CC-FJpy: A Python Package for seismic ambient noise cross-correlation and the frequency-Bessel transform method

Authors:

Zhengbo Li[1,2], Jie Zhou[5], Gaoxiong Wu[3], Jiannan Wang[3], Gongheng Zhang[3], Sheng Dong[6], Lei Pan[3], Zhentao Yang[3], Lina Gao[3], Qingbo Ma[6], Hengxin Ren[3], & Xiaofei Chen[2,3,4*]

1. Academy for Advanced Interdisciplinary Studies, Southern University of Science and Technology, Shenzhen, 518055, China
2. Shenzhen Key Laboratory of Deep Offshore Oil and Gas Exploration Technology, Southern University of Science and Technology, Shenzhen, 518055, China
3. Department of Earth and Space Sciences, Southern University of Science and Technology, Shenzhen, 518055, China
4. Southern Marine Science and Engineering Guangdong Laboratory (Guangzhou), Guangzhou, 511458, China
5. School of Earth and Space Sciences, Peking University, Beijing, 100000, China
6. School of Earth and Space Sciences, University of Science and Technology of China, Heifei, 230026, China

Abstract

In the past two decades, surface wave imaging based on seismic ambient noise cross-correlation (CC) has been one of the most important technologies in the field of seismology. With the development of this technology, high-mode surface waves have received increasing attention, especially after the proposition of the frequency-Bessel transform (F-J) method, which can effectively extract multimode dispersion curves from ambient noise data. In the past few years, our research group has made many attempts to improve this method. We summarized these experiences and the corresponding algorithm for fast CC, and packaged them into a Python package called CC-FJpy. It is commonly understood that CC takes a good deal of time. However, we found that a simple reorganization of the CC logic can achieve computational acceleration by a multiple of tens or even hundreds in comparison with classical CC open-source programs for *N* stations. For the F-J method, we use Nvidia's graphics processing unit (GPU) to speed up computation, and this approach achieves a hundreds-fold computational acceleration. We have encapsulated our experiences and technologies into CC-FJpy and submitted it to various types of data tests to ensure its speed and ease of use. We hope that providing the open source of CC-FJpy can benefit the development of surface wave studies and make it easier to start with high-mode surface waves. We look forward to your use and valuable suggestions.

Introduction

In the past two decades, significant understandings of underground structures of different scales have been facilitated by the development of surface wave imaging with noise cross-correlation (CC) technology (e.g., Campillo & Paul, 2003; Shapiro et al. 2005; Sabra et al., 2005a, b; Yao et al. 2006; Bensen et al. 2009; Lin et al., 2009, 2011; Fang et al, 2015, 2016; Shen et al., 2016). For ambient noise surface wave imaging, especially for lithospheric imaging, most often the fundamental mode is0 obtained and inversed (e.g., Bensen 2007). Numerous studies have confirmed that high-mode surface wave dispersion curves can provide more constraints on underground structures (e.g., Nolet & Panza, 1976; Yokoi, 2010; Pan et al., 2018; Wu et al., 2020). Wang et al. (2019a) proposed the frequency-Bessel transform (F-J) method, which can efficiently extract Rayleigh wave multimode dispersion curves from ambient noise cross-correlation functions (CCFs). Hu et al. (2020) verified that this method can be easily applied to Love waves; Li & Chen (2020a; 2020b) extended this method to the application to seismic records, and confirmed that this method can also extract the dispersion of PL waves. Zhan et al. (2020) applied this method to imaging in Northeast China and updated the local 3-dimensional velocity model. To further promote studies on high-mode surface waves and to ensure that more scholars can easily use this method, we summarized our experiences in recent years and packaged our GPU F-J code with our recently developed fast CC programs into an open-source Python package CC-FJpy.

Noise cross-correlation technology is one of the most important technologies in seismology. CCFs obtained by CC can be approximated as Green's functions, which means that a large number of seismological methods no longer rely on local earthquakes (e.g., Weaver & Lobkis 2004; Sánchez-Sesma & Campillo 2006). CCFs have been widely used in surface wave imaging (e.g., Yao et al., 2006; Bensen et al., 2009), body wave imaging (e.g., Poli et al., 2012; Feng et al., 2017), full wave inversion (Sager et al., 2017, 2020; Wang et al., 2019b), attenuation emulation (e.g., Lawrence et al., 2013) and so on. It is commonly understood that the CC process is often time-consuming, especially when the overlap of time is needed (Seats et al., 2012). Ventosa et al. (2019) attempted to accelerate the CC though GPUs. Although they accelerated the process of a single CC for two stations, they could not accelerate the CC of *N* stations well. After we carefully studied the CC process and some widely used CC codes, we found that although CC technology has been widely used for more than ten years since the early application of CC technology, there is still a relatively large optimization space. For *N* stations, C_N^2 times CCs are required. In many classic programs, each CC between two stations comprises reading data, preprocessing and CC. In fact, only *N* reading data and preprocessing steps are required. Furthermore, the essence of CC between records A and B is multiplication in the frequency domain:

$$CC = A(\omega) * conj(B(\omega)), \#1$$

where *conj* is the conjugation. All the classic programs pack this step as a function (for example, the MATLAB function *xcorr*), which means that every CC needs two fast Fourier transforms (FFTs), which causes many repetitive FFT calculations. The total number of FFTs called is $2C_N^2$, which can also be reduced to *N*. Based on these two points, we adjusted the logic of CC, wrote the kernel in the C language and encapsulated it as a Python interface through Cython. Although this sounds like a simple change, the effect is surprisingly good: its efficiency is ten to hundreds of times higher than that of most classic CC programs. More importantly, our programs have very small requirements for computing resources. In many cases, simple parallelism on a laptop is enough to make it dozens of times faster than traditional programs on a server. In addition, our program is also very easy to modify to adopt different kinds of improvements (e.g., Shen et al., 2012; Xie et al., 2020) to the CC equation (equation 1).

For the F-J method, the core is to numerically realize the integral of equation 2.

$$I(\omega,k) = \int_0^\infty G(r,\omega)J_0(kr)rdr, \#2$$

where k is the wavenumber, r is the epicenter distance, $J_0(x)$ is the 0^{th} Bessel function of the first kind and $G(r,\omega)$ can be CCFs or earthquake records. Generally, a trapezoidal integral is a good choice, but this ignores the known characteristics of the Bessel function. Wang et al. (2019a) gave a more accurate integration format, which will, however, increase the amount of calculation. To balance efficiency and accuracy, we use the GPU to accelerate the process. The GPU is very suitable for this type of calculation and can achieve hundreds of speedups, which can shorten the original FJ process from a range of tens of minutes to hours to one of tens of seconds to minutes. In addition, we encapsulated the integration of using the Hankel function instead of the Bessel function, which has proven to be effective in removing "crossed" artifacts (Forbriger, 2003). Different integration methods and GPU or non-GPU support are provided in CC-FJpy to facilitate the needs of different users.

In recent decades, computer technology has brought revolutionary changes to many industries. One of the most important drivers of these changes is that programming language and complex algorithms have been efficiently encapsulated, so that numerous participants can quickly learn and master the developed technology. The most typical example is the development of machine learning. Now, even a middle school student can train his or her own model using TensorFlow, PyTorch or other Python machine learning packages. In the field of geophysics, there are also many informed scholars who have developed efficient open-source software programs, such as the Generic Mapping Tools (GMT, https://www.generic-mapping-tools.org/) and Obspy (https://docs.obspy.org/, Beyreuther et al. 2010). Encouraged by this, we decided to share our small contribution in the direction of CC and high-mode surface waves to serve all colleagues, and we have committed to maintaining the update for the foreseeable future. You can obtain CCFJpy from https://github.com/ColinLii/CC-FJpy. We hope that through our program package, CC and the extraction of high-order dispersion through the F-J method will become easier, especially for scholars who are beginning to study this area. In addition, we humbly hope for valuable suggestions.

Implementation

The imaging process through the F-J method can be simply summarized as the following four steps: ① read data & preprocess, ② cross-correlation, ③ F-J scan and ④ dispersion curve extraction & inversion (Figure 1). Among them, ① mainly depends on the storage format (e.g., SAC or miniSeed) and storage order of the data, and ④ has a lot of personalized solutions (e.g., Shen et al., 2012; Pan et al. 2018; Dereiling et al., 2019). We highly recommend Obspy for reading and preprocessing data (https://docs.obspy.org/, Beyreuther et al. 2010). The calculations of ② and ③ are relatively fixed. Thus, CC-FJpy mainly deals with ② and ③ and is divided into two sub-packages: CCpy and FJpy. The two sub-packages can be used together or completely independently. In addition, although ④ has a large number of personalized programs, we plan to add several inversion methods that we believe are efficient and robust as examples in future updates.

CCPY: a Python sub-package for rapid cross-correlation

First, let us briefly review the basic formula of cross-correlation:

$$C_{1,2}(t) \approx \int_0^{t_C} v_1(\tau)v_2(t+\tau)d\tau ,\#3$$

where $v_1(t)$ and $v_2(t)$ are the continuous broadband records of stations 1 and 2 in the time window [0, t_C]. Usually, t_C is not the total continuous recording time, as the recording is divided into unit time lengths such as one hour, one day or one week. CCFs are obtained by superimposing a large number of $C_{1,2}(t)$ with different time windows. The realization of equation 3 in the time domain is time-consuming, so for most programs, it is implemented in the frequency domain.

$$C_{1,2}(\omega) \approx dft(v_1)conj(dft(v_2)),\#4$$

where dft is the discontinuous Fourier transform and $conj$ is the conjugation.

From equation 3 and equation 4, the CC of the two stations is appears to be very simple and without much room for acceleration. However, the strategy for calculating the CC affects the calculation time. We first talk about the outputs as the frequency-domain CC functions $C_{j,k}(\omega)$ for N stations. The classic strategy is what we called strategy 1, the completely independent strategy, which means that every time the two stations are cross-correlated, the data of the two stations are read and correlated (Figure 2a). For this strategy, for one CC time unit, N(N-1) times reading data and N(N-1)/2 times CC are required. Obviously, there are many duplications in the reading data step. An improvement in this strategy is to read the data of N stations in once and then N(N-1)/2 times CC can be performed (Figure 2b). We call this strategy 2, the shared memory strategy. Compared with the completely independent strategy, the number of readings drops from N(N-1) times to N times. Furthermore, according to equation 4, we can divide CC into two parts: the FFT and multiplication (Figure 2d). The FFT can be shared like the reading data. Thus, we have strategy 3: shared memory and the FFT strategy (Figure 2c). In addition to the reading time, the CC time can also be reduced. It is worth mentioning that we use the C language to call the fftw-3 package (https://www.fftw.org) for FFT, as it is approximately 3 times faster than Python numpy fft. All the C codes are encapsulated as a Python interface through Cython.

For the outputs are the time-domain CC functions $C_{i,j}(t)$, the shared memory and FFT strategy have more significant time advantages for M CC time units. For the first two strategies, since each CC outputs the time-domain CC functions, $M \times N \times (N-1)/2$ inverse FFTs (IFFTs) are needed before the overlap. However, since the Fourier transform is linear, we can overlap in the frequency domain and then perform the IFFT, which means that only $N \times (N-1)/2$ IFFTs are needed. According to our test, the multiplication takes much less time than reading and preprocessing data, the FFT and the IFFT. The efficiency of the acceleration through strategy 3 is positively related to the total number of stations N and the total number of stacking days M. The larger N and M are, the more obvious the improvement delivered by strategy 3. Different machines and different data will have a great impact on the time cost. After using strategy 3, the overall CC efficiency is improved by one to two orders over strategies 1 and 2. As the amount of calculation is greatly reduced, CCpy is suitable for both personal PCs and notebooks.

In many cases, to improve the quality of the CC, overlaps of units are needed (Seats et al., 2012). In our program, we designed a larger reading unit T_C for reading, and the overlap of t_C is executed within to improve efficiency. You can select an overlap rating, whether to use spectral whitening, whether to use onebit and other options for different situations with the interface ccfj.cc. We will show you a specific example of data from USArray in the next section to show the acceleration efficiency. For details, please read the package manual.

FJPY: a Python sub-package for the F-J method through GPU

The core of the F-J method is the numerical realization of equation 2. For *N* observed $G(r_i,\omega)$ arranged in order from station distance for CCFs or epicenter distance for earthquake records, the trapezoidal integral can be used to approximate equation 2:

$$I(\omega,k) \approx \frac{1}{2}\sum_{i=1}^{N-1}\left(G(r_i,\omega)J_0(kr_i) + G(r_{i+1},\omega)J_0(kr_{i+1})\right)(r_{i+1} - r_i)\,.\#5$$

Note that for trapezoidal integration, $G(r,\omega)$ is only obtained at the observation points r_i, but $J_0(kr)$ is known from 0 to ∞. Thus, Wang et al. (2019a) gave another numerical integral format of equation 4 through linear approximation of Green's function:

$$I(\omega,k) \approx \sum_{i=1}^{N-1}\left\{\frac{1}{k}G(r,\omega)rJ_1(kr) + \frac{b_j}{k^3}\left[krJ_0(kr) - B_0(kr)\right]\right\}\Big|_{r_j}^{r_{j+1}},\#6$$

where $b_j = \frac{G(r_{j+1},\omega)-G(r_j,\omega)}{r_{j+1}-r_j}$, and $B_0(x) = \int_0^x J_0(\eta)d\eta$. In the early implementation of the F-J method, the calculation of $B_0(x)$ is by trapezoidal integration. Later, we find the primitive of $B_0(x)$:

$$\int J_0(x)dx = xJ_0(x) + \frac{\pi x}{2}\left[J_1(x)Q_0(x) - J_0(x)Q_1(x)\right],\#7$$

where $Q_i(x)$ is the i^th Struve function, which can be calculated by the subroutine of Ruckdeschel (1981).

For dispersion features, seismologists prefer to display in the frequency-phase-velocity (f-c) domain over the frequency-wavenumber (f-k) domain. The domain conversion can be performed by:

$$c = \frac{2\pi\omega}{k}.\#8$$

Commonly, when calculating *nc* phase velocity points and *nf* frequency points, regardless of whether equation 5 or equation 6 is used, the size of the calculation of the F-J method is quite large, especially for noise data. However, the F-J method is naturally suitable for parallel acceleration through the Nvidia GPU. Each calculation of c_i and ω_i is not related to each other. Compute unified device architecture (CUDA) programming enables us to execute F-J integration on GPU devices. Equation 5 and 6 can be packaged into different 'kernels' which is the code run on the GPU device, and be scheduled by *nf*×*nc* GPU threads. Further, we encapsulate the CUDA program with Python; this makes it possible to quickly implement equations 5 and 6 by calling function ccfj.fj with different parameters. It is worth noting that changing the Bessel function in equations 5 and 6 into the first kind of Hankel function will help eliminate the "cross" artifact (Forbriger, 2003). We have also added parameters to control using the Bessel function or Hankel function.

It is also worth noting that for noise data, we often only use the real part of the CCFs for calculation, while for seismic data, we calculate the real and imaginary parts of the recorded spectrum and take $|I(\omega,k)|$. This is mainly because an earthquake has a source time function, which will affect the results of the pure real or imaginary part. In addition, Li & Chen (2020) noted that the F-J method for seismic records often requires auxiliary time windows (multi-windows F-J method, MWFJ). Therefore, we specifically designed the ccfj.mwfj interface for earthquake events. For details, please refer to the manual and the Python examples.

Examples of USArray

Two application examples are illustrated: one is an ambient noise example consistent with Wu et al. (2020) and the other is an earthquake example consistent with Li & Chen (2020).

Ambient Noise

The data we use are half a year (182) of continuous records from June 1st (day 152) to December 1st (day 334) in 2011 of 96 stations from the USArray (Figure 3a). The original data size is approximately 55GB. We cropped data by day (T_C), which is probably the most common split time. Then, we downsampled to 4 Hz, demeaned, detrended, removed the instrument response and saved it as SAC files. After decompression, the size of a single SAC file is 1.31 MB and the total file size is 20.9 GB. If the data are stored as a longer period (T_C), such as week or month, the reading efficiency and calculation efficiency will be higher. Hourly CC with 90% time overlaps and spectral whitening is adopted during CC. Figure 3b shows the CCFs in the frequency-domain recovered by CC-FJpy, while Figure 3c shows the time-domain CCFs obtained by the IFFT of the frequency CCFs. Both the frequency-domain and time-domain CCFs have good coherence.

As mentioned in the last section, we care most about the computational efficiency. Please note that different machines and data will have a greater impact on the results. The CPU applied for the test was a 10-cores Intel(R) Core (TM) i9-10900K with 64 GB ddr4 2666 MHz RAM and Seagate Exos 7E8 ROM. For the accuracy of the test, we read 100 different SAC files of one day and performed demeaning and detrending. The average time of reading data, demeaning and detrending is approximately 0.05 seconds. Similarly, we calculated that the time required to calculate the FFT with numpy is 0.093 seconds while that with fftw-3 is 0.029 seconds for hourly CC with 90% overlap at one station. The multiplication time of the numpy array is 0.008 seconds while the of fftw-3 is 0.002 seconds (Figure 4a). Figure 4b shows that the different strategies need to calculate the number of times to read data & demean & detrend, conduct the FFT and multiplication. The number of reads and FFTs required by strategy3 has drops sharply compared to the other two strategies, which leads to the CC time for 96 stations in a day being much less than that of other two strategies. To calculate CC in the frequency-domain for 96 stations in one day, strategy 1 takes 815 seconds, strategy 2 takes 542 seconds, and strategy 3 takes 11.5 seconds. Figure 4c-e shows the percentage of the different strategies, where the area is proportional to the time used, and the read and FFT time saved by strategy 3 is easily seen. For CCpy, which uses strategy 3, it takes less than 1800 seconds to complete the CC of the 4 Hz data of 96 stations for half a year in series. Under parallelism, since a large part of the time in the cross-correlation is reading data, the efficiency of parallelism does not entirely depend on the number of cores, but also depends on the speed of the hard disk. It takes less than 10 minutes to use 20 threads in parallel. For the first two strategies, it takes more than 24 hours to use serial and at least 4 hours to use parallel. It is worth noting that the time is measured on the author's personal computer, which may be quite unstable.

It should be noted that the above comparison does not consider the time taken by the IFFT. If the time of the IFFT is considered, as Figure 2d shows, for strategies 1 and 2, every CC needs a 1-time IFFT, which means that for one-day data with 90% overlap, $231 \times C_N^2$ IFFTs are needed, and for 182-day data, $182 \times 231 \times C_N^2$ IFFTs are needed. However, for strategy 3, only C_N^2 IFFTs are required, which will further highlight the acceleration ratio of strategy 3 compared to those of strategies 1and 2.

We use the trapezoidal integral (equation 5) based on the Bessel function, the linear approximate integral (equation 6) based on the Bessel function, the trapezoidal integral (equation 5) based on the Hankel function and the linear approximate integral (equation 6) based on the Hankel function to extract the dispersion spectrum from the CCFs (Figure 5). Compared with trapezoidal integration, linear approximate integration can improve the quality of the dispersion spectrum. The Hankel function can effectively remove "cross" artifacts. Since GPU acceleration is used, the calculation time is approximately tens of seconds, and the specific values of the time cost are marked in the subfigures in Figure 5. The GPU applied in the test is the Nvidia RTX 2070 super, which is commonly used and at a suitable price. Both the Linux platform and the Windows platform are supported. We also provide the corresponding calculation program without GPU acceleration in the program package, but the calculation without a GPU is relatively slow. If it is completely serialized, it will take close to 3 hours of calculation for the linear approximate integral of the Bessel function (Figure 5b), while the calculation with GPU acceleration is approximately 11 seconds. Therefore, we strongly recommend using an Nvidia GPU to accelerate; even a very ordinary GPU will achieve a high acceleration.

Earthquake

Here, we repeat the example of the Mw 5.7 Oklahoma earthquake in Li & Chen (2020) with the FJPY. In this case, MWFJ with three time windows, NoWin, which means no time window, Win1 [3.2, 3.7] km/s and Win2 [3.7, 4.3] km/s, are applied. Here, we only show the results calculated according to the Bessel function and Hankel function corresponding to equation 6. Figures a, b and c show the dispersion spectrum extracted with three time windows by equation 6 with the Bessel function, while Figures d, e and f show the dispersion spectrum extracted with three time windows by equation 6 with the Hankel function. The specific code calls have been shown in “earthquake.ipynb”. We prefer users try both instead of comparing the results, as, the calculation of seismic records is generally approximately a few seconds.

Discussion and Conclusions

The F-J method is an effective method for extracting high-mode surface wave dispersion from various types of seismic records, and has recently received increasing attention. We summarized the application of our group’s research in recent years on the F-J method, and encapsulated our codes and experiences into a Python package. We hope that through open source, we have made it more convenient for more seismologists to use the F-J method.

At present, this package contains two parts: CCpy, which performs fast noise cross-correlation, and FJpy, which is accelerated by Nvidia’s GPU. Although we only made minor modifications to the existing cross-correlation logic, CCpy still delivers several times more speed up, so that CC that used to take days or weeks will now only take tens of minutes to a few hours. We believe this is helpful not only for the F-J method with ambient noise but also for many other seismological studies. The GPU acceleration drops the time taken by the F-J method, especially the application of ambient noise, from tens of minutes to approximately 1 minute, which greatly improves the efficiency of F-J imaging. The GPU can also be used to accelerate the CC process (Ventosa et al., 2019), and we are considering adding it in a future update. However, we are concerned about that GPUs cannot bring qualitative acceleration, such as CCpy, because reading data is required. As shown in the example in Figure 4, even if the FFT time and the multiplication time are both 0, the reading still needs more than 1/3 of the time. However, with the increasing popularity of dense arrays, especially the application of distributed acoustic sensing (DAS, Mateeva et al., 2014; Hartog, 2017) technology, any attempts to improve efficiency should be encouraged.

We believe that with the popularity of computers today, the ease of use of codes is very important for industry development. We chose to encapsulate our code in the form of a Python package, which is very suitable for embedding existing codes and applications. However, we believe that we are still inexperienced in developing and maintaining open-source code, so we will continue to humbly seek valuable advice.

Data and Resources

All the seismic records used in the examples were requested from the Data Management Center (DMC) of Incorporated Research Institutions for Seismology (IRIS) at https://ds.iris.edu/ds/nodes/dmc. Additionally, USArray information can be obtained from https://www.usarray.org and https://doi.org/10.7914/SN/TA. Detailed information about fftw-3 can be obtained at https://www.fftw.org. The CC-FJpy, manual and examples are available from https://github.com/ColinLii/CC-FJpy. We have also uploaded the Jupyter notebook files of examples in the supplemental materials. All the links mentioned are last accessed on February 4, 2021.

Acknowledgements

The open-source CC-FJpy is only for scientific research, teaching and other non-commercial use. For commercial use, please be sure to consult and obtain our consent (related patent no: US 10,739,482 B2). This work was supported by National Natural Science Foundation of China (Grants No. 41790465, U1901602 and 41974047), Key Special Project for Introduced Talents Team of Southern Marine Science and Engineering Guangdong Laboratory (Guanzhou) (GML2019ZD0203), Shenzhen Science and Technology Program (Grant No. KQTD20170810111725321), Shenzhen Key Laboratory of Deep Offshore Oil and Gas Exploration Technology (Grant No.ZDSYS20190902093007855) and the Leading Talents of Guangdong Province Program (Grant No. 2016LJ06N652)

References

Bensen, G. D., Ritzwoller, M. H., Barmin, M. P., Levshin, A. L., Lin, F., Moschetti, M. P., . . . Yang, Y. (2007). Processing seismic ambient noise data to obtain reliable broad-band surface wave dispersion measurements. Geophysical Journal International, 169(3), 1239-1260. doi:10.1111/j.1365-246X.2007.03374.x

Bensen, G. D., Ritzwoller, M. H., & Yang, Y. (2009). A 3-D shear velocity model of the crust and uppermost mantle beneath the United States from ambient seismic noise. Geophysical Journal International, 177(3), 1177-1196. doi:10.1111/j.1365-246X.2009.04125.x

Beyreuther, M., Barsch, R., Krischer, L., Megies, T., Behr, Y., & Wassermann, J. (2010). ObsPy: A Python Toolbox for Seismology. Seismological Research Letters, 81(3), 530-533.

doi:10.1785/gssrl.81.3.530

Campillo, M., & Paul, A. (2003). Long-range correlations in the diffuse seismic coda. Science, 299(5606), 547-549. doi:10.1126/science.1078551

Dreiling, Jennifer; Tilmann, Frederik (2019): BayHunter - McMC transdimensional Bayesian inversion of receiver functions and surface wave dispersion. GFZ Data Services. http://doi.org/10.5880/GFZ.2.4.2019.001

Fang, H., Yao, H., Zhang, H., Huang, Y.-C., & van der Hilst, R. D. (2015). Direct inversion of surface wave dispersion for three-dimensional shallow crustal structure based on ray tracing: methodology and application. Geophysical Journal International, 201(3), 1251-1263.

Fang, H., Zhang, H., Yao, H., Allam, A., Zigone, D., Ben-Zion, Y., . . . van der Hilst, R. D. (2016). A new algorithm for three-dimensional joint inversion of body wave and surface wave data and its application to the Southern California plate boundary region. Journal of Geophysical Research: Solid Earth, 121(5), 3557-3569.

Feng, J., Yao, H., Poli, P., Fang, L., Wu, Y., & Zhang, P. (2017). Depth variations of 410 km and 660 km discontinuities in eastern North China Craton revealed by ambient noise interferometry. Geophysical Research Letters, 44(16), 8328-8335. doi:https://doi.org/10.1002/2017GL074263

Forbriger, T. (2003). Inversion of shallow-seismic wavefields: I. Wavefield transformation. Geophysical Journal International, 153(3), 719-734. doi:10.1046/j.1365-246X.2003.01929.x

Hartog, A. H. (2017). An introduction to distributed optical fibre sensors. Boca Raton, Florida: CRC Press/Taylor and Francis. https://doi.org/10.1201/9781315119014

Hu, S., Luo, S., & Yao, H. (2020). The Frequency-Bessel Spectrograms of Multicomponent Cross-Correlation Functions From Seismic Ambient Noise. Journal of Geophysical Research: Solid Earth, 125(8), e2020JB019630. doi:https://doi.org/10.1029/2020JB019630

Lawrence, J. F., Denolle, M., Seats, K. J., & Prieto, G. A. (2013). A numeric evaluation of attenuation from ambient noise correlation functions. Journal of Geophysical Research: Solid Earth, 118(12), 6134-6145. doi:https://doi.org/10.1002/2012JB009513

Li, Z., & Chen, X. (2020). An Effective Method to Extract Overtones of Surface Wave From Array Seismic Records of Earthquake Events. Journal of Geophysical Research: Solid Earth, 125(3), e2019JB018511. doi:10.1029/2019jb018511

Li, Z., & Chen, X. (2020). Multiple dispersion curves extracted from seismic PL phase. Earth and Space Science Open Archive. doi: 10.1002/essoar.10505116.1

Mateeva, A., Lopez, J., Potters, H., Mestayer, J., Cox, B., Kiyashchenko, D., . . . Detomo, R.

(2014). Distributed acoustic sensing for reservoir monitoring with vertical seismic profiling. Geophysical Prospecting, 62(4), 679-692. doi:https://doi.org/10.1111/1365-2478.12116

Nolet, A. M. H. (1976). Higher modes and the determination of upper mantle structure: Drukkerij Elinkwijk.

Pan, L., Chen, X., Wang, J., Yang, Z., & Zhang, D. (2018). Sensitivity analysis of dispersion curves of Rayleigh waves with fundamental and higher modes. Geophysical Journal International, 216(2), 1276-1303.

Poli, P., Campillo, M., Pedersen, H., & Group, L. W. (2012). Body-wave imaging of Earth's mantle discontinuities from ambient seismic noise. Science, 338(6110), 1063-1065. Retrieved from https://science.sciencemag.org/content/sci/338/6110/1063.full.pdf

Ruckdeschel F. R. (1981) BASIC Scientific Subroutines, Vol. II.

Sabra, K. G., Gerstoft, P., Roux, P., Kuperman, W. A., & Fehler, M. C. (2005). Extracting time-domain Green's function estimates from ambient seismic noise. Geophysical Research Letters, 32(3).

Sabra, K. G., Gerstoft, P., Roux, P., Kuperman, W. A., & Fehler, M. C. (2005). Surface wave tomography from microseisms in Southern California. Geophysical Research Letters, 32(14).

Sager, K., Boehm, C., Ermert, L., Krischer, L., & Fichtner, A. (2020). Global-Scale Full-Waveform Ambient Noise Inversion. Journal of Geophysical Research: Solid Earth, 125(4), e2019JB018644. doi:https://doi.org/10.1029/2019JB018644

Sager, K., Ermert, L., Boehm, C., & Fichtner, A. (2017). Towards full waveform ambient noise inversion. Geophysical Journal International, 212(1), 566-590. doi:10.1093/gji/ggx429

Sánchez-Sesma, F. J., & Campillo, M. (2006). Retrieval of the Green's function from cross correlation: the canonical elastic problem. Bulletin of the Seismological Society of America, 96(3), 1182-1191. doi:10.1785/0120050181

Shapiro, N. M., Campillo, M., Stehly, L., & Ritzwoller, M. H. (2005). High-resolution surface-wave tomography from ambient seismic noise. Science, 307(5715), 1615-1618. doi:10.1126/science.1108339

Seats, K. J., Lawrence, J. F., & Prieto, G. A. (2012). Improved ambient noise correlation functions using Welch′s method. Geophysical Journal International, 188(2), 513-523. doi:10.1111/j.1365-246X.2011.05263.x

Shen, W., & Ritzwoller, M. H. (2016). Crustal and uppermost mantle structure beneath the United States. Journal of Geophysical Research: Solid Earth, 121(6), 4306-4342.

doi:10.1002/2016JB012887

Shen, W., Ritzwoller, M. H., Schulte-Pelkum, V., & Lin, F.-C. (2012). Joint inversion of surface wave dispersion and receiver functions: a Bayesian Monte-Carlo approach. Geophysical Journal International, 192(2), 807-836.

Sergi Ventosa, Martin Schimmel, Eleonore Stutzmann; Towards the Processing of Large Data Volumes with Phase Cross-Correlation. Seismological Research Letters 2019;; 90 (4): 1663–1669. doi: https://doi.org/10.1785/0220190022

Wang, J., Wu, G., & Chen, X. (2019). Frequency-Bessel Transform Method for Effective Imaging of Higher-Mode Rayleigh Dispersion Curves From Ambient Seismic Noise Data. Journal of Geophysical Research: Solid Earth, 124(4), 3708-3723. doi:10.1029/2018jb016595

Wang, K., Liu, Q., & Yang, Y. (2019). Three-Dimensional Sensitivity Kernels for Multicomponent Empirical Green's Functions From Ambient Noise: Methodology and Application to Adjoint Tomography. Journal of Geophysical Research: Solid Earth, 124(6), 5794-5810. doi:https://doi.org/10.1029/2018JB017020

Weaver, R. L., & Lobkis, O. I. (2004). Diffuse fields in open systems and the emergence of the Green’s function (L). The Journal of the Acoustical Society of America, 116(5), 2731-2734. doi:10.1121/1.1810232

Wu, G.-x., Pan, L., Wang, J.-n., & Chen, X. (2020). Shear Velocity Inversion Using Multimodal Dispersion Curves From Ambient Seismic Noise Data of USArray Transportable Array. Journal of Geophysical Research: Solid Earth, 125(1), e2019JB018213. doi:10.1029/2019jb018213

Yao, H., van Der Hilst, R. D., & De Hoop, M. V. (2006). Surface-wave array tomography in SE Tibet from ambient seismic noise and two-station analysis—I. Phase velocity maps. Geophysical Journal International, 166(2), 732-744. doi:10.1111/j.1365-246X.2006.03028.x

Yokoi, T. (2010). New formulas derived from seismic interferometry to simulate phase velocity estimates from correlation methods using microtremorFormulas for phase velocity estimates. Geophysics, 75(4), SA71-SA83.

Zhan W., Pan L., Chen X.F., (2020). “3D crustal shear velocity structure of China Northeast constrained from inversion of multimodal dispersion curves of Rayleigh waves from ambient seismic noise”, J. Asian Earth Sci., 196, July, doi.org/10.1016/j.jseaes.2020.104372

Figures

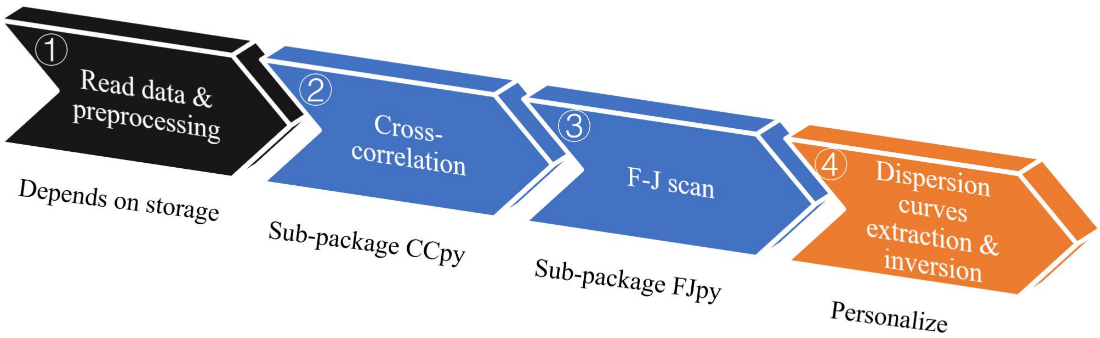

Figure 1. The imaging process through the F-J method. At present, CC-FJpy mainly contains parts ② and ③. We will add ④ to this package in future updates

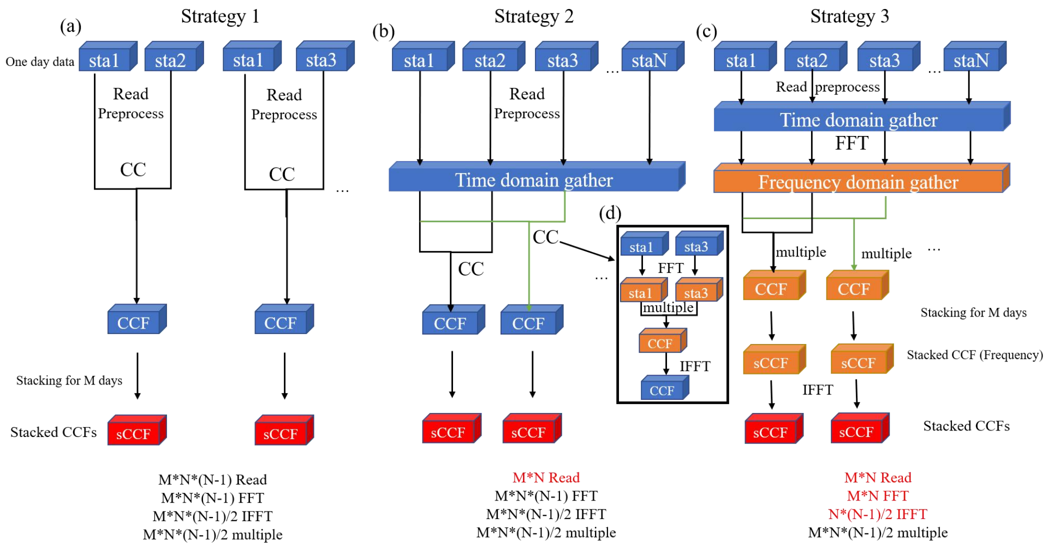

Figure 2. Comparison of the three CC strategies for the continuous records of *M* days and *N* stations (sta1, sta2, …, staN) in units of days. The blue squares are the data in the time domain, the yellow squares are the data in the frequency domain, and the red squares are the final outputs. **(a)** Completely independent cross-correlation: in this approach, each CC reads data and correlates independently. **(b)** Shared memory strategy: in this approach, *N* records of each day are read, shared and then cross-correlated. **(c)** Shared reading and FFT strategy: not only are the memories of N records shared the FFTs and IFFTs are also shared. **(d)** How a single CC is implemented. There is a huge gap in the required calculation between the three strategies. We have marked the number of core calculations for CC below the flowchart of each strategy.

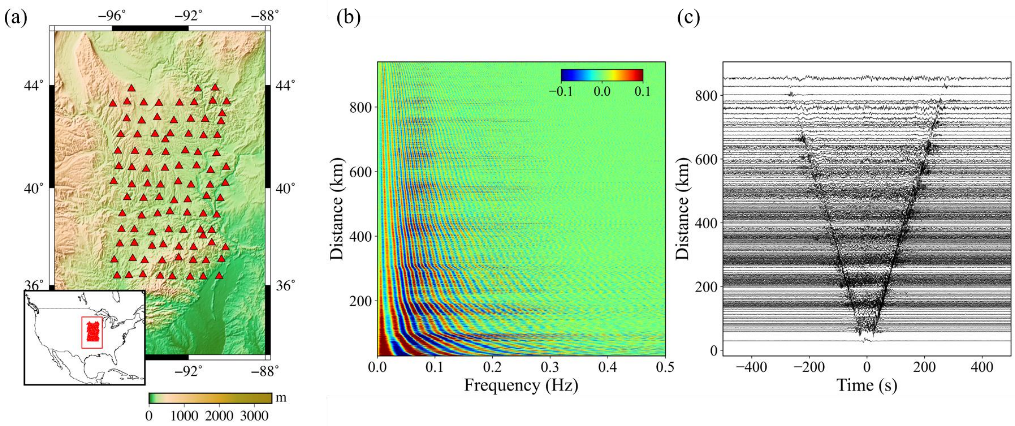

Figure 3. The CCFs recovered from USArray ambient noise data. (a) Stations used. (b) CCFs in the frequency domain recovered by CC-FJpy. (c) Time domain CCFs obtained by the IFFT of the frequency-domain CCFs.

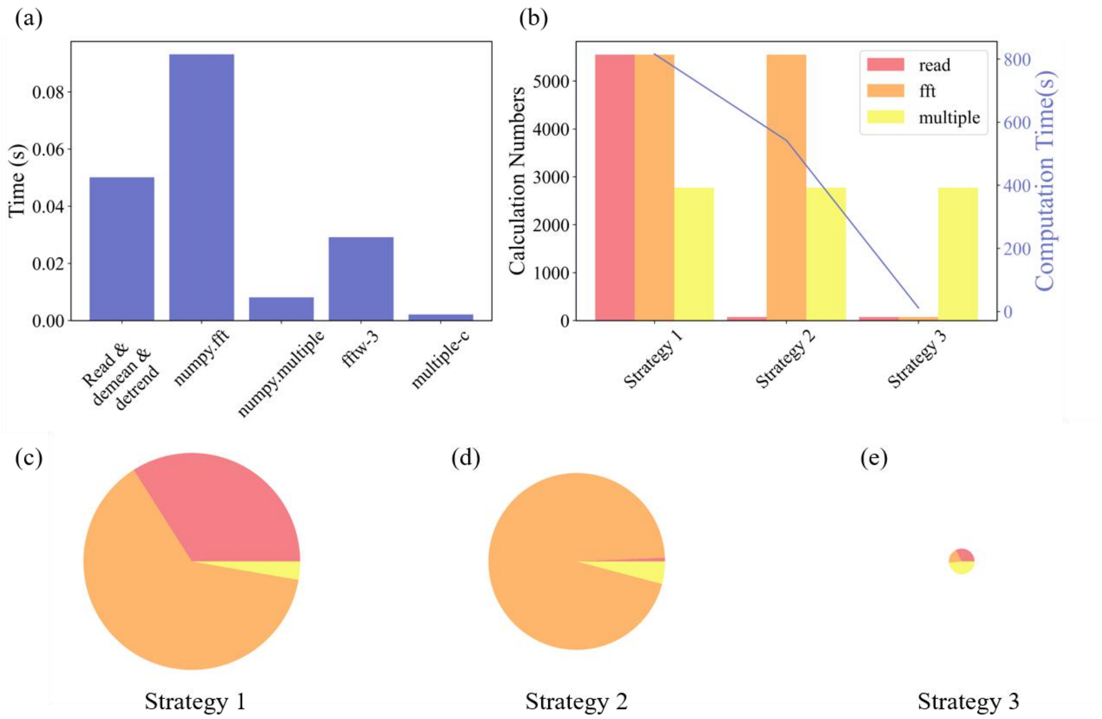

Figure 4. Cross-correlation performance comparison. It should be noted that all the comparisons here are in the case of a serial approach. **(a)** The time taken to read and preprocess data, FFT by numpy, multiplication by Python, FFT by fftw-3 and multiplication by C language for one station per day data. It should be noted that the FFT times and multiple times refer to the sum time of FFT and multiple in the overlap. **(b)** Comparison of the calculation amounts of different CC strategies. **(c), (d), and (e)** Proportion of time consumed by the main operations for the different strategies. The area of the pie chart is proportional to the total time.

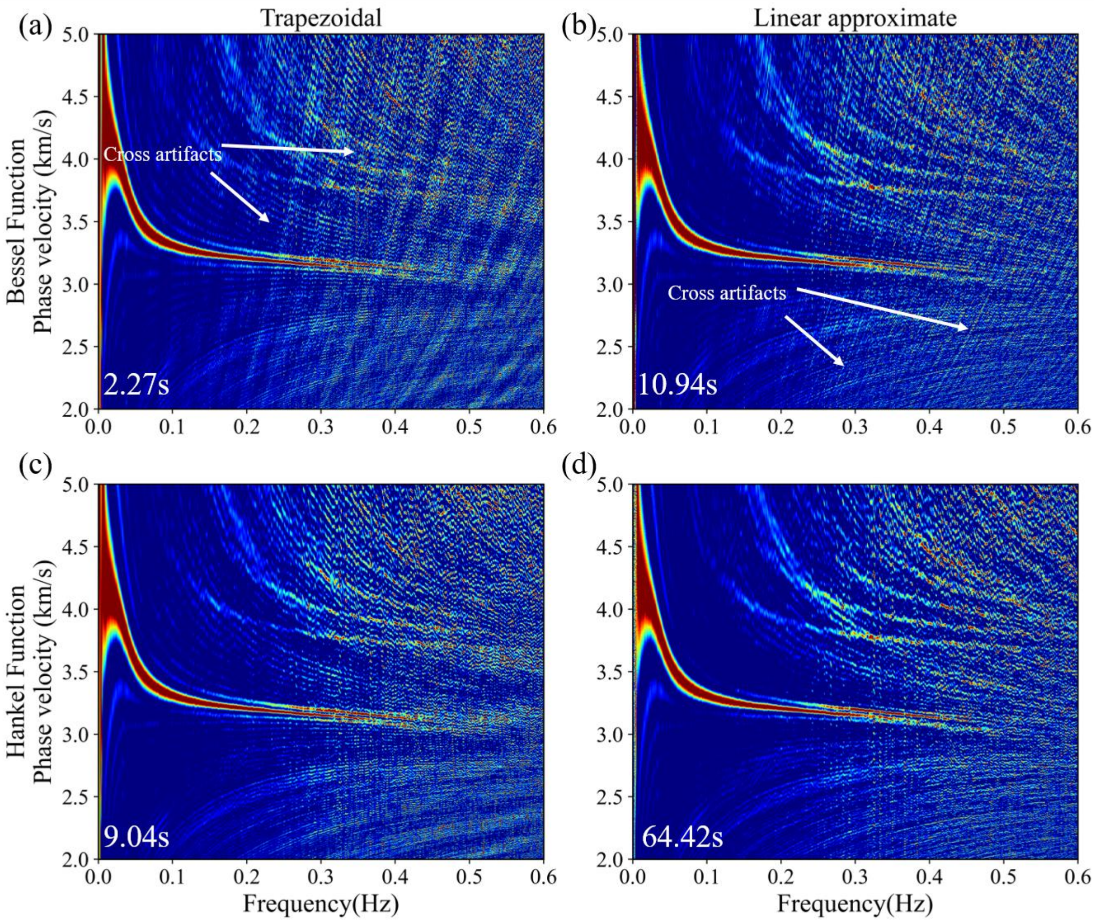

Figure 5. Dispersion spectra extracted from CCFs by FJpy. **(a)** Dispersion spectrum calculated through the trapezoidal integral of the Bessel function. **(b)** Dispersion spectrum calculated through equation 6. **(c)** Dispersion spectrum calculated through the trapezoidal integral of the Hankel function. **(d)** Dispersion spectrum calculated though equation 6 with the Hankel function. The times in the lower left corner of each subfigure are the calculation times after acceleration by the GPU.

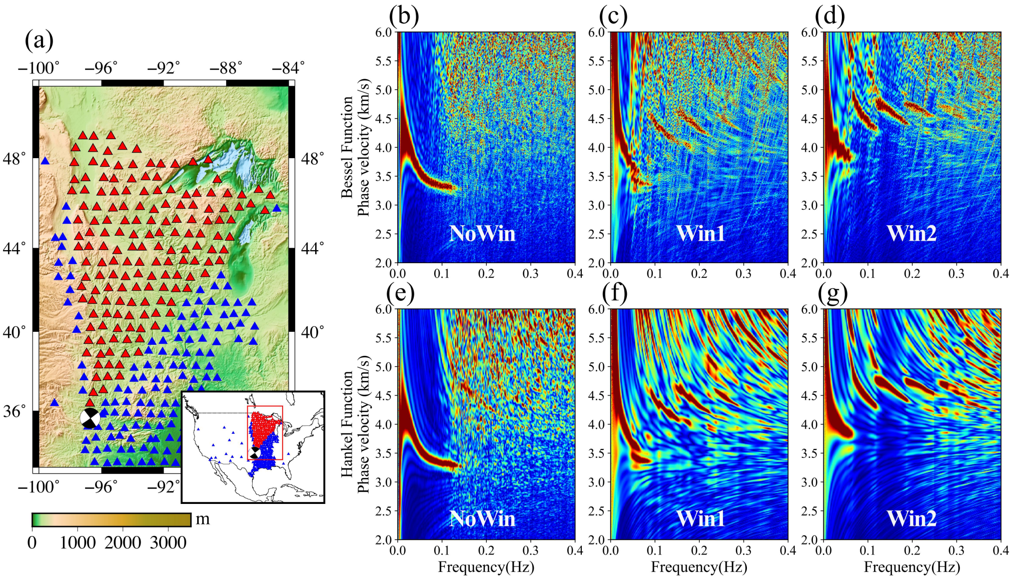

Figure 6. Dispersion spectra extracted from seismic records by FJpy. **(a), (b), and (c)** Dispersion spectra extracted by equation 6 with NoWin, Win1 and Win2. **(d), (e), and (f)** Dispersion spectra extracted by equation 6 with the Hankel function.