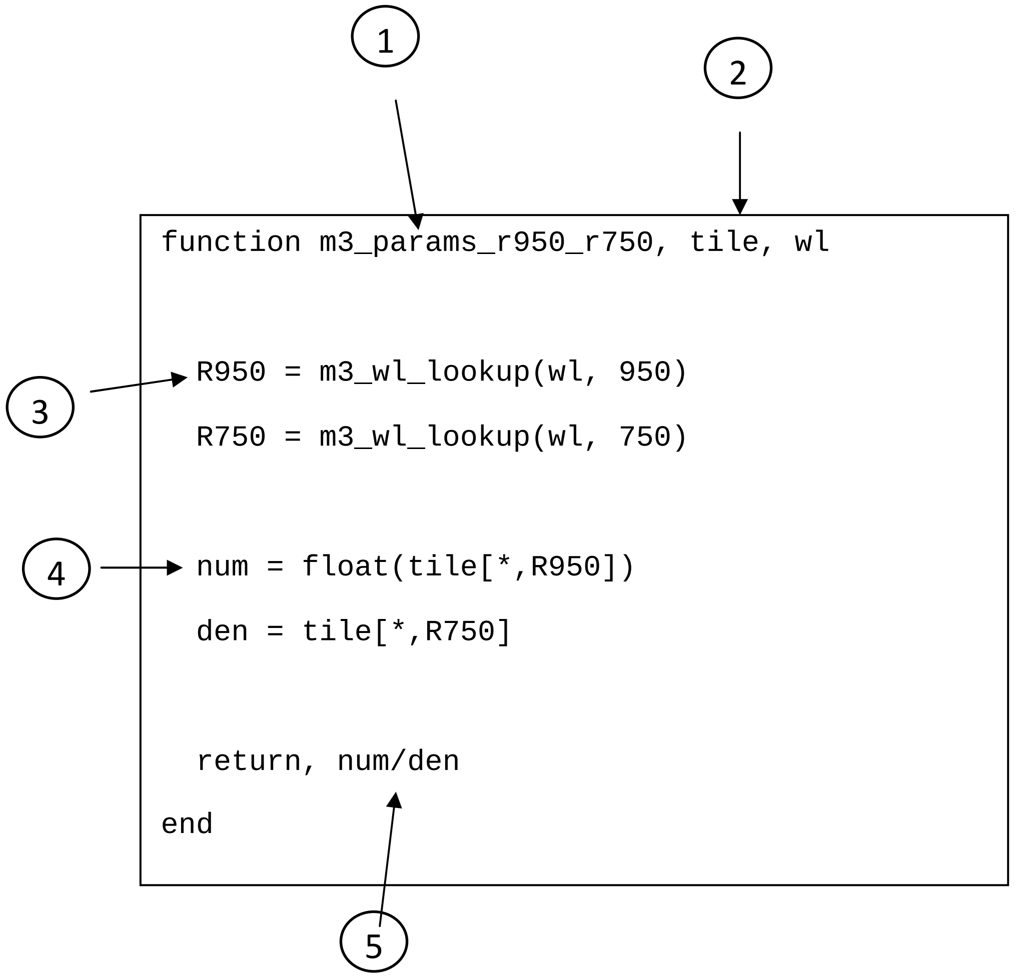
How to write parameters for M3 Tools

Note: These instructions will work for early versions of M3 Tools only. After most of the work of M3 Tools development is complete, we'll start distributing compiled code so that users of ENVI RT can use the toolset. Everything discussed below will only work for distributions of source code (uncompiled *.pro files) and on full ENVI + IDL distributions.

I have listed below one of the simpler of the parameters that has been put into the M3 Tools kit so far, in order to show you how you can easily put together your own parameters code if you wish. At the end of this tutorial, I describe how to add the code you write to the M3 Tools/ENVI system. I do request, however, that you do not distribute code to anyone. I encourage you to use the M3 tools system to test out your own code, but if every person who ever puts a parameter together starts sending them to different people, we're going to end up with different members of the team having a different set of parameters, which will be a mess. So, once you have a parameter in a state that you consider ready to be distributed, please send it to me (Jeff). I will add it to the M3 Tools distribution after I give it a check to make sure everything is as it should be. Thank you!

The example: R950/R750 Ratio

Here's the code I wrote for this parameter. Notes for each numbered circle are below.



```
function m3_params_r950_r750, tile, wl


  R950 = m3_wl_lookup(wl, 950)

  R750 = m3_wl_lookup(wl, 750)


  num = float(tile[*,R950])

  den = tile[*,R750]


  return, num/den

end
```

1. The name – The name of your function (and it should always be a function, by the way) should always start be 'm3_params_<name>', where <name> is the name of your parameter (without the brackets). M3 Tools automatically builds the parameters menu based on the name of this function, and it strips off the text 'm3_params_' and adds the remaining text (<name>) to the parameters menu.

2. Arguments – Your code needs to specify these two arguments (no more, no less).
   -*Tile* is a slice of the input data. It will be a two-dimensional array, with dimensions of (#samples x #bands), so each column in this array is a spectrum.
   -*wl* is the list of wavelengths for an given spectrum.

3. Wavelength lookup – The function m3_wavelength_lookup takes in a wavelength array (wl, first argument) and a specific wavelength (second argument, in nanometers) and returns the index of the closest value in the array (wl) to the specific wavelength given. So the line
   ```
           R950 = m3_wl_lookup(wl, 950)
   ```
   is looking through all the wavelengths contained in the array *wl* for the value closest to 950 nm. It stores in the variable *R950* the **index** of the closest value to 950 it finds.

   We need to use these wavelength lookups because until the final calibrations are finished, we do not know the exact values for M3 wavelengths. This also gives us the ability to easily use datasets other than M3 for comparison.

4. Subsetting tiles – Once you have the position for the wavelength you're looking for stored in a variable like *R950*, you can get the value of the spectrum at that wavelength via an array subset. Remember that what's stored in *tile* is a 2D array, so you're actually processing many spectra at once. To get the value of the spectrum you're looking for in each individual spectrum, you want a statement like this:
   ```
             val = tile[*, R950].
   ```
   The input tile should contain integer values, but for parameters we must convert them to floating point. We also need to do this to make array arithmetic work correctly. So our final statement should look like this:
   ```
             num = float(tile[*,R950])
   ```
   If you need the actual wavelength, rather than the reflectance value, you can get that value using the *wl* array:
   ```
             wvl = wl[R950]
   ```
   Note that in the example code we retrieved the value of the input spectra at two points (which we called *num* for numerator and *den* for denominator). We only need to change the first one to a float because IDL will automatically promote the second variable to a float for us in order to do floating point arithmetic.

5. Return value – You can do anything you want with the input data, but whatever you do, it needs the value you return needs to be a floating point array with the same number of elements as a

single row of *tile*. That number of elements should work out to equal the number of samples in the input data cube. In essence, parameter code should be simple band math, and you should think of your code as being able to return one band per parameter. In the example code we return a simple ratio. It can be considerably more complicated than this if you need it to be. You can call the continuum parameter if you need it. However, if your code gets much more complicated than 1) arithmetic operations or 2) calling another parameter, let me know and we'll give it a separate menu entry. All of the parameters in the main parameter menus need to take in and return a single tile in a fairly self contained way or it will break the loop that is processing the input data.

Adding your code to M3 Tools

Again, please add code to M3 Tools for testing only. Once your parameter is finished, please send it to me and I will add it to the M3 Tools distribution files so that everyone will have the same set of files to the extent possible. I will also make compiled versions so that people using ENVI RT can use the code.

To add your code to M3 tools, follow these instructions.

1. Find the M3 Tools distribution on your hard drive. For most people, this will be in the *save_add* directory of the main ENVI distribution. M3 Tools should be stored in a directory named, oddly enough, *m3tools*. Under *m3tools*, find the *resources* directory, which should itself contain a directory named *parameters*. Inside *parameters* you will see three directories: *pipeline, supplemental, and work*. Save your code in one of these directories. The directory you choose will determine which menu it shows up in in M3 Tools.
2. Make sure to follow the naming convention (m3_params_<name>) outlined in the example code above. If you do not do this, you code might not show up in the M3 Tools menu. You might even cause an ENVI error when the parameters code is run.