Inputs

The inputs for the conversion script are acceleration recordings. Simulated recordings can be downloaded from SeisFinder as explained in the instructions.

.000 and .090 files are perpendicular recordings of the horizontal motion.
.ver files are recordings of the vertical motion. They can be converted into audio but the ground motion graph is designed to compare .000 and .090 files. Generally the horizontal shaking is also larger than the vertical shaking.

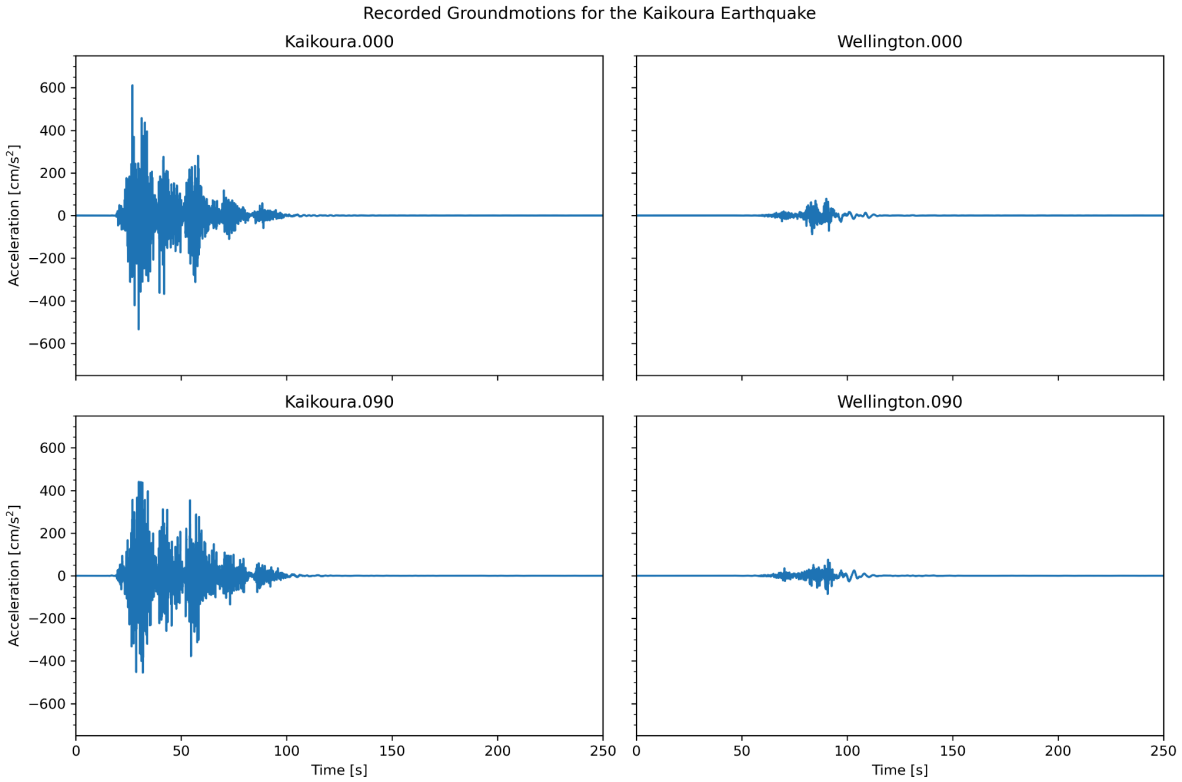
Files downloaded from SeisFinder are in units of cm/s2. Actual recordings of earthquakes generally use units of gravity. The script has a built-in converter for g to cm/s2 so historic earthquakes are still easy to convert to audio.

The script is calibrated based on the 2011 Christchurch earthquake recorded at the HVSC station (HVSC.090). This was the largest ground motion tested during development. If the input ground motion is too large to be accurately represented the script will print a message to that effect. The impact on the output audio is explained in the Process section.

Outputs

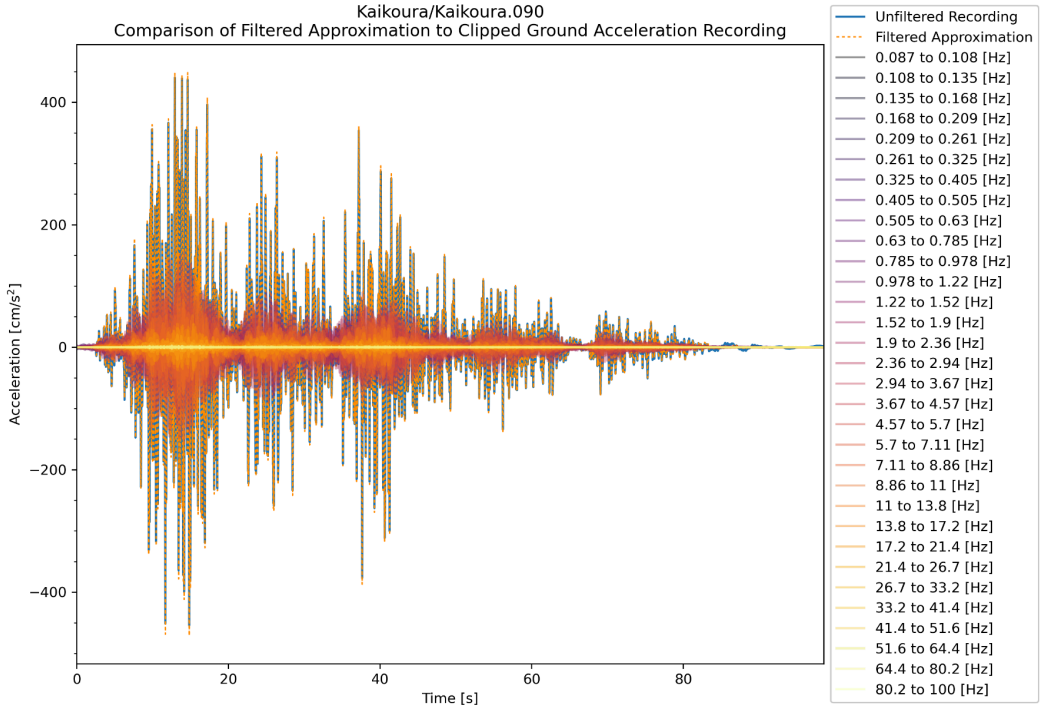
Below are examples of each of the outputs from the conversion script in order of creation. They are titled as the actual outputs will be, with {} as placeholders for the actual information. There is an explanation of what the output is and a description of what it is useful for.

{earthquake}.png



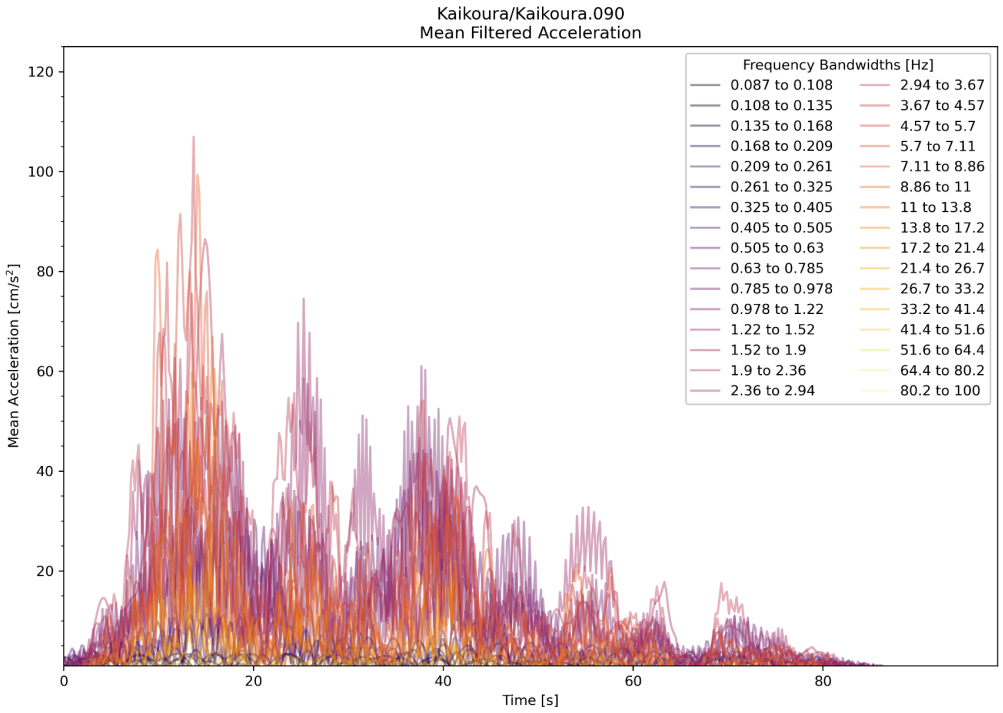
The script saves a .png of the above graph to the eq2audio folder for each earthquake/folder of recordings that it processes. Each graph should have the .000 recordings in the upper row and the corresponding .090 recordings in the bottom row. If you are looking at a lot of locations for the same earthquake the formatting may be lost.
This allows you to compare the .000 and .090 recordings across different recording locations for the same earthquake. This makes it clear where the most intense shaking was for the earthquake. The y-axis is calibrated from the HVSC.090 recording so that it is easy to see if each recording is relatively large or small.

{recording} Comparison of Filtered Approximation to Clipped Ground Motion Recording.png



The script saves a .png of the above graph to the respective earthquake folder for each recording that it processes. Each graph shows the original ground motion recording automatically clipped to the time span where it is most intense. This is overlaid with the sum of all the bandwidth filtered waves as well as every individual bandwidth filtered wave. This allows you to check that the bandwidth filtering is a good approximation of the actual recorded ground motion. The approximation (dashed orange line) generally follows the actual recording (blue line) pretty well through the main shaking. However, there can be some divergence at the lead-in/tail of the pictured time span, and at the peak of each spike, as shown above. The y-axis is fitted to the recording being shown so that smaller recordings can still be inspected. Because it also shows the filtered waves this can also indicate whether the recording is mostly low or high frequency (indigo and yellow respectively).

{recording} Mean Filtered Accelerations.png



The script saves a .png of the above graph to the respective earthquake folder for each recording that it processes. Each graph shows the mean acceleration of the filtered waves, which is the data used for the conversion to audio.
This means you can visually assess which parts of recordings are likely to sound interesting, which is very useful if you are deciding what time span to animate. The y-axis is calibrated from the HVSC.090 recording so you can tell whether the audio recording will be loud or quiet (mostly high or low accelerations displayed). This graph is also very useful for seeing whether the earthquake is mostly low or high frequency.

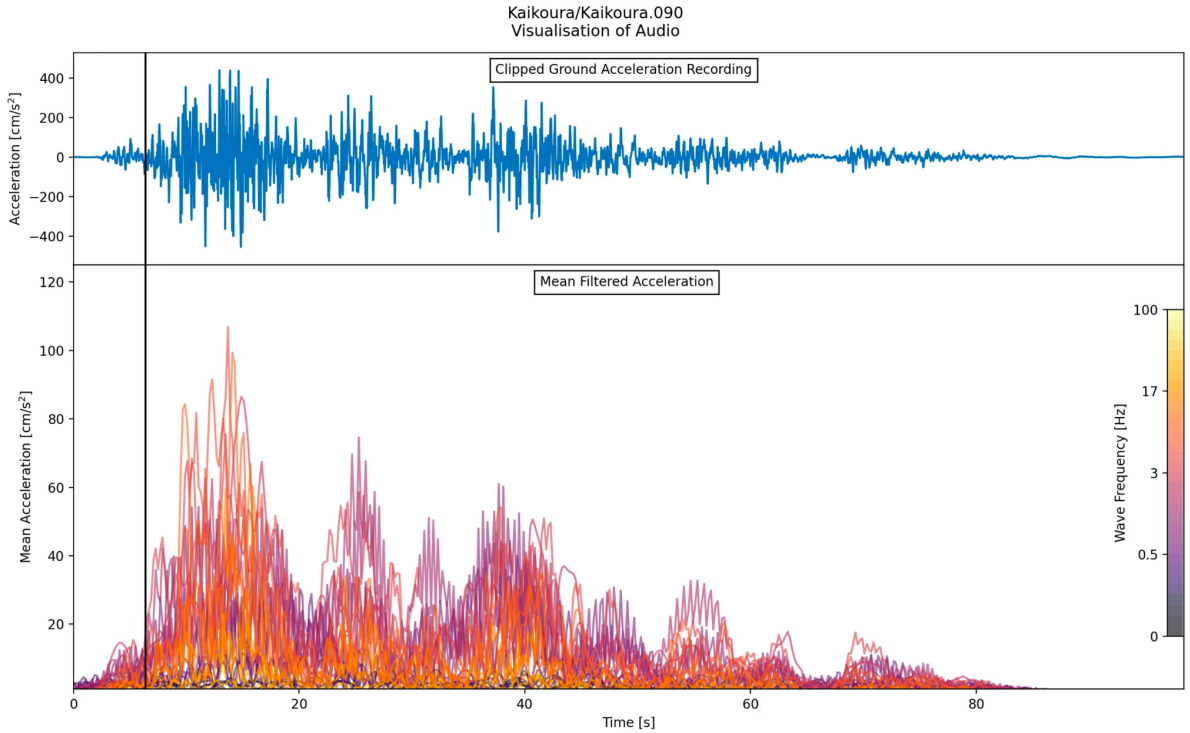
{recording}.mid

The script saves a .mid MIDI file to the respective earthquake folder for each recording that it processes. This file contains all the information for the converted audio recording. MIDI files will sound slightly different depending on what audio processing software you open them with as different software uses different soundfonts. However, the information is the same. This file could be manually edited using software, but this would mean it is not comparable to other recordings produced by the conversion script.

{recording}.flac

The script saves a .flac audio file to the respective earthquake folder for each recording that it processes. FLAC is a lossless audio format that can be opened with most media players. It can be converted to other audio formats by changing the file extension (the part after the “.”) or using software.
This is it! The audio version of the earthquake. You can use this to demonstrate features of an earthquake (see here for some interesting features), or just to represent the earthquake in a new way.

{recording} animation ({start time} to {end time}).mp4



The script saves a .mp4 video file to the respective earthquake folder for each recording that it processes. A screenshot of the animation is shown above. The top graph is the inputted ground acceleration recording clipped to the time span of interest. The bottom graph is the mean filtered accelerations graph, with a gradient key for the wave frequencies instead of showing the discrete frequency bands. A vertical black line moves across both the graphs with time.

This animation allows you to easily relate the audio generated by the script to the ground motion recording you chose as the input. At the same time you can follow the progress of the audio using the mean filtered accelerations graph so you can tell when the loudest parts of the audio recording will be and whether high or low pitches will be the loudest.

Process

This talks through the process that the script follows. While the code is reasonably well commented this may be more helpful if you are wondering why something was done.

Setting key variables

Base (default: 1 cm/s2): This is the minimum ground acceleration that registers as an earthquake. It is used to clip the duration of the ground motion recording, and to silence small motions to reduce the drone effect in the audio. It sets the bottom of the note length and velocity interpolation ranges.

Top (default: 125 cm/s2): This is the maximum mean ground acceleration that can be accurately displayed as sound. It sets the top of the interpolation ranges so when frequency bands have mean values above the top value those set to the top value. The default was set based on simulation acceleration recordings. Real-life recordings can be significantly above the top value due to site effects, but this is uncommon.

Block (default: 0.2 s): This is the time span that the frequency bands get their acceleration averaged over. If the sampling time is 0.005 this means the mean of 40 data points is the value used to interpolate the note length and velocity.

Bands: This generates frequencies between 0 and 100 Hz to use as the centre values for bandwidth filtering. They are generated so that the bands they create in the butterworth filter border each other without overlapping, so there end up being 32 frequency bands. This makes sure no frequencies are missed and that there are no double-ups. Because the standard sampling time is 0.005 seconds the nyquist frequency is 100 Hz. This determines the upper limit of the frequency bands that can be used in the butterworth bandwidth filtering.

Pent (default: 34, or Bb1): This generates the pitches for a 32 step (6 octaves and 1 note) pentatonic scale in MIDI notation starting on a given pitch. This means each frequency band can be assigned to a discrete pitch. The pentatonic scale was chosen as it doesn't sound terrible when all the notes are played, unlike other scales/modes. In order to fit all the pitches in it needs to start pretty low.

Looping through each earthquake

First the data is extracted from the input files. This is done using an extract of the sample code on the SeisFinder website. The code is copied directly from the "retrieve data" section on this page.
Immediately after this the data is converted from gravity to cm/s2 if required.

The raw data is then plotted for each earthquake (see {earthquake}.png for details).

Looping through each recording for the current earthquake

First the data is checked to see if every entry is smaller than the base value. If it is smaller the script will immediately go on to the next recording, skipping the rest of this iteration of the loop. This is mostly relevant if data in units of g is not converted as typically at least one acceleration data point in the recording will be larger than the base value.

If any data is greater than the base value the recording is filtered. The first and last times the acceleration is greater than the base value are located, and all data outside is discarded. If the remaining data is still greater than some time span (default: 3 minutes) it is then clipped to that length, discarding data after that period. It is assumed that the earthquake event will begin reasonably soon after the acceleration exceeds the base value and then the shaking will take a while to trail off. The data is then trimmed to the base value again to tidy up the end.

Looping through each of the 32 filter frequencies

The data is run through a butterworth filter using the selected filter frequency as the centre value for bandwidth filtering. The butterworth filter code is copied directly from the QuakeCoRE GitHub library, the only change made was adding the edge frequencies of each band to the returns. The filt_vals array stores the filtered values for the bandwidth, the filt_bands array is the outer frequencies of the bandwidth, and filt_sum keeps a running total of all the filtered values.

End of filter frequency loop

The filtered outputs are then plotted (see {recording} Comparison of Filtered Approximation to Clipped Ground Motion Recording.png for details).

The number of datapoints in each time block is determined based on the sampling time for the recording.

The mean of the absolute data values for each wave is then taken over that span of data points. This means that generally the mean is taken every 40 data points.

On low frequency waves the period is larger than the set time block so the shape of the wave is maintained through the mean values. For high frequency waves (above 5 Hz on default settings, less than half the waves) the mean includes troughs as well as peaks, which reduces the mean value. This could result in the audio under-representing the higher frequencies.

In development some audio files were made by taking the maximum value in each span of data points to test this. The audio produced was very similar in terms of pitches emphasised, which means that the high frequency waves were not being majorly underrepresented by taking the mean. However, using only the maximum values decreased the rhythmic nuance so it was decided that it would be better to use the mean and have more interesting audio.

Because the width of the band filtered by the butterworth function is proportional to the frequency, higher frequencies have larger bandwidths. This means there are a lot more filtered waves at lower frequencies, which could cause lower frequencies to have smaller amplitudes which may balance some of the effects from taking the mean.

The resulting means are plotted to provide a visualisation of what is actually being converted to audio (see {recording} Mean Filtered Accelerations.png for details).

Before audio is created the interpolation functions are set up for note velocity and length. The interpolation functions have velocity and length as y(x) where x is the mean acceleration. This means feeding in a mean acceleration returns a velocity or length.

Velocity is how hard each note is hit, so it affects the volume and the onset tone. The default limits for velocity are 20 and 120. 0 velocity means the note doesn't play and 127 velocity sounds quite aggressive.

The velocity array is created as a logarithmic function starting at minimum velocity. It is then clipped so that it only goes from the min to max values. A linear array of sample

accelerations from the base to top values is mapped to the logarithmic velocities. This creates the interpolation function that is used later.
Length is how many ticks the note is held for. The tempo (default: 120 bpm) controls how fast the ticks pass. The default limits for length are 40 and the block length in ticks (default: 192). It converts from seconds (block length) to ticks (note length) using the tempo in beats/minute, and the MIDI standard of 480 ticks/beat. 40 ticks means the short notes are fast without having too much distortion.
The length array is created as a linear function starting at the minimum length. It is then clipped so that it only goes from the min to max values. A linear array of sample accelerations from the base to top values is mapped to the linear lengths. This creates the interpolation function that is used later.

A MIDI file is created to add tracks to. The filename can be customised by adding a string so that if you want to compare audio from the same recording it won't overwrite it each time.
Looping through each of the 32 pentatonic notes
A MIDI track is created to add messages to. Each track will hold messages for one frequency bandwidth, so all notes will use the same pitch. The track name is set to the filename plus the pitch of the track. The tempo for the track is set (default: 120 bpm).

Most MIDI controls seem to affect everything on whatever channel they are sent on. Since there are 15 usable MIDI channels (channel 10 is only percussion) and 32 frequency bands there aren't enough channels for each pitch to get its own channel. Because of this the channel changes every two pitches (except the last channel which gets four pitches).

When the channel is selected the instrument and pan location for the channel is also selected. The instrument is selected from a set that was chosen after experimenting with different General MIDI instrument sounds. The pan is randomly selected from an array ranging from 4 (fully left is 0) to 124 (fully right is 127).

Looping through each of the mean accelerations
The current acceleration is selected from the averages dictionary. If it is smaller than the base value the note is made silent by setting the velocity to 0. If it is between the base and top values the velocity and length are interpolated using the functions previously set up. If it is larger than the top value the velocity and length are both set to the maximum values, and big is set to True to record that one of the mean values was larger than the top value.

Now that the velocity and length have been set notes can start being written for the time block. There is an initial delay proportional to the note length (default: 10%) before the notes start playing. This is to stop all the pitches from playing at exactly the same time at the beginning of each time block, which sounded very clunky.
The first note is only 90% of the note length due to the delay. While there is enough room in the time block it will write a rest and then a note, both with the same length. Once there isn't enough room in the time block for this pair a rest is held until the end of the time block.
So in each time block: 10% rest, 90% note, (100% rest, 100% note repeated while there is space), rest until the block is full.
End of mean acceleration loop
End of pentatonic note loop

If big is True (any of the mean accelerations were above top) a message is printed and the file name is edited.
Now the MIDI file is saved since every mean acceleration for every filter band has been used to make audio.

FluidSynth is called to convert the MIDI file to audio. On Mac this sometimes prints two panic messages so before it is called a message is printed telling the user not to worry about them.
The default settings eq2audio uses for FluidSynth are as follows:
-o synth.reverb.width=40 sets the stereo width of the reverb.
-o synth.reverb.room-size=0.4 sets the room-size for the reverb. Without reverb all the notes sound very flat/1-Dimensional, but with too much reverb it sounds very echoey.
-ni deactivates both MIDI input and the shell so that FluidSynth quits once it has completed the conversion.
-q means that FluidSynth doesn't print any regular messages like the intro.
-g 0.5 sets the gain for the synthesiser. Increasing the gain increases the volume of the audio, but if it is too large there will be lots of distortion on the loud notes.
MuseScore_General.sf2 tells FluidSynth what soundfont to use to make the sounds for each instrument and pitch.
{MIDI file name}.mid -F {audio file name}.flac tells FluidSynth to do a fast render from MIDI to raw audio data. This is what actually tells it to convert the file. The file format is determined by the audio file name extension.

If the user wants an animation made this is the last thing the script does for each recording (see {recording} animation ({start time} to {end time}).mp4 for details).

End of recording loop

End of earthquake loop

Possible Next Steps

If I make any major improvements I will release the new version of eq2audio.py on GitHub.

Instead of taking the mean over some block of time to determine note lengths and velocities these could be set at the beginning of the recording and then change based on the gradient of the wave. This may result in more interesting audio that more accurately represents the shape of the waves.

To make converting files easier there could be some sort of user interface, a file browser for selecting files, etc.