**RMB3.py and RasDif.py as a GW Tech Unit project**

Quick review of model results currently requires postprocessing with RMB2.py as well as RD3.py to visualize the model results. The actual “visualizing” requires an additional couple steps to see what the model generated: importing .tif files to ArcMap, setting symbology, tinkering with color ramps

The purpose of this document is to detail the steps of integrating the demonstration code,ColorRasters.py, into RMB2.py or RD3.py. Full implementation may require a couple weeks, depending upon how detailed the training ends up, and how much is improvised, or is just demonstrated.

*Note: Some steps may need to be split into individual work to be done outside the group GW tech meetings, if this provides staff a better chance to fully understand what the code does, for staff at their own pace and exploring their personal coding strengths, with follow-up presentations sharing their work.*

**Step one** is understanding what the demonstration code does.

How and why, it was organized as it is, and discussion on possible improvements before integration.

**Step two** provides how it may be implemented initially and in later steps, and steps I’ve already taken.

Primarily: Make png generation automatic with raster tif creation.

Brainstorm ideas for implementing via command line args and gui.

*Note: Preparation for this involved setting up new draft versions for RMB2.py [as RMB3.py] and a renamed RD3.py [as RasDif.py]*

Step two as a task is to add the new ColorRasters.py functions to the shared MFgis module.

*Note: RMB and RD have already been combined and relabeled RMFB. Sub modules were copies of the same code in separate folders and will be kept together going forward.*

**Step three** will be to make color png creation an option, rather than a default feature.

PNG creation will impact overall post processing time and may not always be desired.

**3a [MFargDefaults.py edits]**MFargDefaults.py is a very simple set of functions and will be the first step in adding the new option

**3b [MFgui.py edits]** . Updating the MFgui.py module is cumbersome.

*Note: Some ideas regarding options include user defined colorfiles, or refining colorfiles to represent a group of rasters rather than one, or changing how intervals are calculated. An added step, may refine these two modules, making new feature integration easier.*

**Step four** is primarily removal of all arcpy module dependence.

A great deal of code exists in the current applications to provide functionality with either Python 2.7 and Python 3.6 or later, as well as use of arcpy in the Citrix environment, or of local ArcGIS installations.

*Note :* Migration to ArcGIS Pro was started in IT nearly 2 years ago and transitioning all the Water Supply Bureau staff is coming soon. Further investigaton into ArcPy for *ArcGIS Pro shows it provides an ArcPy module for Python 3.6. Python 3.6 code that includes previous arcpy calls may work without modificaton even though the geoprocessing library is a complete upgrade. Also it is stated most of the Python 2.7 geoprocessing tools have been ported to Python 3.6. Some features are no longer supported.*

*The RMB code for arcpy currently has little value as it is very slow by comparison to GDAL. The benefit of keeping these arcpy calls is to support Python in a Citrix environment. Code complexity needs to be considered to weigh keeping the arcpy hooks.*

**Step five** is less a step than an on-going requirement for each step.

Rigorous testing of RMB3.py and RasDif.py with the new options. This might be an area where we explore some ideas about unit testing, or at least define testing scenarios up front.

**A general description of ColorRasters.py:**

[\\ad.sfwmd.gov\dfsroot\data\wsd\SUP\devel\source\Python\RMFB\MFgis\ColorRasters.py](file:///\\ad.sfwmd.gov\dfsroot\data\wsd\SUP\devel\source\Python\RMFB\MFgis\ColorRasters.py)

RMB relies on GDAL from osgeo to create single band floating point geoTiffs.

GDAL typically provides raster colors with multiband, or in the case of single band rasters, only does integer rasters, while our postprocessed results are almost always floating point. There was some less than detailed descriptions, on the ‘web’ about “gdaldem”, but rarely in a Python context.

“gdaldem” seemed to be the way to go but requires colorfiles to define color ramp ranges and RGB colors. Setting up RGB color themes, is challenging and seems to be somewhat of an art-form.

The ColorRasters.py code uses 35 color ramp themes I generated with an Rscript using RColorBrewer and an external GDAL function called gdaldem [installed with GDAL when it is added to Anaconda or other set stack of Python modules.]

Execution of “gdaldem” requires a module named “subprocess” and may result in you having to install a new module via Anaconda. This will ultimately result in us working with each of the modelers to make sure they have the necessary module installed, and they may need to upgrade to the new Anaconda Professional, which facilitates the stack administration. Note Mike Carney has retired, so having him in our corner for quick admin permissions, isn’t an option, and we may need to train his replacement, even for setup of Anaconda Professional.

The code calculates min, max and break intervals of the data in the raster, [and currently breaks it into equal ranges of value categories. This could be revised to implement alternate break intervals such as geometric, or standard deviation, etc. as ArcMap provides]

Another area for refinement would be how themes with RGB colors are organized in a data structure (currently a Panadas dataframe). After building the dataframe, it became obvious it was not a very efficient structure to add new themes, let alone the selecting the values, is a bit awkward.

The code also does a little math trick to calculate precision of the break intervals, which his helpful when data value ranges are very small (between -1 and 1)