Step-by-step NMO correction

Leonardo Uieda[1]

Open any textbook about seismic data processing and you will inevitably find a section about the normal moveout (NMO) correction. There you'll see that we can correct the measured traveltime of a reflected wave t at a given offset x to obtain the traveltime at normal incidence t_0 by applying the following equation:

$$t_0^2 = t^2 - \frac{x^2}{v_{\text{NMO}}^2}, \quad (1)$$

in which v_{NMO} is the NMO velocity. There are variants of this equation with different degrees of accuracy, but we'll use this one for simplicity.

When applied to a common-midpoint (CMP) section, the equation above is supposed to turn the hyperbola associated with a reflection into a straight horizontal line. What most textbooks won't tell you is *how, exactly, to apply this equation to the data*.

Read on and I'll explain step-by-step how the algorithm for NMO correction from Yilmaz (2001) works and how to implement it in Python. The accompanying Jupyter notebook (Perez and Granger, 2007) contains the full source code, with documentation and tests for each function. You can download the notebook at https://github.com/seg or at https://github.com/pinga-lab/nmo-tutorial.

What the equation doesn't tell us

Equation 1 relates traveltimes: the one we can measure (t) and the one we want to know (t_0). But the data in our CMP gather are actually a matrix of *amplitudes* measured as a function of time (t) and offset. Our NMO-corrected gather will also be a matrix of amplitudes as a function of time (t_0) and offset. So what we really have to do is transform one matrix of amplitudes into the other. *But the equation has no amplitudes!*

This is a major divide between the formula we've all seen before and what actually goes on in the software that implements it. You have probably never thought about it — I certainly hadn't — so let's bridge this divide. Next, I'll explain an algorithm that maps the amplitudes in the CMP to amplitudes in an NMO-corrected gather.

Doing it backwards

It's surprisingly difficult to find a description of a method for calculating the amplitudes in the NMO correction. The only one I could find is a single paragraph in the book by Yilmaz (2001) (available on the SEG Wiki at http://wiki.seg.org/wiki/NMO_for_a_flat_reflector):

"The idea is to find the amplitude value at A′ on the NMO-corrected gather from the amplitude value at A on the original CMP gather. Given quantities t_0, x, and v_{NMO}, compute t from equation 1. [...] The amplitude value at this time can be computed using the amplitudes at the neighboring integer sample values [...] This is done by an interpolation scheme that involves the four samples."

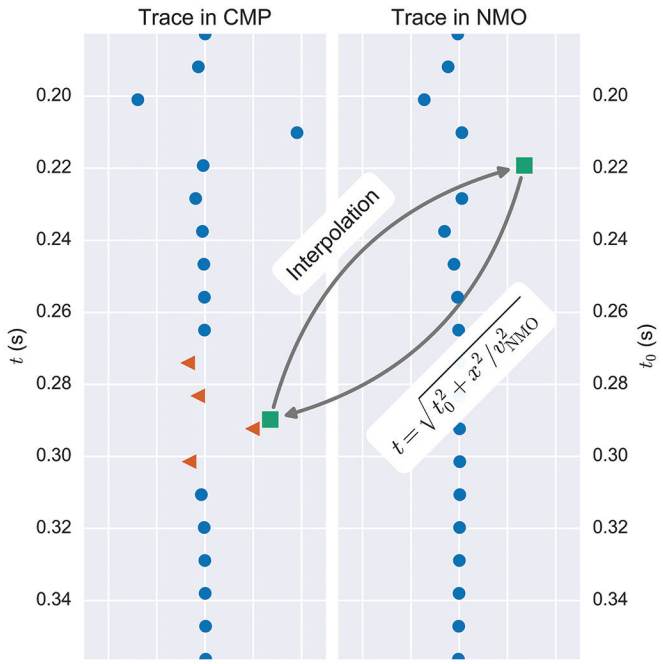

Figure 1. Sketch of the algorithm for a single trace and t_0. To the left is a trace from the CMP and to the right the corresponding trace from the NMO-corrected gather. The green square in the NMO is the amplitude at t_0 that we want to calculate. We apply the equation to find time t in the CMP, then interpolate the amplitude using the four samples around t (orange triangles). This amplitude is then copied over to the NMO.

This paragraph is telling us to do the calculation backwards. Instead of mapping where each point in the CMP goes in the NMO-corrected gather, we should map where each point in the NMO gather comes from in the CMP. Figure 1 shows a sketch of the procedure to calculate the amplitude of a point (t_0, x) in the NMO gather.

Here is the full algorithm:

1) Start with an NMO gather filled with zeros.
2) For each point (t_0, x) in the NMO gather:
   a) Calculate the reflection traveltime (t) given v_{NMO} using the equation in Figure 1.
   b) Go to the trace at offset x in the CMP and find the two samples before and the two samples after time t.
   c) If t is greater than the recording time or if it doesn't have two samples after it, skip the next two steps.
   d) Use the amplitude in these four samples to interpolate the amplitude at time t.
   e) Copy the interpolated amplitude to the NMO gather at (t_0, x).

At the end of this algorithm, we will have a fully populated NMO gather with the amplitudes sampled from the CMP. Notice that we didn't actually use the equation for t_0. Instead

[1]Universidade do Estado do Rio de Janeiro, Brazil.

http://dx.doi.org/10.1190/tle360200179.1.

we calculate the reflection traveltime (t). Good luck guessing that from the equation alone.

The code for NMO

Now I'll show how to implement the above algorithm in Python using the NumPy and SciPy libraries (van der Walt et al., 2011). We'll split the algorithm into three functions. This is very important when programming any moderately complex code because it allows us to test each part of our code independently. It also reduces the amount of code we have to search through to find the bug that is messing up our results. Modular code is easier to understand and to reuse.

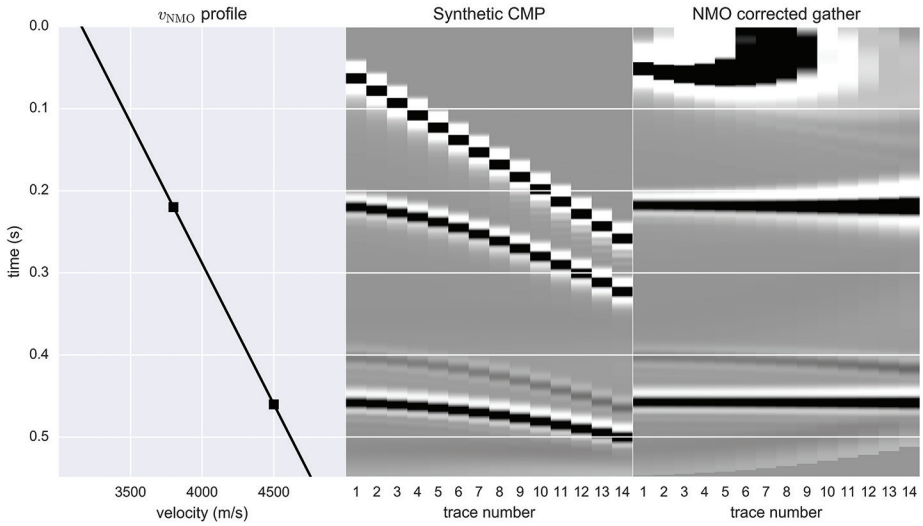

Figure 2. (a) The v_{NMO} profile passed to `nmo_correction`. The profile was interpolated on a line using the two picked velocities (black squares). (b) A synthetic CMP gather. (c) The output from our `nmo_correction` function.

The first function I'll define performs the NMO correction on a given CMP gather. We'll assume that the CMP gather is a 2D array of amplitudes and that the velocities are a 1D array with v_{NMO} for each time sample.

```
import numpy as np

def nmo_correction(cmp, dt, offsets, velocities):
    nmo = np.zeros_like(cmp)
    nsamples = cmp.shape[0]
    times = np.arange(0, nsamples*dt, dt)
    for i, t0 in enumerate(times):
        for j, x in enumerate(offsets):
            t = reflection_time(t0, x, velocities[i])
            amplitude = sample_trace(cmp[:, j], t, dt)
            if amplitude is not None:
                nmo[i, j] = amplitude
    return nmo
```

This function is essentially the algorithm above translated to Python with some of the details pushed into the `reflection_time` and `sample_trace` functions, which we will define below.

First, the function that calculates the reflection traveltime:

```
def reflection_time(t0, x, vnmo):
    t = np.sqrt(t0**2 + x**2/vnmo**2)
    return t
```

For the `sample_trace` function, we'll use cubic splines from the `scipy.interpolate` package. For more information on interpolation with scipy, see the tutorial by Hall (2016).

```
from scipy.interpolate import CubicSpline

def sample_trace(trace, time, dt):
    before = int(np.floor(time/dt))
    N = trace.size
    samples = np.arange(before - 1, before + 3)
    if any(samples < 0) or any(samples >= N):
        amplitude = None
    else:
        times = dt*samples
        amps = trace[samples]
        interpolator = CubicSpline(times, amps)
        amplitude = interpolator(time)
    return amplitude
```

The Jupyter notebook contains the full code for these functions, including documentation through Python documentation strings or "docstrings" and code that tests that the functions work as expected. Also included is an application of our `nmo_correction` function to a synthetic CMP (Figure 2). TLE

Acknowledgments

My sincerest thanks to Evan Bianco, Gregorio Kawakami, Jesper Dramsch, and Matt Hall for comments and suggestions and to Öz Yilmaz for generously making the full text of the *Seismic Data Analysis* book available for free and in the open.

Corresponding author: leouieda@gmail.com

References

Hall, M., 2016, Geophysical tutorial: The function of interpolation: The Leading Edge, **35**, no. 4, 367–369, http://dx.doi.org/10.1190/tle35040367.1.

Yilmaz, Ö., 2001. Seismic data analysis: Processing, inversion, and interpretation of seismic data: Society of Exploration Geophysicists, http://dx.doi.org/10.1190/1.9781560801580.

Perez, F. H., and B. E. Granger, 2007, IPython: A system for interactive scientific computing: Computing in Science & Engineering, **9**, no. 3, 21–29, http://dx.doi.org/10.1109/mcse.2007.53.

van der Walt, S., S. C. Colbert, and G. Varoquaux, 2011, The NumPy array: A structure for efficient numerical computation: Computing in Science & Engineering, **13**, no. 2, 22–30, http://dx.doi.org/10.1109/mcse.2011.37.