withers (1982)] :

$$dx_i = x_i - x_{i-1} \tag{1}$$

Si le signe de dx a été constant pour moins de 8 valeurs consécutives, alors

$$mdx_i = mdx_i + dx_i \tag{2}$$

Sinon,

$$mdx_i = dx_i \tag{3}$$

Cette transformation permet de mettre en avant les variations . Il agit en quelque sorte comme un filtre passe haut, et est particulièrement utile pour des signaux bruts, donc l'intérêt est limité étant donné que notre signal a déjà été filtré.

3.1.3 Enveloppe supérieure et approximation géométrique

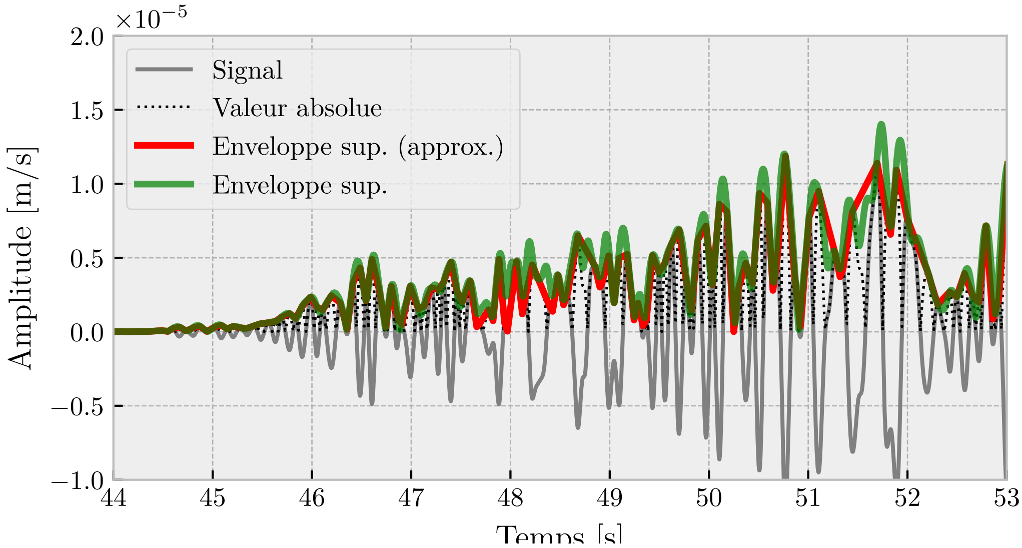

FIGURE 2 – Comparaison de l'enveloppe du signal (gris) obtenue à partir d'une transformée de Hilbert (vert) ou d'une méthode géométrique (rouge). La valeur absolue (noir pointillé) permet de distinguer la nuance avec l'enveloppe.

D'autres auteurs utilisent l'enveloppe classique appelée enveloppe supérieure, une courbe lisse qui décrit les amplitudes extrêmes du signal. Elle est définie par rapport à la transformation de Hilbert. Mais elle peut être approximée par des méthodes géométriques, par exemple en gardant la valeur absolue d'extremums locaux du signal.

En ne gardant que les points correspondant aux extremums locaux du signal initial et sans faire d'interpolation, nous réduisons grandement le nombre de points. Une autre conséquence est que les points ne sont plus espacés linéairement ce qui rajoute de la complexité et une source d'erreur pour la suite des algorithmes.

3.1.4 Enveloppe de Allen

Allen (1982) définit une nouvelle enveloppe [d'après Küperkoch (2010)]

$$E_i^2 = x_i^2 + C_i \times (x_i^2 - x_{i-1}^2) \tag{4}$$

avec C_i un coefficient tel que

$$C_i = \frac{\sum_{j=1}^{i} |x_j|}{\sum_{j=1}^{i} |x_j - x_{j-1}|} \tag{5}$$

Cette enveloppe approxime l'enveloppe supérieure sans pour autant réduire le nombre de points ni même d'avoir à réaliser une transformée de Hilbert (figure 3).

3.1.5 Enveloppe de Baer et Kradolfer

M. Baer (1987) proposent une amélioration de l'enveloppe d'Allen définit ainsi (figure 3) [d'après Küperkoch (2010)]

$$E_i^2 = x_i^2 + C_i \times (\dot{x_i}^2) \tag{6}$$

avec C_i un coefficient tel que

$$C_i = \frac{\sum_{j=1}^{i} x_j^2}{\sum_{j=1}^{i} \dot{x_j}^2} \tag{7}$$

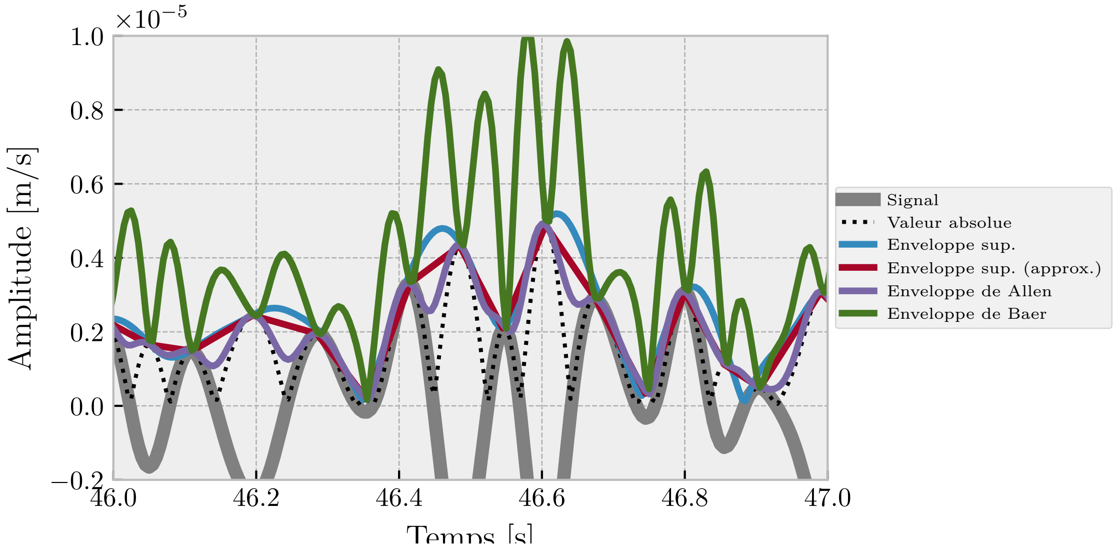

FIGURE 3 – Comparaison de différentes enveloppes calculées à partir du même signal (gris). La valeur absolue (noir pointillé), l'enveloppe supérieure (bleu), l'approximation géométrique de l'enveloppe supérieure (rouge), l'enveloppe de Allen (violet) et de l'enveloppe de Baer et Kradolfer (vert).

3.2 Calcul de fonctions caractéristiques

3.2.1 Méthode STA/LTA

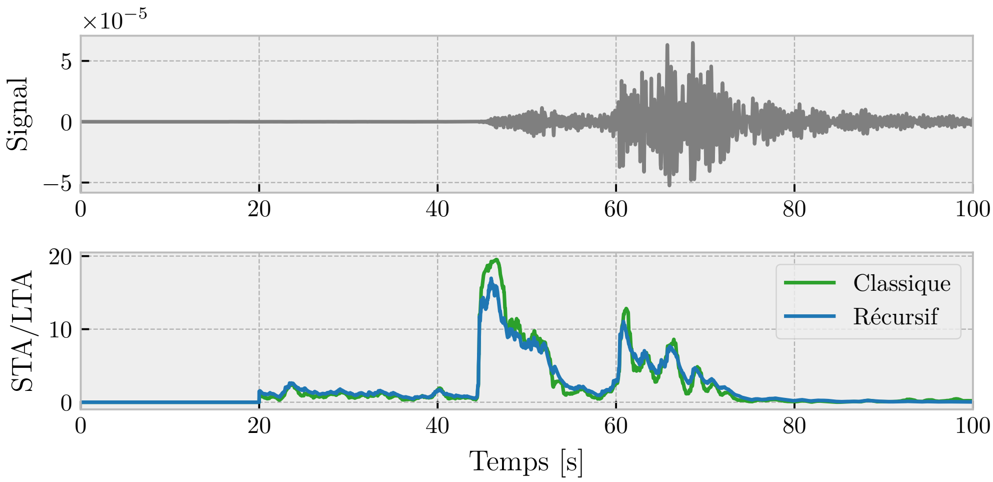

FIGURE 4 – Comparaison de la fonction caractéristique STA/LTA obtenue par méthode classique (vert) et récursive (bleu) en utilisant des largeurs de fenêtres de 2 et 10s.

Une grande partie des algorithmes de calcul de fonction caractéristique se basent sur la méthode appelée STA/LTA. Cette méthode consiste à comparer la moyenne à court terme (STA = Short Term Average) avec la moyenne à long terme (LTA = Long Term Average). À court terme, nous mesurons en quelque sorte l'amplitude instantanée du signal. À long terme, nous mesurons plutôt l'amplitude locale du bruit de fond. Le rapport entre les deux permet de comparer ces deux grandeurs. Nous constatons que la fonction obtenue caractérise bien le signal puisqu'on observe 2 pics correspondant au début de la phase P et de la phase S, figure 4. Cet algorithme se base donc sur deux paramètres qui sont la longueur des fenêtres STA et LTA, respectivement *nsta* et *nlta*. Dans la littérature nous trouvons généralement que la longueur de la fenêtre de LTA est 20 à 100 fois plus grande que la longueur de la fenêtre STA [d'après Küperkoch (2010), M. Vassallo (2012), Zhu. Weiqiang (2018)]. Le principal défaut de cet algorithme c'est qu'il est coûteux en calcul en raison des nombreuses moyennes mobiles. Ce coût se fait particulièrement ressentir en utilisant un langage de haut niveau, ici Python. Si nous considérons un signal de n points, auquel nous voulons calculer des moyennes glissantes de largeur n_f, il faudrait réaliser $n - n_f - 1$ somme de n_f éléments soit un total de $(n - n_f - 1) \times n_f$ opérations d'addition puis $n - n_f - 1$ opérations de divisions pour calculer les moyennes glissantes en chaque point. Une solution qui permet de limiter ce nombre consiste à calculer les moyennes mobiles à partir de la somme cumulée des valeurs des amplitudes du signal. En effet en utilisant une somme cumulée, nous devons réaliser $n - 1$ opérations d'addition pour calculer la somme cumulée puis seulement $n - n_f - 1$ opérations d'addition pour calculer les sommes spécifiques à chaque point, pour enfin calculer la moyenne en divisant $n - n_f - 1$ fois. La méthode classique utilise donc pour un grand nombre d'opérations environ $n \times n_f$ opérations d'addition contre seulement $2 \times n_f$ opérations d'additions en utilisant les sommes cumulées. Nous avons $2 \times n_f > n \times n_f$ pour $n_f > 2$, qui est toujours atteint. L'implémentation du calcul des moyennes glissantes d'un signal *data* à partir d'une somme cummulée peut se faire efficacement en utilisant le module Numpy :

```
import numpy as np

cumsum_data = np.cumsum(data)

moyennes[nf:] = (cumsum_data[nf] - cumsum_data[:-nf])/nf
```

3.2.2 Algorithme d'Allen

Allen (1982) propose un nouvel algorithme qui approxime l'algorithme STA/LTA trop coûteux en calculs, en calculant les coefficients STA et LTA par la relation de récurrence [d'après Cuenot (2003), Khalaf (2016), Gaol (2021)] :

$$STA_i = STA_{i-1} + \frac{1}{nsta}(E_i^2 - STA_{i-1}) \tag{8}$$

$$LTA_i = LTA_{i-1} + \frac{1}{nlta}(E_i^2 - LTA_{i-1}) \tag{9}$$

avec E_i^2 le signal d'entrée, *nsta* et *nlta* les largeurs des intervalles STA et LTA.

Cette approximation astucieuse permet de réduire le nombre de calculs à également $2 \times n_f$ opérations d'addition pour chaque moyenne mobile. Même si elle a comme limite qu'elle caractérise moins le signal d'entrée, le pic est légèrement moins marqué lors de l'arrivée du champ d'ondes et elle bruite la fonction caractéristique, il y a plusieurs pics au niveau du maximum global, figure 4.

Nous avons jusqu'à présent traduit ces algorithmes en Python, un langage haut niveau, pour des raisons d'aisance et de simplicité. Mais pour augmenter d'avantage l'efficacité de calcul de notre programme, nous pouvons le traduire dans un langage plus bas niveau. Nous avons choisi le langage C, qui présente une bonne compatibilité avec Python.

```c
#include <stdlib.h>

float *stalta_recursiv(float *data, int nsta, int nlta, int N)
{
    float *fc = malloc((sizeof(float) * N));
    float sta, lta;
    sta = 0;
    lta = 0;

    int i;
    for (i = 0; i < N; i++)
    {
        sta = (*(data + i) - sta) / nsta + sta;
        lta = (*(data + i) - lta) / nlta + lta;
        if (i > nlta && lta != 0)
            *(fc + i) = sta / lta;
        else
            *(fc + i) = 0;
    }

    return fc;
}
```

Pour réaliser la passerelle entre Python et C, nous utilisons le module *ctypes*. Le code C doit d'abord être compilé

```
gcc -c -Wall -Werror -fPIC `python-config --cflags` stalta.c
gcc -shared -o stalta.so stalta.o `python-config --ldflags`
```

Puis il peut être chargé en Python

```python
import ctypes

stalta_c = ctypes.CDLL('./stalta.so')

stalta_c.stalta_recursiv.argtype = (ctypes.POINTER(ctypes.c_float * n),
    ctypes.c_int, ctypes.c_int, ctypes.c_int)
stalta_c.stalta_recursiv.restype = ctypes.POINTER(ctypes.c_float * n)

fc_allen_recursiv_ptr = stalta_c.stalta_recursiv((ctypes.c_float * n)(*sqa_data),
    nsta, nlta, n) # exécution de la fonction
fc_allen_recursiv = fc_allen_recursiv_ptr.contents
```

En comparant les temps d'exécution d'un signal de 36000 points avec des fenêtres de 200 et 4000 points, nous obtenons 0.07s en Python, poussé à 0.06s en utilisant une array Numpy, contre 0.01s en C. Pour donner un ordre de comparaison, l'algorithme STA/LTA classique écrit entièrement en Python a un temps d'exécution de plusieurs secondes (sans l'utilisation de Numpy). Nous avons ainsi résolu les problèmes de la puissance de calcul engendré par cet algorithme.

3.2.3 Z-détecteur et algorithme de Baer et Kradolfer

Swindell et Snell (1977) proposent le Z-détecteur, une méthode de calcul basé sur une unique fenêtre contrairement à la méthode STA/LTA. Cette fonction caractéristique estime l'écart des données sismiques à la valeur-moyenne, exprimée en unité de son écart-type [d'après Withers (1982), Cuenot (2003), Küperkoch (2010)]

$$Z_i = \frac{STA_i - \mu}{\sigma} \tag{10}$$

avec STA_i la moyenne glissante de la fenêtre, μ la moyenne et σ l'écart-type des moyennes des fenêtres STA.

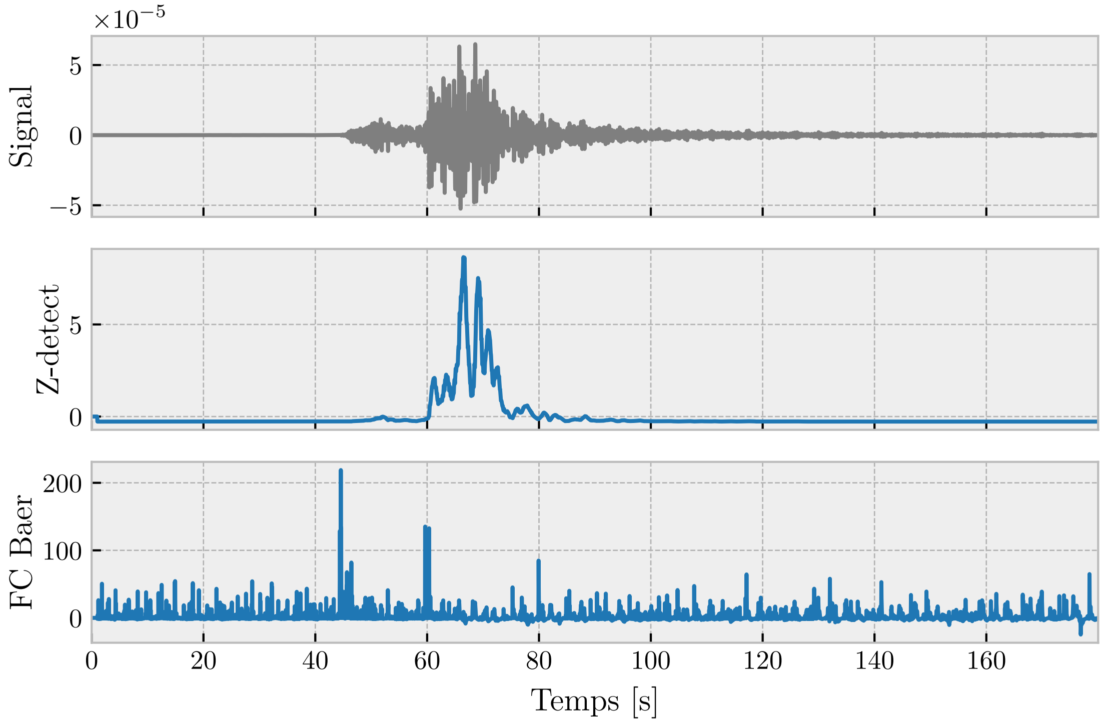

Figure 5 – Comparaison de la fonction caractéristique de Swindell et Snell avec celle de Baer et Kradolfer

M. Baer (1987) introduisent une variante

$$FC_i = \frac{E_i^4 - \bar{E_i^4}}{\sigma(E_i^4)} \tag{11}$$

avec $\bar{E_i^4}$ la moyenne de E^4 sur l'intervalle de la fenêtre choisie et $\sigma(E_i^4)$ l'écart-type de E^4 sur l'intervalle de la fenêtre choisie. Cette autre fonction caractéristique présentent des pics très marqués lors du début des phases P et S, mais présentent beaucoup de bruit, qui pourraient être liés à la fenêtre choisie trop petite.

4 Détection et pointage

4.1 Détection

Contrairement au calcul de fonctions caractéristiques vu précédemment, la détection d'un éventuel séisme implique l'introduction de différents paramètres qui dépendent de la nature et la localisation du

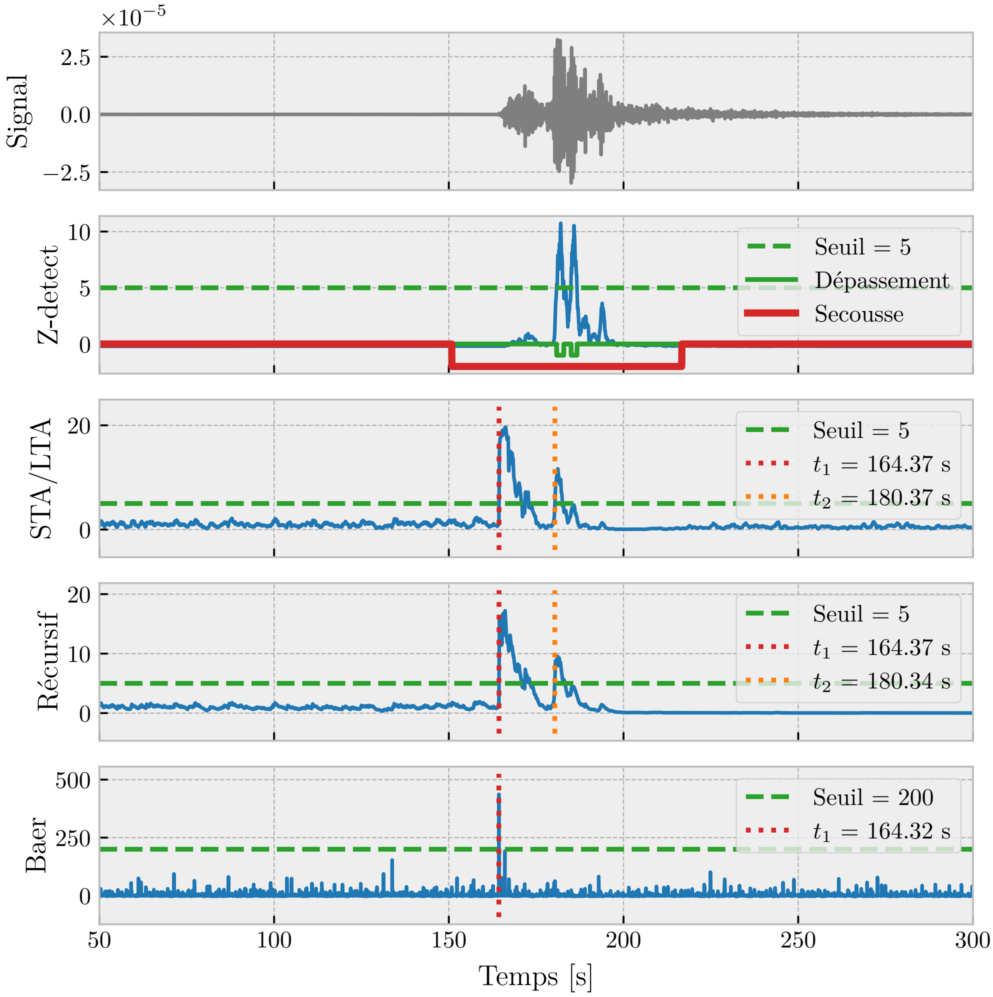

FIGURE 6 – Visualisation de la détection et du pointage du séisme induit de Strasbourg du 26 juin 2021 de magnitude 4. Le signal d'entrée (gris) correspond à l'enregistrement de la vitesse de déplacement du sol au niveau de la station permanente d'Illfurth (ILLF) traité. Les fonctions caractéristiques de ce signal (bleu) sont représentés avec leurs seuils respectifs (vert pointillé).

séisme. Par exemple, nous pouvons considérer un premier seuil qui permet de déclencher des opérations de vérification. Mais ensuite il faut vérifier que le seuil est dépassé pendant une certaine durée. De la même façon nous pouvons introduire de nombreux paramètres.

Ces paramètres se déduisent de la littérature mais varient beaucoup d'une publication à une autre. Ils sont ajustées à partir de données de pointages manuels sur les échantillons de séismes qu'on cherche à étudier. Ces ajustements s'éloignant du "projet informatique" nous avons limité le nombre de paramètres. Le Z-détecteur présente l'avantage de s'adapter automatiquement à la variance du bruit fond [d'après Withers (1982)]. Ainsi les paramètres de détection varient peu et cette fonction caractéristique est intéressante à utiliser pour la détection de séismes plus que pour le pointage, même si les autres fonctions caractéristiques auraient également pu être utilisées pour jouer ce rôle de détection.

Notre algorithme se base d'abord sur la détection du séisme en vérifiant que le seuil est dépassé pendant une certaine période. Si c'est le cas, nous pouvons considérer qu'il y a un séisme et l'algorithme va sélectionner un intervalle en se basant sur un deuxième seuil plus faible et en élargissant cet intervalle. En sortie nous avons l'intervalle de temps correspondant à la secousse à partir duquel nous pourrons pointer le début des phases sismiques en se basant sur d'autres fonctions caractéristiques.

4.2 Pointage

Le pointage est dépendant de la détection, il a lieu que si un séisme est détecté et prend en compte l'intervalle de temps supposé correspondre à la secousse. Nous gardons tous les intervalles où le seuil d'amplitude est dépassé par une nouvelle fonction caractéristique et qui correspondent à l'intervalle supposé de la secousse précédent. Pour chacun de ces nouveaux intervalles, nous calculons le maximum local d'amplitude de l'intervalle et le début de l'intervalle. Nous trions les intervalles par ordre décroissante par rapport à leur maximum, pour ensuite ne garder que les temps de début d'intervalles. Nous obtenons ainsi plusieurs temps de pointages, normalement deux, correspondant à l'arrivée des ondes P puis S.

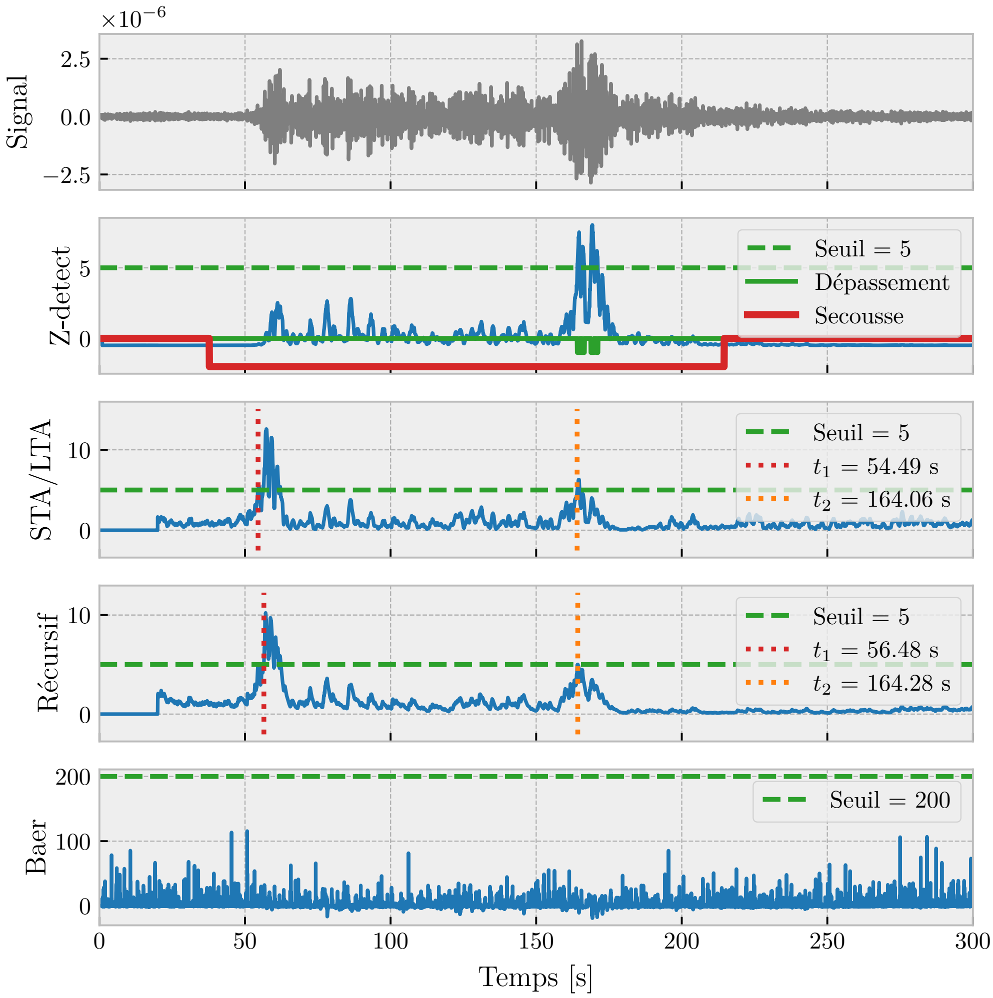

FIGURE 7 – detect

5 Conclusion et discussion

L'objectif de détection d'un séisme et de pointage des ondes P et S de façon automatique a été atteint dans une certaine mesure. Nous constatons que cette méthode fonctionne particulièrement bien pour les séismes régionaux comme le montre la figure 6, un séisme de magnitude 4 à de 100 km de la station. La détection permet d'isoler un intervalle relativement précis de la période de secousse. Le pointage est cohérent pour diverses fonctions caractéristiques, il y a des écarts de l'ordre de la centaine voir de la dizaine de millisecondes. Néanmoins nous constatons que la fonction caractéristique de M. Baer (1987) ne permet pas de pointer le début de la phase S, alors que le pic est très marqué pour le début de la phase P. Nous avons également essayé d'implémenter d'autres fonctions caractéristiques comme celle de Carl, sans succès, du fait du nombre important de paramètres. La méthode classique STA/LTA s'avère particulièrement efficace, et sa variante récursive montre des résultats similaires.

Mais cette méthode n'est plus aussi efficace pour des séismes lointains. En effet, nous voyons sur la figure 7 que le pointage n'est plus aussi précis pour un séisme de magnitude 8 à plus de 3000 km de la station sismique. Cette baisse de précision s'explique naturellement par la plus faible amplitude du signal, il est plus bruité. Le signal bruité obstrue donc les variations d'amplitudes liées aux débuts des phases, en particulier celles de la phases S dont le pointage semble être décalée avec plusieurs secondes de retard.

Cependant, des ajustements peuvent être réalisés pour augmenter la précision de pointage. En effet, nous avons introduit un grand nombre de paramètres qui dépendent de la nature des séismes. Parmi ces paramètres, nous pouvons citer les largeurs des fenêtres STA et LTA, les seuils de détection et de pointage, et la durée minimale de détection. Nous pouvons également rappeler que les fonctions caractéristiques sont calculées à partir d'un signal modifié appelé enveloppe, et le choix de cette enveloppe constitue également un paramètre important.

6 Code source

6.1 Exemple

```python
from picker import *
from obspy import UTCDateTime
from obspy.clients.fdsn import Client

# Chargement des données
starttime = UTCDateTime(2021, 6, 26, 2, 58, 0)
duration = 5*60  # secondes
station = "ILLF" # station d'Illkirch
client = Client("RESIF")  # Réseau français
S = client.get_waveforms(network="FR", station="ILLF", location="00", channel="HH*", starttime=starttim
inventory = client.get_stations(network="FR", station=station, channel="HH*", starttime=starttime, leve

# Correction du signal
pre_process_data(S, inventory)

# On garde les arrays des temps et des amplitudes
T = S.select(channel="HHZ")[0]
times = T.times(type='relative')
data = T.data

# Paramètrage de la taille de la fenêtre
tlta = 20
tsta = 1
nsta, nlta = nstalta_from_times(times, tsta, tlta)

# Calcul des enveloppes
sqa_data = data**2
fc_stewart = mdx_stewart(times, data, 10)
```

```python
allen_data = allen_envelope_np(data)
baer_data = baer_envelope_np(times, data)

# Calcul des fonctions caractéristiques
fc_allen = stalta_allen_np(allen_data, nsta, nlta)
fc_allen_recursiv = stalta_recursiv(allen_data, nsta, nlta)
z = z_swindell_snell(sqa_data, nsta)
fc_baer = z_baer_kradolfer(baer_data, nsta)

# Détection
threshold = 5
potentials = potential_waves(times, z, threshold=threshold, tol=1, delta=3)
earthquake = any(potentials)

# Pointage et affichage
if earthquake:
    earthquake_loc = potential_waves(times, z, threshold, tol=10, delta=3, before=30, after=30)

    potentials = potential_waves(times, fc_allen, threshold, tol=1)
    picks = point_potentials(fc_allen, potentials)
    picks = possible_picks(picks, earthquake_loc)

    print(f'Un séisme a été détecté avec {len(picks)} débuts de phases pointées.')
    for pick in picks:
        time = times[pick]
        print(f't={time}, soit à {(starttime + time).strftime("%H:%M:%S.%f")}.')

else:
    print('Pas de séisme détecté sur ce signal.')
```

6.2 Code Python

```python
import numpy as np


def pre_process_data(S, inventory):
    """Corrige le signal grâce au module Obspy (objets Obspy)"""
    S.attach_response(inventory)
    for T in S:
        # Conversion en signal alternatif (moyenne nulle)
        T.detrend("demean")

        # Filtrage passe-bande
        T.filter("bandpass", freqmin=2, freqmax=10)

        # Conversion l'unité du signal de counts en vitesse (optionnel)
        gain = T.meta.response.instrument_sensitivity.value
        T.data = T.data/gain


def derivate(times, data):
    """Calcul la dérivée temporelle du signal"""
    der_data = [0]
    n = len(data)

    for k in range(n-1):
        dy = (data[k+1] - data[k])
        dt = (times[k+1] - times[k])
```

```python
        der_data.append(dy/dt)

    return np.array(der_data)


def derivate_np(times, data):
    """Calcul la dérivée temporelle du signal de façon optimisée"""
    n = len(data)
    der_data = np.zeros(n)

    der_data[1:] = np.diff(data)/np.diff(times)

    return der_data


def mdx_stewart(times, data, n_check=8):
    """Calcul l'enveloppe modifiée de Stewart"""
    n = len(data)
    dx = derivate(times, data)
    mdx = np.zeros(n)
    n_good = 0

    for i in range(1, n):
        if np.sign(dx[i-1]) == np.sign(dx[i]):
            n_good += 1
        else:
            n_good = 0

        if n_good > n_check:
            mdx[i] = mdx[i-1]+dx[i]
        else:
            mdx[i] = dx[i]

    return mdx


def envelope_approx(times, data):
    """Calcul l'enveloppe géométrique du signal"""
    env_times = []
    env_data = []
    n = len(data)

    for k in range(1, n-1):
        v1 = data[k] - data[k-1]
        v2 = data[k] - data[k+1]

        # ce point est un extremum local (un "pic")
        if v1 > 0 and v2 > 0 or v1 < 0 and v2 < 0:
            env_times.append(times[k])
            env_data.append(abs(data[k]))

    return np.array(env_times), np.array(env_data)


def envelope_approx_np(times, data):
    """Calcul l'enveloppe géométrique du signal de façon optimisée"""
    indices = np.nonzero(np.diff(np.sign(np.diff(data))))[0]+1

    env_times = times[indices]
    env_data = np.abs(data[indices])

```

```python
    return env_times, env_data


def allen_envelope(data):
    """Calcul l'enveloppe de Allen du signal"""
    allen_data = [0]
    n = len(data)

    ci = sum(abs(x) for x in data) / \
        sum(abs(data[i]-data[i-1]) for i in range(n))

    for i in range(1, n):
        allen_data.append(data[i]**2+ci*(data[i]-data[i-1])**2)

    return np.array(allen_data)


def allen_envelope_np(data):
    """Calcul l'enveloppe de Allen du signal de façon optimisée"""
    allen_data = np.zeros(len(data))

    ci = np.sum(np.abs(data))/np.sum(np.abs(np.diff(data)))
    allen_data[1:] = data[1:]**2 + ci*np.diff(data)**2

    return allen_data


def baer_envelope(times, data):
    """Calcul l'enveloppe de Baer et Kradolfer du signal"""
    baer_data = [0]
    n = len(data)
    der_data = derivate(times, data)
    sqa_data = data**2
    sqa_der_data = der_data**2

    ci = sum(sqa_data)/sum(sqa_der_data)

    for i in range(n):
        baer_data.append(sqa_data[i]+ci*sqa_der_data[i])

    return baer_data


def baer_envelope_np(times, data):
    """Calcul l'enveloppe de Baer et Kradolfer du signal de façon optimisée"""
    sqa_data = data**2
    sqa_der_data = derivate(times, data)**2

    ci = np.sum(sqa_data)/np.sum(sqa_der_data)

    baer_data = sqa_data+ci*sqa_der_data

    return baer_data


def nstalta_from_times(times, tsta, tlta):
    """Convertit les durées des fenêtres STA et LTA en nombre de points"""
    dt = times[1] - times[0]

    nsta = int(tsta/dt)
    nlta = int(tlta/dt)
```

```python

    return nsta, nlta


def stalta_allen(data, nsta, nlta):
    """Calcul la fonction caractéristique d'Allen"""
    fc = [0]*(nlta-1)
    n = len(data)

    for k in range(n-nlta+1):
        i1 = k
        i2 = k + nlta
        im = i2 - nsta

        # L'utilisation de la moyenne de Numpy réduit considérablement les calculs
        sta = np.average(data[im:i2])
        lta = np.average(data[i1:i2])

        fc.append(sta/lta)

    return fc


def stalta_allen_np(data, nsta, nlta):
    """Calcul la fonction caractéristique d'Allen de façon optimisée"""
    n = len(data)
    fc = np.zeros(n)
    sta = np.zeros(n)
    lta = np.zeros(n)

    # Calcul de la somme cummulée
    cumsum_data = np.cumsum(data)

    # Calcul des moyennes glissantes de STA et LTA à partir de la somme cummulée
    sta[nsta:] = (cumsum_data[nsta:] - cumsum_data[:-nsta])/nsta
    lta[nlta:] = (cumsum_data[nlta:] - cumsum_data[:-nlta])/nlta

    # Le rapport STA/LTA est calculé pour les termes non nuls
    indices = lta.nonzero()
    fc[indices] = sta[indices] / lta[indices]

    return fc


def stalta_recursiv(data, nsta, nlta):
    """Calcul la fonction caractéristique d'Allen de façon récursive"""
    sta = 0
    lta = 0
    fc = np.zeros(len(data))

    # Pour calculer les moyennes glissantes on doit calculer dès le 1er terme
    # même si on ne garde après que les termes après nlta
    for i, y in enumerate(data):
        sta = (y - sta)/nsta + sta
        lta = (y - lta)/nlta + lta

        if i >= nlta and lta != 0:
            fc[i] = sta / lta

    return fc

```

```python

def z_swindell_snell(data, nsta):
    """Calcul la fonction caractéristique de Swindell et Snell"""
    n = len(data)
    z = np.zeros(n)
    sta = np.zeros(n)

    for k in range(nsta, n):
        sta[k] = np.average(data[k-nsta:k])

    mu = np.average(sta)
    # théorème de König-Huygens
    sqa_sigma = np.sqrt(np.average(sta**2) - mu**2)

    for k in range(nsta, n):
        if sqa_sigma != 0:
            z[k] = (sta[k] - mu)/sqa_sigma

    return z


def z_swindell_snell_np(data, nsta):
    """Calcul la fonction caractéristique de Swindell et Snell de façon optimisée"""
    n = len(data)
    fc = np.zeros(n)
    sta = np.zeros(n)

    cumsum_data = np.cumsum(data)
    sta[nsta:] = (cumsum_data[nsta:] - cumsum_data[:-nsta])/nsta

    mu = np.average(sta)
    sqa_sigma = np.sqrt(np.average(sta**2) - mu**2)

    indices = sqa_sigma.nonzero()
    fc[indices] = (sta - mu)/sqa_sigma

    return fc


def z_baer_kradolfer(data, nsta):
    """Calcul la fonction caractéristique de Baer et Kradolfer"""
    n = len(data)
    fc = np.zeros(n)

    for k in range(nsta, n):
        sqa_mu = np.average(data[k-nsta:k])**2
        sqa_sigma = np.average(data[k-nsta:k]**2) - sqa_mu
        if sqa_sigma != 0:
            fc[k] = (data[k]**2 - sqa_mu) / sqa_sigma

    return fc


def potential_waves(times, data, threshold, tol=0, delta=0, before=0, after=0):
    """Détecte un éventuel séisme à partir du signal d'une fonction caractéristique"""
    n = len(data)
    dt = times[1]-times[0]

    ntol = round(tol/dt)
    ndelta = round(delta/dt)
    nbefore = round(before/dt)
```

```python
    nafter = round(after/dt)

    potentials = data > threshold

    # Ajoute une tolérence en temps pour la désactivation du dépassement du seuil
    if ntol > 0:  # évite les calculs inutiles
        i = 0
        while i < (n-ntol-1):
            if potentials[i] and not potentials[i+1]:
                for j in range(ntol):
                    if potentials[i+j]:
                        potentials[i+1:i+j] = 1
                        i += j
            i += 1

    # Vérifie s'il existe un dépassement de seuil assez long
    if ndelta > 0:
        i = 0
        while i < (n-ndelta):
            if potentials[i]:
                j = i
                while j < n and potentials[j]:
                    j += 1
                if (j-i) < ndelta:
                    potentials[i:j] = 0
                i += j
            i += 1
        potentials[i:n] = 0

    # Ajoute un contour pour sélectionner une plage plus grande
    if nbefore > 0 or nafter > 0:
        i = 1
        while i < (n-1):
            if potentials[i]:
                if not potentials[i-1]:
                    potentials[max(i-nbefore, 0):i] = 1
                if not potentials[i+1]:
                    potentials[i+1:min(i+nafter+1, n)] = 1
                    i += nafter
            i += 1

    return potentials


def point_potentials(data, potentials):
    """Pointe le début de toutes les potentielles fenêtre d'arrivée des ondes et les tries par rapport à leur maxi
    above = False
    points = []
    for i in range(len(potentials)):
        if potentials[i]:  # cas d'une fenêtre
            y = data[i]
            if not above:  # pointage du début de la fenêtre
                points.append([i, y])
                above = True
            if y > points[-1][1]:  # recherche du maximum local de la fenêtre
                points[-1][1] = y
        elif above:
            above = False

    # On trie par rapport aux maximums locaux de chaque fenêtre
    points.sort(key=lambda pair: pair[1], reverse=True)
```

```python
    # On ne garde que les indices de pointage
    return [pair[0] for pair in points]


def possible_picks(picks, earthquake_loc):
    """Garde les instants compris dans l'intervalle détecté"""
    return [pick for pick in picks if earthquake_loc[pick]]


def point_lmax(fc, threshold):
    """Renvoie l'index correspondant au début de l'intervalle dépassant le seuil et comprenant le maximum global""
    threshold = 5
    imax = 0
    emax = fc[0]
    n = len(fc)
    for i in range(1, n):
        if fc[i] > emax:
            emax = fc[i]
            imax = i

    j = imax
    while j > 0 and fc[j] > threshold:
        j -= 1

    return j
```

6.3 Code C

```c
#include <stdlib.h>

float *stalta_recursiv(float *data, int nsta, int nlta, int N)
{
    float *fc = malloc((sizeof(float) * N));
    float sta, lta;
    sta = 0;
    lta = 0;
    int i;

    for (i = 0; i < N; i++)
    {
        sta = (*(data + i) - sta) / nsta + sta;
        lta = (*(data + i) - lta) / nlta + lta;
        if (i > nlta && lta != 0)
            *(fc + i) = sta / lta;
        else
            *(fc + i) = 0;
    }

    return fc;
}

float *stalta_allen(float *data, int nsta, int nlta, int N)
{
    float average(float *data, int N);
    float *fc = malloc((sizeof(float) * N));
    float sta, lta;
    int i;

    for (i = 0; i < nlta; i++)
        *(fc + i) = 0;

```

```c
    for (i = nlta; i < N; i++)
    {
        sta = average(data + i - nsta, nsta);
        lta = average(data + i - nlta, nlta);
        *(fc + i) = sta / lta;
    }

    return fc;
}

float average(float *data, int n)
{
    float s = 0.0;
    int i;
    for (i = 0; i < n; i++)
        s += *(data + i);
    return s / n;
}

void freeptr(void *ptr)
{
    free(ptr);
}
```

Références

R. Allen. Automatic earthquake recognition and timing from single traces. *Bulletin of the Seismological Society of America*, 68(5) :1521–1532, 1978. doi : 10.1785/BSSA0680051521. URL `https://doi.org/10.1785/BSSA0680051521`.

R. Allen. Automatic phase pickers : their present use and future prospects. *Bulletin of the Seismological Society of America*, 72(6) :225–242, 1982. doi : 10.1785/BSSA07206B0225. URL `https://doi.org/10.1785/BSSA07206B0225`.

Olivier Cuenot. Les algorithmes de détection automatique d'ondes sismiques, 2003.

Y. Gaol Lumban Gaol. Preliminary results of automatic p-wave regional earthquake arrival time picking using machine learning with sta/lta as the input parameters. *IOP Conference Series : Earth and Environmental Science*, 873(1), oct 2021. doi : 10.1088/1755-1315/873/1/012060. URL `https://dx.doi.org/10.1088/1755-1315/873/1/012060`.

A. Khalaf. *Développement d'une nouvelle technique de pointé automatique pour les données de sismique réfraction.* PhD thesis, Université Pierre et Marie Curie, 2016.

L. Küperkoch. Automated determination of P-phase arrival times at regional and local distances using higher order statistics. *Geophysical Journal International*, 181(2) :1159–1170, 05 2010. ISSN 0956-540X. doi : 10.1111/j.1365-246X.2010.04570.x. URL `https://doi.org/10.1111/j.1365-246X.2010.04570.x`.

U. Kradolfer M. Baer. An automatic phase picker for local and teleseismic events. *Bulletin of the Seismological Society of America*, 77(4) :1437–1445, 1987. doi : 10.1785/BSSA0770041437. URL `https://doi.org/10.1785/BSSA0770041437`.

A. Lomax M. Vassallo, C. Satriano. Automatic picker developments and optimization : A strategy for improving the performances of automatic phase pickers. *Seismological Research Letters*, 83 : 541–554, 05 2012. doi : 10.1785/gssrl.83.3.541.

Obspy. Obspy : A python toolbox for seismology/seismological observatories. URL `https://github.com/obspy/obspy`.

S. Stewart. Real-time detection and location of local seismic events in central california. *Bulletin of the Seismological Society of America*, 67(2) :443–452, 1977. doi : 10.1785/BSSA0670020433. URL `https://doi.org/10.1785/BSSA0670020433`.

S. Lorenz W. Vanderkulk, F. Rosen. Large aperture seismic array signal processing study. *IBM Final Report*, 1965.

Withers. A comparison of select trigger algorithms for automated global seismic phase and event detection. *Bulletin of the Seismological Society of America*, 88(1) :95–106, 1982. doi : 10.1785/BSSA0880010095. URL `https://10.1785/BSSA0880010095`.

Gregory. Beroza Zhu. Weiqiang. Phasenet : A deep-neural-network-based seismic arrival time picking method. *Geophysical Journal International*, 216, 03 2018. doi : 10.1093/gji/ggy423.