Using signal lag to find the distance between two sensors

(Dated: 5th September 2020)

I. BACKGROUND

Two sensors located a distance $x\,\text{m}$ apart sample the same sound signal at a rate of $f = 44.1\,\text{kHz}$. Sensor one outputs a vector $\mathbf{f}$ and sensor two outputs a vector $\mathbf{g}$.

To find x, we must find where $\mathbf{f}$ and $\mathbf{g}$ overlap which gives us the lag between samples being captured (Δp), and thus the time between the two sensors Δp. Given the sampling rate is constant and the speed of sounds is a known constant ($333\,\text{m.s}^{-1}$), we have:

$$x = \Delta p\,[\text{pnts}] \times \frac{1}{f}\,[\text{s/pnts}] \times 333\,\left[\text{m.s}^{-1}\right]$$
$$= \frac{333}{44100} \times \Delta p\,[\text{s}]. \qquad (1)$$

To get Δp we sample each signal, normalise and then take the cross-correlation of each signal. the maximum cross-correlated value will reveal where the signals are aligned, and thus yield Δp.

The normalised cross correlation $cc_{f,g}$ between $\mathbf{f}$ and $\mathbf{g}$ is given by:

$$cc_{f,g} = \frac{\sum_{i=0}^{n} f_i \cdot g_i}{\sqrt{\sum_{i=0}^{n} f_i^2 \cdot \sum_{i=0}^{n} g_i^2}}, \qquad (2)$$

where $f_i\,\epsilon\,\mathbf{f}\;\forall\,i\,\epsilon\{0,1,\ldots n_f\}$. Here n_f is the length of $\mathbf{f}$. Similarly $g_i\,\epsilon\,\mathbf{g}\;\forall\,i\,\epsilon\{0,1,\ldots n_g\}$ and n_g is the length of $\mathbf{g}$, where $n = n_f = n_g$.

II. APPROACH

Both $\mathbf{f}$ and $\mathbf{g}$ are large, with $n_g = n_f = 176,401$, thus brute force methods were ruled out on my early 2015 *MacBook Air*. Specifically I could not sample $\mathbf{f_s}$ (where $\mathbf{f_s} \subset \mathbf{f}$) and calculate the cross correlation $cc_{f_s,g}$ for the full length of $\mathbf{g}$.

Additionally while visual inspection of $\mathbf{f}$ and $\mathbf{g}$ show the signal is not repetitive, I could not assume local repetition was completely absent from $\mathbf{f}$ (and $\mathbf{g}$). The question is thus: how to choose the right place to sample?

In the next paragraphs, I'll highlight sections of the code which are relevant in computing Δp. In NumPy a vector, say $\mathbf{f}$ has both values f_i and an index i I use to calculate Δp.

To start I took a random sample (without replacement) of 0.5% of indices in (`s1_sample`) in `s1` ($\mathbf{f}$). A random sample was chosen to confirm Δp at many points in case there's any local repetition.

```
s1_sample=np.random.choice(
          range(s1.shape[0]),
          floor(s1.shape[0]
               *0.005),
          replace=False).
```

Where `floor(s1.shape[0]*0.005)` calculates the size of a 0.5% sample of `s1` ($\mathbf{f}$). A 0.5%

was chosen to minimise computational time, while providing a big enough sample to yield reliable results.

With a random sample of indices in `s1`, I look in `s2` where `s2[i]=s1[j]` ($\mathbf{g_i} = \mathbf{f_j}$):

```
s1_sample[
    np.array(
        [s1[s] in s2
         for s in s1_sample]
        )].
```

Knowing the indices where `s1=s2` I sample 100 points in this vicinity (using the function `TakeTimeSnapshot`) and calculate the normalised cross correlation (with the function `CalcNormedCrossCorr`, which yields `cc_s1_s2`). I extract the location of all maxima (`ccmax`) in the cross correlated result and use this to find the corresponding indices in `s1` (`ind_s1_match[cc_s1_s2==ccmax]`) and `s2` (`ind_s2_match[cc_s1_s2==ccmax]`). Subtracting these values gives me the lag:

```
lag=ind_s2_match[cc_s1_s2==ccmax] -
    ind_s1_match[cc_s1_s2==ccmax]
```

III. RESULTS

One run using the method above calculated a lag of 50, 082 s, which corresponds to 378.17 m between sensor 1 and sensor 2, and is confirmed at 613 points in the sample (or 0.35% of the sample population). This result took 1.62 s (but could take up to 1.93 s).

A. Improvements

Visual inspection of the printed logs reveals the most expensive operation is reading in each file. In addition to experimenting with new **`read`** functions I could have read in only the start, or end of one signal file, say $\mathbf{f}$, then applied my method as usual, requiring me to only read in one file in full ($\mathbf{g}$). I would estimate my run time would be reduced by at most half. This suggestion only works if the vectors are of the same length and thus the files must overlap at the start or end.

Finally, taking a smaller sample (`sample_s1`) will reduce the run time.