Center of Space Research

Gravity Recovery and Climate Experiment

www.csr.utexas.edu

Google Earth Engine (GEE) Visualization

GRACE/GRACE-FO

User Manual

Version 1.0.0

10/27/2020

Document Number: 1

Prepared by:

Riley Matthews Jacobs, UT CSR
2020 Undergraduate Research Assistant

Contact Information:
Center for Space Research
The University of Texas at Austin
3925 W. Braker Lane, Suite 200
Austin, Texas 78759-5321, USA
grace@csr.utexas.edu

Reviewed by:

Teresa Howard, Research Associate
UT CSR, Austin

Approved by:

Dr. Srinivas V. Bettadpur, UT CSR
Director, Center of Space Research

Table of Contents

1. Introduction

The following document outlines a generalized procedure to upload as well as maintain the Gravity Recovery and Climate Experiment (GRACE) as well as the Gravity Recovery and Climate Experiment - Follow On (GRACE-FO) data into the Google Earth Engine (GEE). Google Earth Engine is a geospatial visualization platform that allows scientists to directly interact with various satellite data such as MODIS, LANDSAT, and Sentinel data.

All GRACE and GRACE-FO dataset solutions discussed throughout this manual are produced by the Center of Space Research, University of Texas - Austin (CSR). Three critical datasets -- RL06 Mascon Solution, Ocean Mask, and Land Mask -- were ingested and published into the GEE in accordance with CSR. Such datasets can be accessed through the GEE user platform known as the earth code editor. This code editor allows any scientist to simultaneously perform data analysis while also visualizing the deviations of equivalent liquid water thickness data measured by GRACE and GRACE-FO.

It is important to note that this document references the most recent version published by CSR RL06 Mascon Solutions, Version 2. All of the versions are downloaded as NetCDF time series in which the mascon estimation solutions are represented as harmonical coefficients. Additionally, the earth is represented as an ellipsoid. The NetCDF files provide the time series for the monthly mass grids in which gravitational anomalies are relative to the 2002-2009 time-mean baseline.

2. Overview

Three critical stages must be undertaken to properly upload, ingest, and perform data calculations for all GRACE and GRACE-FO data in Google Earth Engine. The first stage is file and date manipulation within ArcGIS Pro. ArcGIS Pro is a geographic information system (GIS) software that provides an interface to work with maps and geospatial data. This platform provides the user with the tools to perform large data management operations such as the transformation of GRACE and GRACE-FO mascon solutions. Through the addition of ArcGIS Pro into the typical procedure, the source NetCDF file (accessed through the CSR website) is exported to individual GeoTIFF files. ArcGIS Pro is a user-friendly option for data format transformation, in contrast to other tools tested. Following export to a TIFF file, the next step uses GEE to upload and asset each monthly mass grid. The final stage is processing the data within GEE. Scripts to perform typical tasks are outlined below.

2.1. Conventions

This document provides screen shots as well as a corresponding narrative to describe how to use the various platforms.

When an action is required on the part of the reader, it is indicated by a line beginning with the word “Action:” For example:

Action: Copy

Often this will follow with a script that requires the user to copy the script as well as properly run in the user’s respective program to generate the desired action.

Note: The term ‘user’ is used throughout this document to refer to a person who requires and/or has acquired access to the Google Earth Engine.

2.2. Warnings

The user must request access to the Google Earth Engine code editor prior to working with scripts and writing scripts for each image collection uploaded. Additionally, the ArcGIS Pro software may have associated costs.

3. Getting Started

Before file conversion can take place, the latest version of the mascon solution must be downloaded. Such products are published directly on the Center of Space Research website. These mascon solutions are periodically updated as new corrections are applied and new geographic data is measured. The data can be downloaded in NetCDF format. The land and ocean grids that accompany the CSR RL06 Mascon Solutions (V02) are downloaded in a similar format from the CSR website. It is important to download these masks rather than a third party mask as the CSR solution in particular minimizes the leakage along the coastline.

At the time of this publication, all solutions are available at:

http://www2.csr.utexas.edu/grace/RL06_mascons.html

3.1. User Access Considerations

A user may run the code by copy and pasting each script within their own GEE code editor platform. Thus, when applying any of the information or scripts, first import the GRACE data into GEE and then copy scripts as desired.

3.2. Accessing the System

Before working directly with the code in Google Earth Engine, each new user must request access from Google. Access is granted through the google form: signup.earthengine.google.com. This platform allows the user access to all GRACE uploaded data, as well as many other datasets from other satellites. It is recommended that the user is granted access before beginning this tutorial.

3.3. System Organization & Navigation

Google Earth Engine has generated an extensive and detailed “Get Started” Guide for most scientific desired actions. It is once again recommended to traverse this beginners guide before beginning this tutorial. It provides an overview as to how to begin operating in Google Earth Engine and will allow a more complex understanding of this User’s Guide’s inner workings.

For this paper's purpose, the predominant work will be directly within the code editor of the Google Earth Engine. Within that format, there are three main aspects: the left panel, the code interface, and the right panel.

The left panel will be home to the user's file and folder organization. These contain the scripts as well as quick Javascript commands. Additionally, some time will be spent in the "Assets" section in which the user will be able to upload and house new datasets in GEE.

The actual code interface is housed in the center panel. This is where all code and scripts will be copied and run. It is important to be aware that GEE's code editor is a Javascript platform, but certain API's may be downloaded and connected to run other programming languages, most notably, Python.

Finally, the right panel is the real interface with the user. This allows the user to interact with the data stored. It houses an inspector for querying the map, an output console, and a manager for long-running tasks.

4. Using the System

This section will examine the sequential stages of downloading the datasets, converting the datasets to a compatible GEE format, and finally, uploading and running GRACE and GRACE-FO data within GEE. Each of the following topics will expand on how to properly address the task of large data consumption. Thus, a thorough understanding of the usage of several files -- NetCDF, GeoTiff, and KML files -- is highly advised.

4.1. Generating GEE Images

The following flow chart visualizes the workflow that must be undertaken to upload into GEE. Certain steps, depending on the user's preferences, may be deemed unnecessary or undertaken in a different manner.

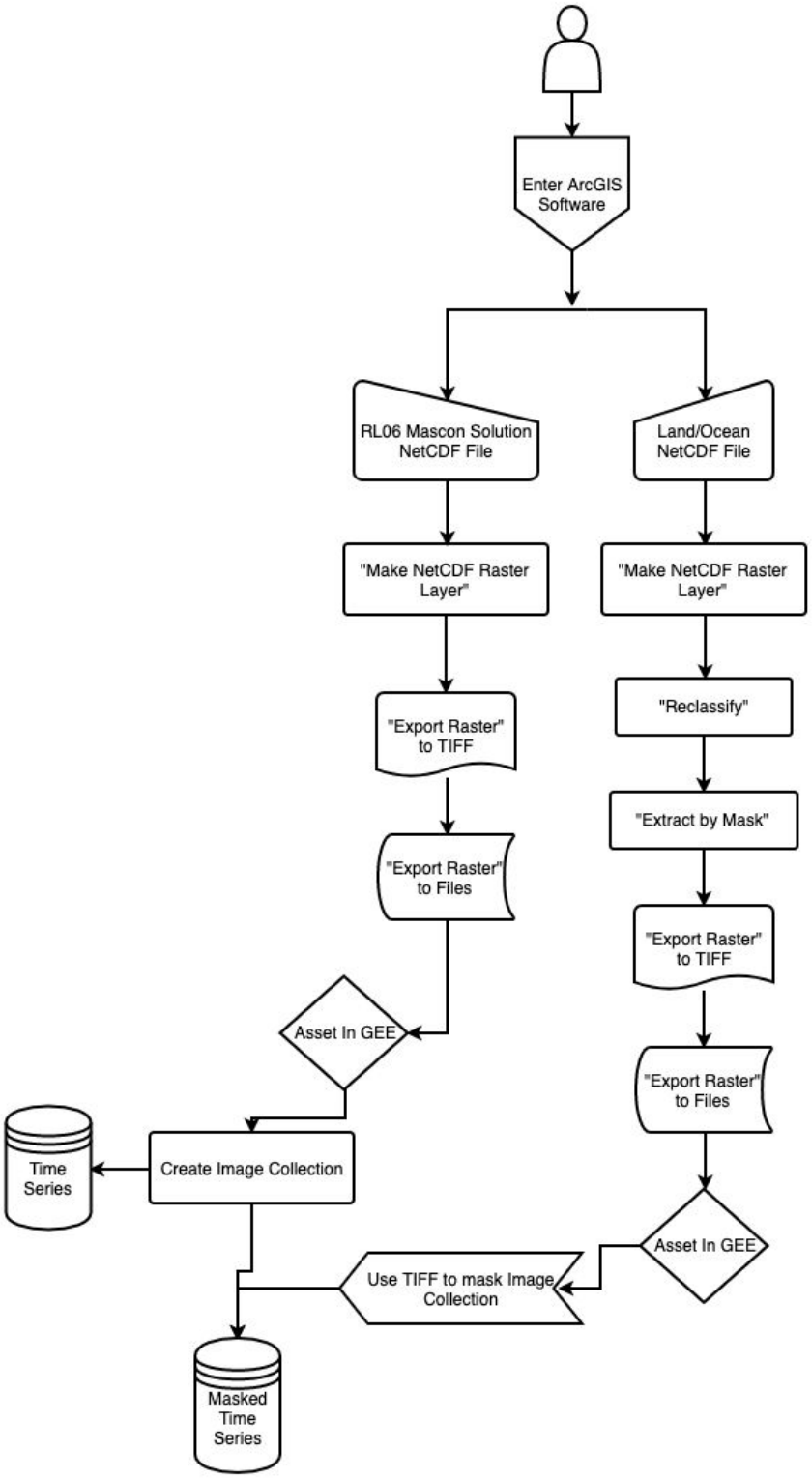


Fig 1. Flow Chart outlining the workflow to take .nc file and asset it to GEE

4.1.1. Understanding and Downloading RL06 Mascons/Datasets

The RL06 mascons on the CSR website are available in two formats. The first is the GRACE and GRACE-FO solution times series provided as a single NetCDF with the appropriate corrections applied and the second in separate component files. It is advised that the user downloads the former through the proper links as the latter is recommended for more advanced users. The current NetCDF file for the data spans from April 2002 to August 2020. The NetCDF is regularly updated and uploaded as more monthly mass grids become produced and new corrections are applied, so the latest NetCDF should be downloaded. For more information about time series - the CSR website provides proper documentation.

4.1.2. ArcGIS Pro

When accessing the data from the CSR website, for convenience and data manipulation on the technical side the data is packaged in a NetCDF file. When uploading data as an “Asset” in GEE or more simply inputting your own spatial data for visualization, the data must be one of the following formats: GeoTIFF (.tif, .tiff) or TFRecord (.tfrecord + .json). This requires downloading the NetCDF file and converting it to either a GeoTIFF or TFRecord. This process has several acceptable methods.

This section outlines one method that is rather user friendly and does not require extensive knowledge of geospatial files or coding experience. ArcGIS Pro is a software package that provides the user with an understandable interface -- both in the form of a downloadable app as well as online. This software can be used to create and use maps, compile geographic data, analyze maps and geographic information, but most importantly for the purposes of this paper, manage geographic information in a database.

Within the appendix, however, there is also an example code that might be run through a Python platform. This Python platform however must support the ArcGIS Pro package of Arcpy. This package is critical for the creation of the coordinate system WGS84 and EPSG 4326. Without that particular package, the NetCDF will output a GeoTIFF file that is unreadable by GEE. The benefit of using a third-party software such as ArcGIS Pro is the ability to output data with the correct coordinate system embedded in the file without additional work on the part of the user. Arcpy automatically reads the native coordinate system when interacting with the NetCDF eliminating one additional step for the user.

Figure 2 below depicts a typical screen that is shown on the ArcGIS Pro platform. Very little in-depth information about using ArcGIS Pro is provided here except the information that pertains to using ArcGIS Pro to convert data for GEE.

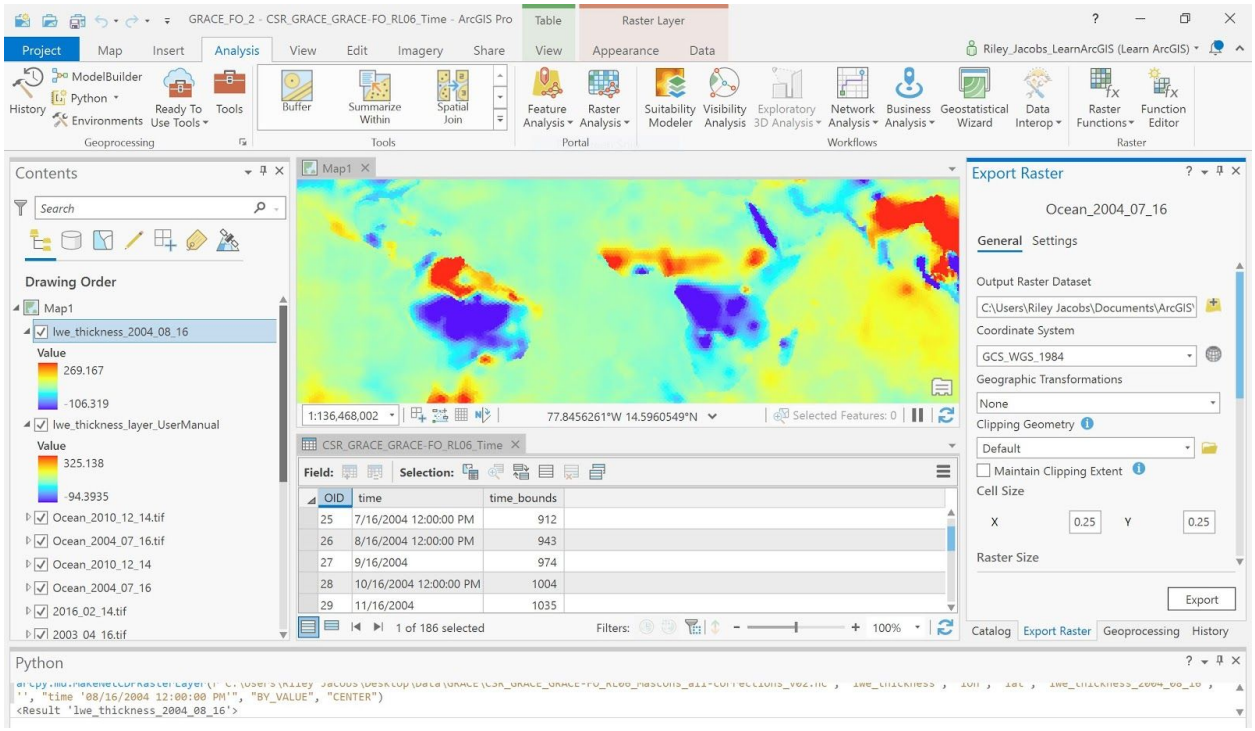
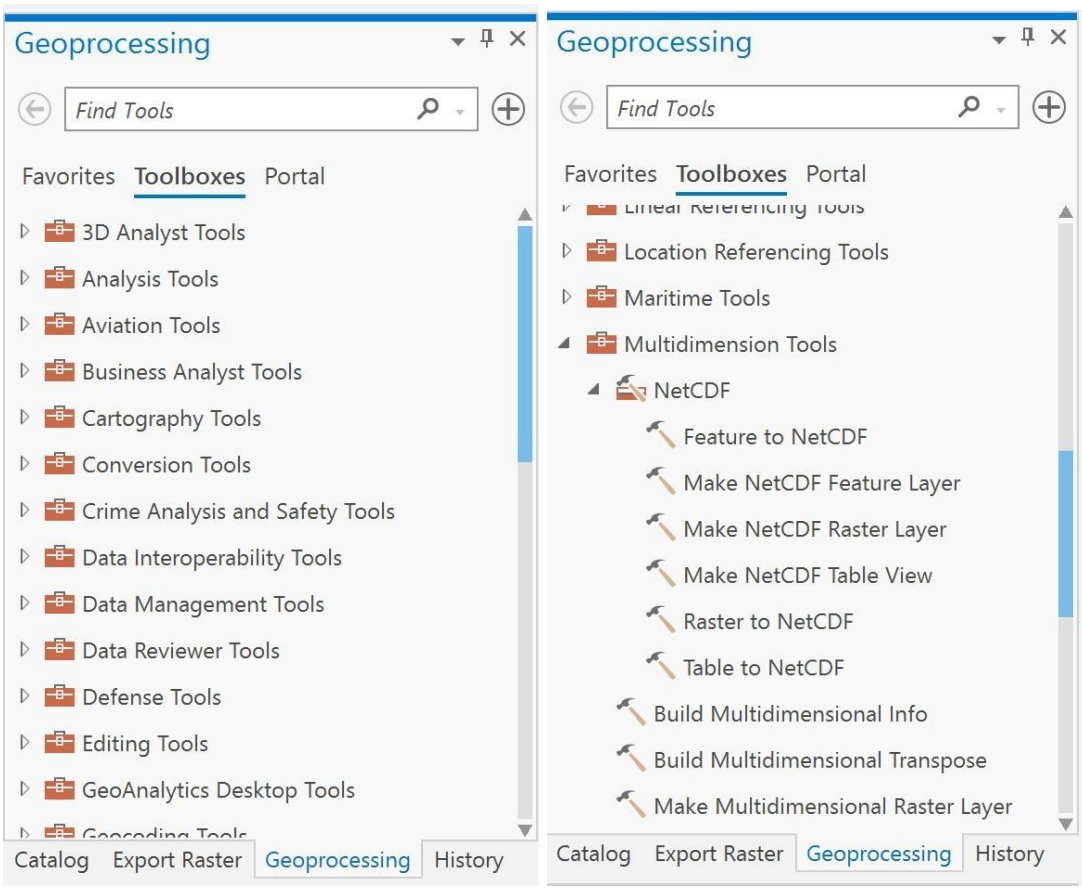


Fig 2. A typical screen for GRACE and GRACE-FO data in ArcGIS Pro

4.1.2.1. Toolboxes

The simplified goal is to create a TIFF file from the NetCDF and export it from the ArcGIS Pro platform onto the local computer disk or other location. This process means that the majority of the user's time will be spent in the analysis tab under "Tools." Figure 3 below demonstrates a typical visual of the toolboxes that ArcGIS Pro houses under the tab, "Tools."



(a) (b)

Fig 3. (a) Various Toolboxes that a user may choose and (b) expanding each toolbox provides the user with more geospatial tools

For this specific task, the user will be accessing three specific toolboxes and tools to generate a TIFF file.

The first task is creating a table to demonstrate to the user each timestamp associated with the monthly mass grids. This is an optional task but can be very helpful as the user becomes more knowledgeable about the inner workings of the NetCDF time series. In order to see the output table, make sure that both a Map pane and the “Contents” pane are open.

To create a table in ArcGIS, traverse to the “Multidimensional Tools” under the “Tools” tab and open the “NetCDF” sub-tab in “Multidimensional Tools.” Once the sub-tab is open, click “Make NetCDF Table View.” This is depicted in figure 3 (b). Once ArcGIS Pro opens the depicted in figure 4, select the variables “time bounds” and “time.” Be sure to replace the automatically generated “Output Table View” name with a name that is understandable by any viewer. Finally, set the Row Dimensions to “time” in order to ensure that each GRACE time step is output to the

table view. The resulting table will include the time, either as “MM/DD/YYYY” or “MM/DD/YYYY 12:00:00 PM”, and the time bounds, an integer representing the number of days from a specific date (ask Himanshu what date that is).

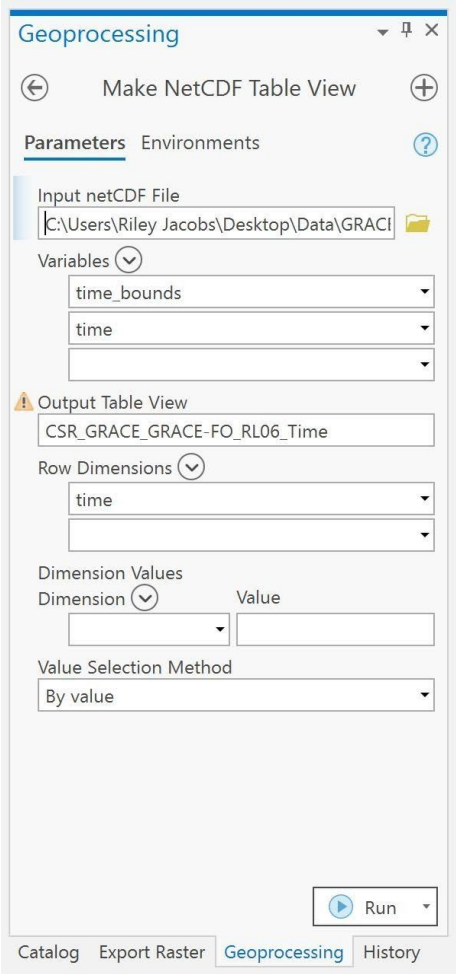


Fig 4. Generating a Table of “time” from Geoprocessing Tools

The second aspect is returning once again to the “Multidimensional Tools” subtab of “NetCDF” and choosing “Make NetCDF Raster Layer.” This will generate what is known as a raster layer or a layer that represents one month of the water mass grids. Thus, over the time series, you may have a large number of raster layers depending on the time span of your time series. Choose the data information desired, in this case, “lwe_thickness.” Leave the X dimension as “lon” and the Y dimension as “lat.” Notice that the Output Raster Layer is automatically named by adding “_Layer” to the variable name. Skip the pull down option for Band Dimension. Instead, look for Dimension Values, choosing “time” for Dimension and a time stamp for Value. Begin by choosing the first month of GRACE data, April 18th, 2002, to practice making a raster layer (as

shown in the figure below). Run the tool to generate your first raster layer. It will appear in the map.

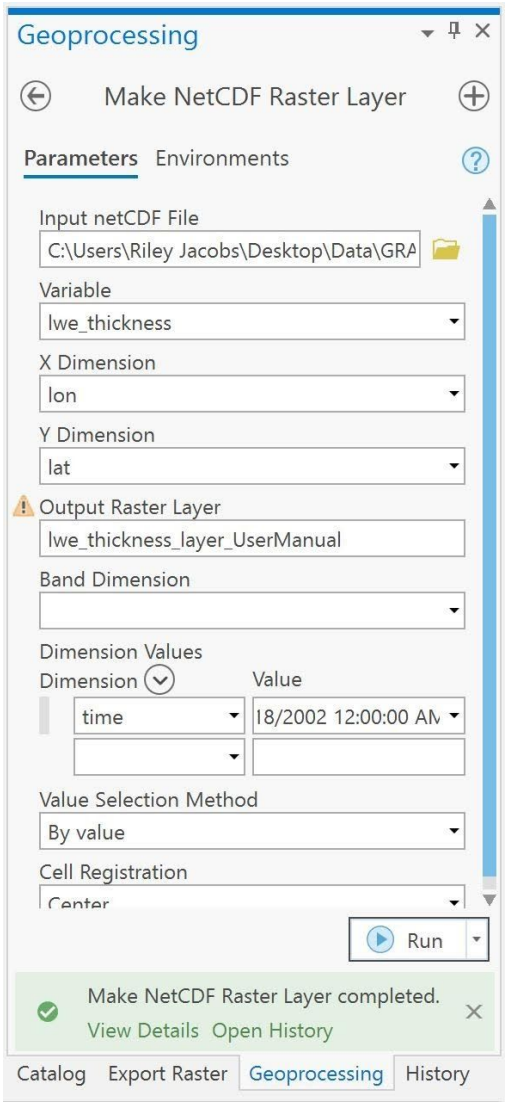


Fig 5. Generating a new raster layer for each timestamp

ArcGIS will notify the success of each task if a green check icon appears at the bottom of the geoprocessing tab. Similarly to outputting a table, make sure to correctly name and label each raster layer as the time stamp you are referring to.

4.1.2.2. Create Mask

An important tool in geospatial visualization is masking data over land or over water. In simple terms, “masking” makes the regions in which you don’t want data an integer of 0 so those

regions become “blank.” This is a very helpful geospatial tool, especially when generating visuals such as maps or animations.

To begin to create a mask, the user must first download the ocean and land mask .nc files that are found on the CSR website. These masks are specifically created to fit the GRACE and GRACE-FO RL06 Mascon Solutions and thus, decrease the leakage along the coastline. At the time of user guide publication those masks can be found by following the link:

http://www2.csr.utexas.edu/grace/RL06_mascons.html

Once each mask is downloaded, you must once again create a raster layer from the .nc file. Following the previously outlined steps, create a Raster Layer from the “Input NetCDF File” from either the land or ocean mask .nc. At the end of this process, the user should have two rasters: one land mask raster and one ocean mask raster.

The next step is converting these rasters into an 8-bit integer raster. This will allow you to “mask” the data points by making those values 0 or “transparent.” Under “Toolboxes,” scroll until you find “Spatial Analysis Tools” then “Reclass.” Finally, the “Reclassify” tool will be shown. Using the “Reclassify” tool, input each raster layer and signify that 0 is equivalent to “NODATA” and 1 is 1, as shown in figure 8.

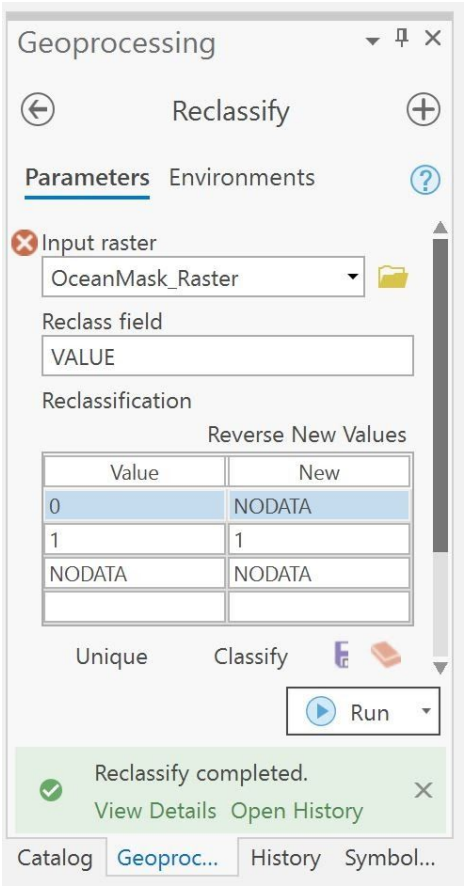


Fig. 8 Tool "Reclassify" reassigns integer to mask data from the visualization

Through this process, the user is essentially making a conditional statement that describes that if the mask is demonstrating a zero value then, in fact, "no data" exists and that area will appear transparent. This process will automatically create an 8-bit file for each mask raster.

Once you have properly reclassified your mask raster, ArcGIS Pro provides the user with the tool of "Extract by Mask." Essentially, this provides the user with combining two raster layers. Therefore, by combining the mask raster layer with the time series raster layers of the overall RL06 mascon solution, ArcGIS Pro outputs a third raster layer that contains the proper mask with the equivalent liquid water thickness data involved in only the desired areas.

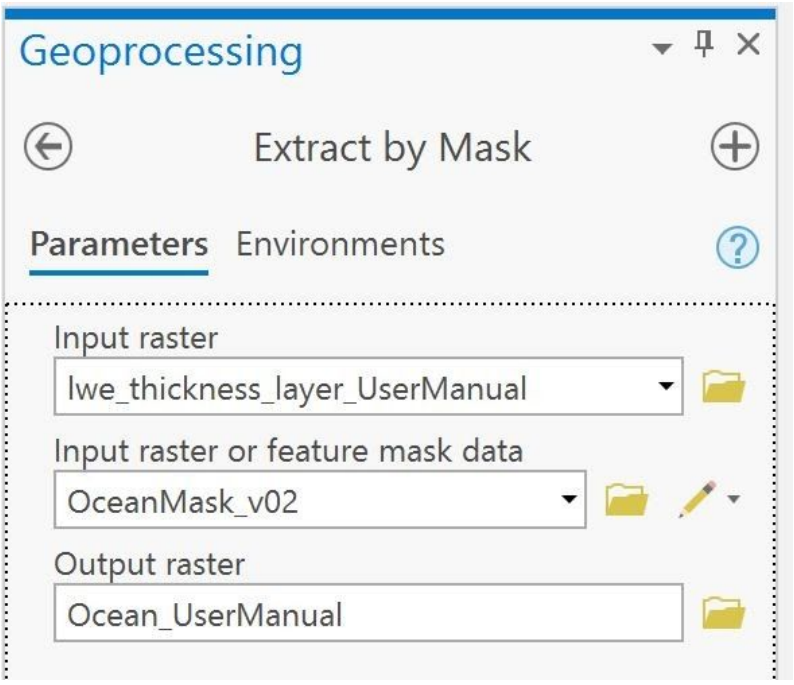


Fig. 9 “Extract by Mask” allows the user to apply a mask to a raster layer

After the mask is extracted, two new collections of TIFF files are subsequently created. One collection of TIFF’s will contain the Ocean Mask .tif files and the other will contain the land mask .tif files for each ascending time step in the time series.

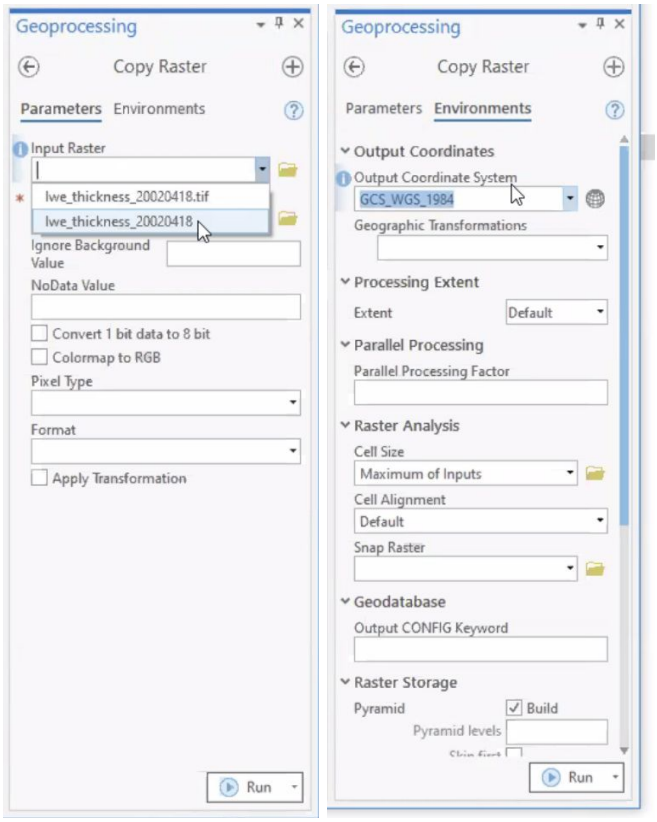
Together, this manual has outlined the creation of TIFF files from three NetCDF files into three collections: overall RL06 mascon solutions, ocean mask, and land mask.

4.1.2.3. Copy Raster to TIFF

Once you have a raster layer for each time step, the following task generates a TIFF file for each raster layer. The eventual goal is to create a series of TIFFs that the user will individually upload as an “image” to GEE and compile each “image” into an “image collection.” This will be more formally addressed later.

Once again, return to “Tools” below the “Analysis” tab overhead in ArcGIS Pro. Afterward, scroll and double click on “Data Management” and then “Raster.” Double click on “Raster Dataset” until it shows the “Copy Raster” tool. Double click on “Copy Raster” and a similar screen to figure 6 will appear. Output the Raster Dataset of the time layer you are desiring. Following the earlier example, you would choose the raster layer of April 18th, 2002.

Again, it is important to choose an “Output” that is clear to the user. This is especially critical within this ArcGIS Pro step as the raster layer will be converted and placed as a .tif file in the output location of the computer’s hard drive.



(a) (b)

Fig 6. Exporting a raster layer of a timestamp to a TIFF file utilizing “Copy Raster” (a) and applying a coordinate system (b)

Before exporting, it's critical that the coordinate system chosen is “GCS_WGS_1984” and the Pixel Type is 32 Bit Float. Without these chosen, GEE will be unable to “asset” your data and will not properly upload the TIFF file. This is investigated by clicking on the environments tab shown in figure 6.

If the export (or file export) is successful, a new .tif file will be produced under the left panel of “Contents” and “Drawing Order.” Most importantly, a .tif file will be on the disk of your computer in the file location of choice. This tool enables the user to generate a TIFF file from raster data and output data to disk in the process. This is not merely a layer in which it is held in the software’s memory, but rather is exported to the actual hard drive of the computer.

Taking advantage of coding languages such as Python, a user can automate this process. This will be elaborated more in-depth in the following section 4.1.2.5.

4.1.2.4. Exporting TIFF

As the user begins to output more and more .tif files for each time step in the time series of the original .nc file, the user may choose to utilize the tool to export the TIFF rather than the tool of "Copy Raster." This is done rather easily by taking advantage of the tool "Export Raster." In the "Contents," left most panel, right-click on the raster layer you want to export, click "Data" and then, click "Export Raster." The following tab will open as shown below.

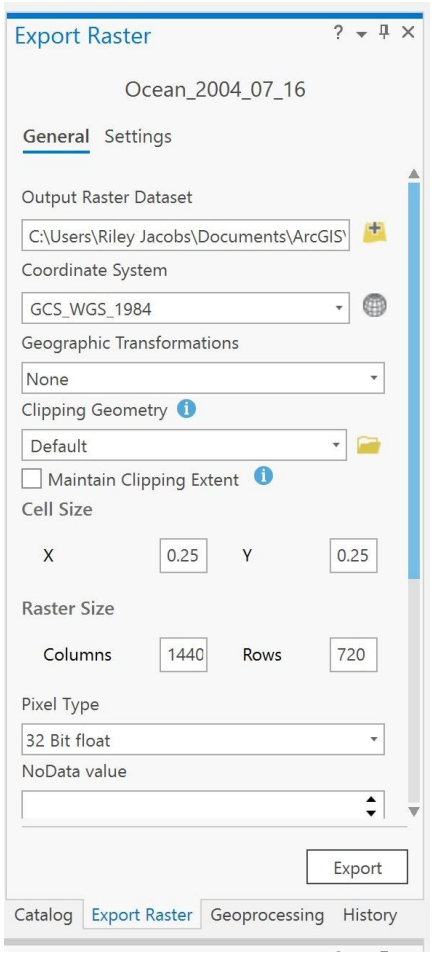


Fig 7. The final "Export Raster" to export to computer files

Without changing any settings, choose the location on your computer you wish to house your .tif file such as a folder, and click on the right lower button "Export." Once again, a green banner will appear on the screen indicating it will have successfully exported and the .tif file will appear within the desired folder.

4.1.2.5. Python

ArcGIS Pro has provided a dual-platform within the software that allows the user to leverage geoprocessing tools by creating a stand-alone script or live script for some of the more tedious and cumbersome tasks. In other words, utilize a previously coded solution so the user does not need to manually click each step from a NetCDF to TIFF file. Like previous sections, this section will address the processes outlined in this paper and will refrain from addressing other tasks that can be achieved. It is highly encouraged to read the help page that ArcGIS has produced to properly take advantage of all that you can do.

There are several hard coding lines that take advantage of the dual nature of Python in ArcGIS Pro. The "Analysis" tab houses a sub-tab called "Python" in which the user can quickly launch an interactive coding platform without creating a stand-alone script. It displays a window similar to the image shown in figure 10.

Fig 10. Python command prompt that allows the user to directly program commands while running ArcGIS Pro

This package provides a direct line to program and run code within ArcGIS Pro. Several commands are outlined below to circumvent traversing the "Toolboxes" and run commands to quickly output resources.

Python Commands	
Traverse Times in Table	>> Import arcpy >> cursor = arcpy.da.SearchCursor('CSR_GRACE_GRACE-FO_RL06_Time', ['time']) >> for row in cursor >> print(row) >> print(row)
Generate Raster Layer	>> Import arcpy >> arcpy.md.MakeNetCDFRasterLayer(r"C:\Users\Riley Jacobs\Desktop\Data\GRACE\CSR_GRACE_GRACE-FO_RL06_Masconns_all-corrections_v02.nc", "lwe_thickness", "lon", "lat",

	"lwe_thickness_2002_04_18", '', "time '04/18/2002 12:00:00 AM'", "BY_VALUE", "CENTER")

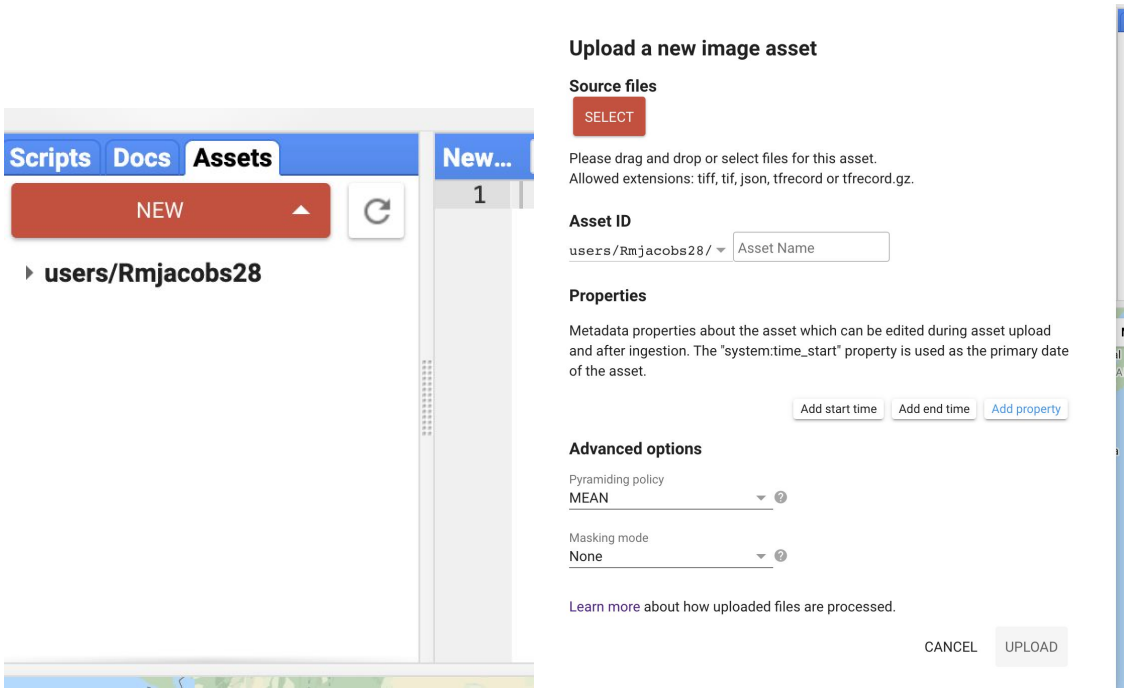
Table 1. Python Commands in ArcGIS

4.1.3. Google Earth Engine

With a collection of three larger groups of TIFF files, the user is now ready to import the data into GEE. Such a process takes place within the "Assets" tab of the GEE. Here the user may upload as many or as little TIFF images as desired. This process is rather time intensive so is recommended that the .tif files be fully generated and downloaded before being this stage in the process.

4.1.3.1. Asset an Image

When uploading a TIFF into GEE, begin by clicking on the "Asset" tab. There you will click the large red "New" in the left panel, revealing a box listing several options. Select the Image Option, GeoTIFF (.tif, .tiff) or TFRecord (.tfrecord and .json). In the Upload a new image asset window, follow the directions to select a source .tif file from your local drive. The Asset ID will automatically update to the source .tif file name. For this tutorial's purposes, enter the start and end time of each uploaded TIFF, using the recommended time format: yyyy-mm-dd hh:mm:ss. For the GRACE and GRACE-FO data, it is sufficient to only include the start time as the time step of the .tif file. That will allow you to differentiate over the months as you begin to analyze and further visualize the time series. Both of those processes are documented in figure 11.



(a) (b)

Fig 11. (a) Uploading a new TIFF through "New" and (b) outlining specific monthly properties

From here it is rather mindless and requires more time than thought. As you begin to build more image assets into GEE over time (creating those time series stamps of the original NetCDF) other processes can occur -- such as an Image Collection -- to fully take advantage of the computational powers of GEE.

Some additional resources for user's wishing to bulk upload data to Google Earth Engine can be found at:

https://www.tylanderson.com/post/uploading-custom-raster-image-collections-to-google-earth-engine/

https://gis.stackexchange.com/questions/303346/code-earthengine-uploading-multiple-images-ingestion-failed-time-out

4.1.3.2. Create an Image Collection

Image Collections provide an easy way to consolidate the images back into a manageable time series. Begin by clicking "Asset" and scroll down to "Image Collection" and name your new image collection. From there, enter the image collection by double-clicking. For each TIFF in the time series individually click the "Add Image." Each associated TIFF uploaded to the "Assets"

will have an assigned source ID. Copy the source ID before entering the "Add Image" to make the transition less convoluted. It is also possible to manually drag and drop images in the assets folder directly into an image collection. Additionally, a helpful tool is using the command imagecollection1.Merge(imageCollection2) to create an additional third image collection from two previously created Image Collections. This tool is especially helpful as GEE limits Image Collections to 100 images so if the user wants a time series of more than 100 time steps then the user will need to later merge two separate Image Collections.

4.2. Taking Advantage of GEE

This section is a combination of copied script and helpful tips that might begin to build your repertoire of code within GEE. All of these scripts can be applied to the actual GRACE and GRACE-FO images uploaded or expanded to analyze data for various other satellites. A greater collection of scripts are within the Beginners Guide provided by GEE.

4.2.1. Utilizing Image Collection

Image collections are extraordinarily helpful in analyzing over the time span that GRACE and now GRACE-FO is in operation. From imaging to data collection, GEE provides a simple way of interfacing with the data and catering it to each scientific need.

4.2.1.1. Project One Image

```
// Project One Image
//
// Import the Image Collection you wish to view
// This is a GRACE/GRACE-FO Image Collection that has an applied land mask
// One helpful tool is enabling the Image Collection to sort the TIFFs in order of ascending
date
var imagecol = ee.ImageCollection('users/Rmjacobs28/GRACE/Land-Mask')
  .sort('system:time_start');

// Using .filterDate choose the time span you want to investigate
var image = imagecol.filterDate('2007-01-01','2007-12-31')

// Rename an Image Collections band if desired
var adjustBands = image.select(['b1'],['lwe_thickness'])
print(adjustBands)

var equivalentWaterThicknessCsr = adjustBands.select('lwe_thickness');

// Define the visualization parameters based on the maximum and minimum data points
```

```
var equivalentWaterThicknessCsrVis = {
  min: -20.0,
  max: 20.0,
};

// To project just one image the command “Map.addLayer” will project one image into GEE’s
lower panel
// Add as many layers as desired (a good way to quickly compare months to one another)
Map.addLayer(
    equivalentWaterThicknessCsr, equivalentWaterThicknessCsrVis,
    'Equivalent Water Thickness CSR');
```

4.2.1.2. Animated Video

Animations are a quick and easy way to display visually the information to the viewer. GEE is an excellent resource in providing a fast way to quickly project information on the screen to fit the user’s preferences. These preferences may come in the form of various color scales and time spans. GEE also provides two commands: ui.Thumbnail and getVideoThumbURL. The former prints to the console and the latter allows the user to generate URLs that provide the user a sharable video for presentations or websites.

```
// Animated Video
//
// Import the Image Collection you wish to view
// This is a GRACE/GRACE-FO Image Collection that has an applied land mask

// Define an area of interest geometry with a global non-polar extent.
var aoi = ee.Geometry.Polygon(
  [[[-179.0, 78.0], [-179.0, -58.0], [179.0, -58.0], [179.0, 78.0]]], null,
  false);

// Import the lwe thickness from the image collection under the band name ‘b1.’ Note that the
// image collection extends through the entire time series; limit the collection to the year 2007
var lweCol = ee.ImageCollection('users/Rmjacobs28/GRACE/Land-Mask')
  .filterDate('2007-01-16', '2007-12-16')
  .select('b1');

// Define arguments for animation function parameters.
var videoArgs = {
  dimensions: 768,
  region: aoi,
```

```
  framesPerSecond: 7,
  crs: 'EPSG:3857',
  min: -40.0,
  max: 35.0,
  palette: ['blue', 'purple', 'cyan', 'green', 'yellow', 'red']
};

// Print the animation to the console as a ui.Thumbnail using the above defined
// arguments. Note that ui.Thumbnail produces an animation when the first input
// is an ee.ImageCollection instead of an ee.Image.
print(ui.Thumbnail(lweCol, videoArgs));

// Alternatively, print a URL that will produce the animation when accessed.
print(tempCol.getVideoThumbURL(videoArgs));
```

Here is an example of a quick animation that you can make to highlight the changes experienced by Australia. This is slightly different from the previous animation as it narrows the scope for the viewer. It demonstrates an easy way to quickly visualize that to a specific region of interest without too much hard coding on the part of the user.

```
// Animated Video Australia
//
// Import the Image Collection you wish to view
// This is a GRACE/GRACE-FO Image Collection that has an applied land mask

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Australia
// Define an empty image to paint features too
var empty = ee.Image().byte();

// Draw a rectangle around Australia for proper viewing
var Australia =
    /* color: #d63000 */
    /* displayProperties: [
      {
        "type": "rectangle"
      }
    ] */
    ee.Geometry.Polygon(
        [[[104.88157659262788, -3.2419988010251535],
          [104.88157659262788, -44.396516698200834],
          [163.76829534262788, -44.396516698200834],
```

```
        [163.76829534262788, -3.2419988010251535]]], null, false);

// Import the lwe thickness from the image collection under the band name 'b1.' Note that the
// image collection extends through the entire time series; limit the collection to the year 2007
var lweCol = ee.ImageCollection('users/Rmjacobs28/GRACE/Land-Mask')
  .filterDate('2005-01-16', '2005-12-16')
  .select('b1');

// Define arguments for animation function parameters.
var visArgs = {
  min: -40.0,
  max: 40.0,
  palette: ['red', 'yellow', 'green', 'cyan', 'purple', 'blue']
};


// Convert each image to an RGB visualization image by mapping the visualize
// function over the image collection using the arguments defined previously.
var lweColVis = lweCol.map(function(img) {
  return img.visualize(visArgs);
});


// Import country features and filters to Australia.
var australiaCol = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017')
  .filterMetadata('wld_rgn', 'equals', 'Australia');

// Paint country features edges to the empty image.
var australiaOutline = empty
  .paint({featureCollection: australiaCol, color: 1, width: 1})
  // Convert to an RGB visualization image; set line color to black.
  .visualize({palette: '000000'});

// Map a blend operation over the temperature collection to overlay the country
// border outline image on all collection images.
var lweColOutline = lweColVis.map(function(img) {
  return img.blend(australiaOutline);
});

// Define animation arguments.
var videoArgs = {
  dimensions: 768,
  region: Australia,
  framesPerSecond: 7,
  crs: 'EPSG:3857'
};
```

```
// Display the animation.
print(ui.Thumbnail(lweColOutline, videoArgs));
//print(waterColOutline.getVideoThumbURL(videoArgs));
```

4.3. Analyzing Data

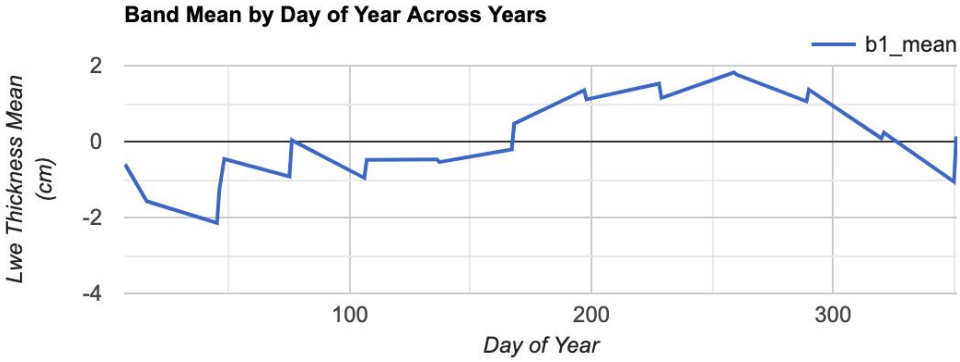
An important aspect of geospatial data is the ability to utilize the data to supplement scientific research and conclusions. The ability to quickly analyze data, apart from visualizing, is critical to take advantage of all that GRACE and GRACE-FO have to offer to the scientific community. These are only a few of the many analytical scripts that can be applied to an image or image collection to provide the basis for scientific conclusions.

4.3.1. Day-of-Year Chart

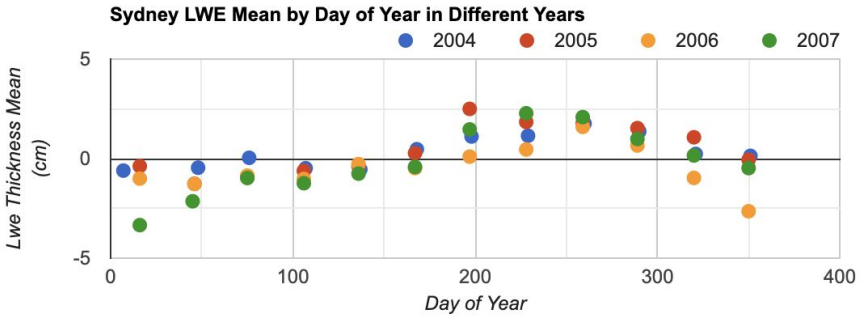
```
// DOY Chart
//
// Import the Image Collection you wish to view
// This is a GRACE/GRACE-FO Image Collection that has an applied land mask

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Define a FeatureCollection: three regions of interest.
var Austin = ee.Feature(    // Austin
   ee.Geometry.Rectangle(-97.733, 30.27, -97.00, 30.00), {label: 'Austin'});
var Sydney = ee.Feature(  // Sydney.
   ee.Geometry.Rectangle(151.197, -33.84, 151.99, -33.00), {label: 'Sydney'});
var Delhi = ee.Feature(  // New Delhi
   ee.Geometry.Rectangle(77.176, 28.648, 77.10, 28.40), {label: 'New Delhi'});

var regions = new ee.FeatureCollection([Austin, Sydney, Delhi]);

// Load several years of GRACE data.
var collection = ee.ImageCollection('users/Rmjacobs28/GRACE/Land-Mask')
   .filterDate(ee.Date('2004-01-01'), ee.Date('2007-12-31'))
   .select('b1');


// Define a chart with one series in the sydney region, averaged by DOY.
var series1 = ui.Chart.image.doySeries(
   collection, Sydney, ee.Reducer.mean(), 500)
```

```
    .setChartType('LineChart')
    .setOptions({
        title: 'Band Mean by Day of Year Across Years',
        hAxis: {title: 'Day of Year'},
        vAxis: {title: 'Lwe Thickness Mean (cm)'}
});

// Define a chart with a different series for each year in the sydney region.
// Can also define a chart to appear as a scatter rather than a line.
var series2 = ui.Chart.image.doySeriesByYear(
    collection, 'b1', Sydney, ee.Reducer.mean(), 500)
    .setChartType('ScatterChart')
    .setOptions({
        title: 'Sydney LWE Mean by Day of Year in Different Years',
        hAxis: {title: 'Day of Year'},
        vAxis: {title: 'Lwe Thickness Mean (cm)'}
});

// Define a chart with different series for each region, averaged by DOY.
// Output the natural syntax of GEE based on your code
var series3 = ui.Chart.image.doySeriesByRegion(
    collection, 'b1', regions, ee.Reducer.mean(), 500, ee.Reducer.mean(), 'label');

// Display the three charts.
print(series1, series2, series3);
```

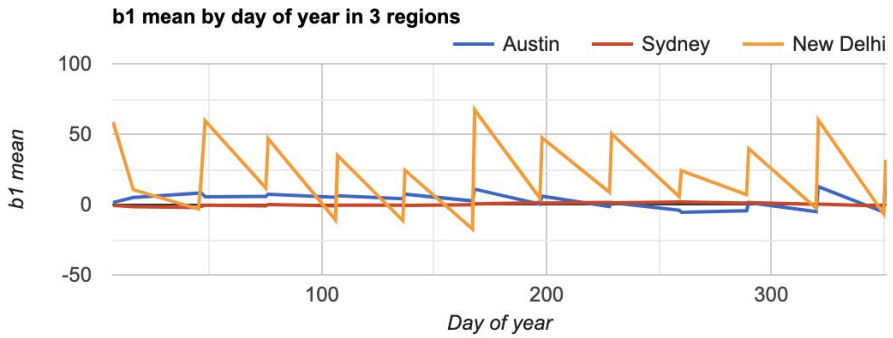
Such a script will produce an output of figure 12 below in the console.



(a)



(b)



(c)

Fig 12. (a) DOY Chart line chart with chosen axes, (b) DOY scatter chatter with chose axes, and (c) the natural GEE DOY chart produced

4.3.2. Time-series Chart

An equivalent chart that quickly compares various areas of interest is a time-series chart. This chart can draw from any location of interest or KML file.

```
// Time-series Chart
//
// This is a GRACE/GRACE-FO Image Collection that has an applied land mask

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Define a FeatureCollection: regions of the American West.
```

```
var regions = ee.FeatureCollection([
  ee.Feature(    // Austin
    ee.Geometry.Point(-97.733, 30.27), {label: 'Austin'}),
  ee.Feature(  // Sydney
    ee.Geometry.Point(151.197, -33.84), {label: 'Sydney'}),
  ee.Feature(  // NewDelhi
    ee.Geometry.Point(77.176, 28.648), {label: 'New Delhi'})
]);


// Import the image collection desired and filter the time span addressing
var water2007 = ee.ImageCollection('users/Rmjacobs28/GRACE/Land-Mask')
    .filterDate('2007-01-16', '2007-12-16')
    .select('b1');

// Create a time series chart.
var LWESeries = ui.Chart.image.seriesByRegion(
    water2007, regions, ee.Reducer.mean(), 'b1', 200, 'system:time_start', 'label')
        .setChartType('ScatterChart')
        .setOptions({
          title: 'Equivalent Water Thickness',
          hAxis: {title: 'Time of the Year'},
          vAxis: {title: 'Lwe Thickness (cm)'},
          lineWidth: 1,
          pointSize: 4
});

// Display.
print(LWESeries);
```

Such a script will produce an output of figure 13 below in the console.

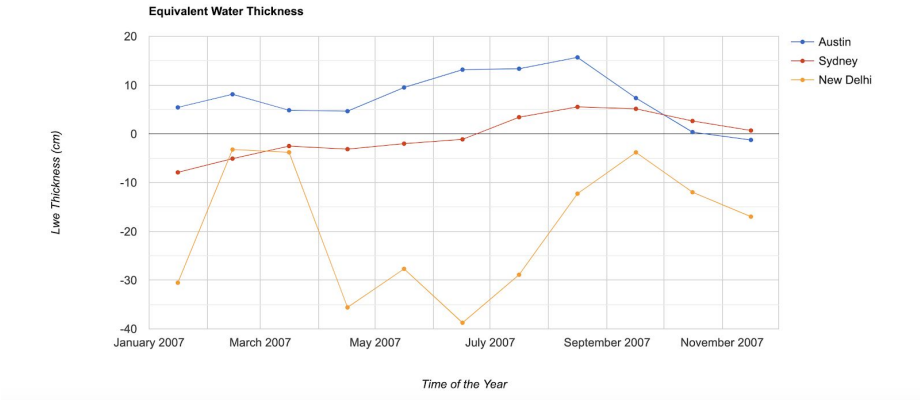


Fig 13. A time-series chart comparing equivalent liquid thickness in Austin, Sydney, and New Delhi during the year 2007

4.4. Creating User-Friendly Widgets

Similar to most computer programming languages, Google Earth Engine has generated an extensive repertoire of lingo. The "widget" is lingo in GEE and computing that means an application, or a component of an interface, that enables a user to perform a function. These helpful scripts of code may be added to most programs. This is presented in various forms such as text boxes, sliders, or panels. For the full listing of widgets, visit the Widgets Guide produced by Google Earth Engine.

4.4.1. Generating specific regions

```
// Generating Specific Regions

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Allows the user to quickly choose places of interest for the map to travel to
// This provides a helpful tool when examining data
// Four arbitrary picked places for the user to choose from
var places = {
  NewDelhi: [77.176, 28.648],
  Austin: [-97.733, 30.27],
  Sydney: [151.197, -33.84],
  GrandCanyon:[-112.8598, 36.2841]
};

var select = ui.Select({
  items: Object.keys(places),
  onChange: function(key) {
    Map.setCenter(places[key][0], places[key][1]);
  }
});

// Set a placeholder.
select.setPlaceholder('Choose a location...');

// Prints to the console for the user to operate
print(select);
```

Such a script will produce an output of figure 14 below in the console.

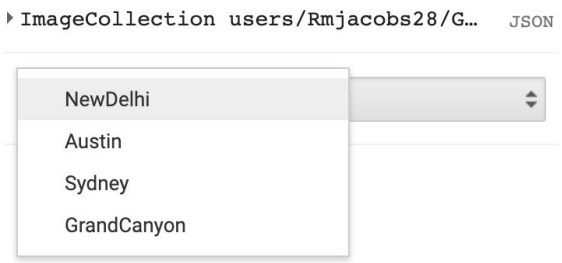


Fig 14. A user-friendly "widget" that allows the user to quickly zoom to a location and subsequently reduce work

4.4.2. Creating a Legend

```
// Creating a legend
//
// No images or Image Collections are imported so no map layers will be displayed in the
// interactive map

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Create visualization arguments (This scale allows lwe thickness to specify in which a deficit
is highlight in red hues and greater thickness in blue hues)
var viz = {
   'lt -30 cm':'8c2d04',
   '-30 to -25':'cc4c02',
   '-25 to -20': 'ec7014',
   '-20 to -17': 'fe9929',
   '-17 to -14': 'fec44f',
   '-14 to -11': 'fee391',
   '-11 to -8': 'fff7bc',
   '-8 to -5':  'ffffd4',
   '-5 to 5': 'f0f0f0',
   '5 to 8': 'edf8b1',
   '8 to 11': 'c7e9b4',
   '11 to 14': '7fcdbb',
   '14 to 17': '41b6c4',
   '17 to 20': '1d91c0',
```

```
    '20 to 25': '225ea8',
    '25 to 30': '0c2c84',
    'gt 30 cm': '081d58'
}

// Set position of panel
var legend = ui.Panel({
  style: {
    position: 'bottom-left',
    padding: '8px 15px'
  }
});

// Create legend title
var legendTitle = ui.Label({
  value: 'My Legend',
  style: {
    fontWeight: 'bold',
    fontSize: '18px',
    margin: '0 0 4px 0',
    padding: '0'
    }
});

// Add the title to the panel
legend.add(legendTitle);

// Creates and styles 1 row of the legend
var makeRow = function(color, name) {

    // Create the label that is actually the colored box.
    var colorBox = ui.Label({
      style: {
        backgroundColor: '#' + color,
        // Use padding to give the box height and width
        padding: '8px',
        margin: '0 0 4px 0'
      }
    });

    // Create the label filled with the description text
    var description = ui.Label({
      value: name,
      style: {margin: '0 0 4px 6px'}
    });
```

```
    // Return the panel
    return ui.Panel({
      widgets: [colorBox, description],
      layout: ui.Panel.Layout.Flow('horizontal')
    });
};

// Name of each legend input
var names =['lw to -30', '-30 to -25', '-25 to -20', '-20 to -17', '-17 to -14', '-14 to -11','-11 to -8',
'-8 to -5', '-5 to 5','5 to 8','8 to 11','11 to 14','14 to 17','17 to 20','20 to 25','25 to 30','gt 30 cm'];

// Palette with the colors
var palette =
['8c2d04','cc4c02','ec7014','fe9929','fec44f','fee391','fff7bc','ffffd4','f0f0f0','edf8b1','c7e9b4','7f
cdbb','41b6c4','1d91c0','225ea8','0c2c84','081d58'];

// Add the colors and names
for (var i = 0; i < 17; i++) {
  legend.add(makeRow(palette[i], names[i]));
  }

// Add legend to map (alternatively you can also print the legend to the console)
Map.add(legend);
```

Such a script will produce an output of figure 15 below in the interactive map.

Fig 15. A user defined legend

4.4.3. Masking an Image

```
// Masking GRACE Image
//
// Import the Image Collection you wish to view
// This is a GRACE/GRACE-FO Image Collection without an applied land mask

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

var GRACE = ee.ImageCollection('users/Rmjacobs28/GRACE/GRACE_GRACE-FO')
  .sort('system:time_start');

// Choose the month you wish to visualize
var image = GRACE.filterDate('2007-01-01','2007-12-31')

// Renames the bands properly output the proper name to the user.
var adjustBands = image.select(['b1'],['lwe_thickness'])
print(adjustBands)

var equivalentWaterThicknessCsr = adjustBands.select('lwe_thickness');

// Define the visualization parameters
// Without a defined palette, the map will print the data in a gradient in black and white.
var equivalentWaterThicknessCsrVis = {
  min: -30.0,
  max: 30.0,
};

Map.addLayer(
    equivalentWaterThicknessCsr, equivalentWaterThicknessCsrVis,
    'Equivalent Water Thickness CSR');

// Import the mask you wish to apply
var landmask =
ee.Image('users/howard_csr_utexas/CSR_GRACE_RL06_Mascons_v02_LandMask')


var median = image.filterDate('2007-01-01','2007-12-31').median();
// Select the land/water mask
var datamask = landmask.select('b1');

// Create a binary mask
// We use eq(1) to create a binary image in which all the pixels that do not have the value of 1
var mask = datamask.eq(1);
```

```
// Update the composite mask with the water mask
var maskedComposite = median.updateMask(mask);

// Upload the same month of data with the image filter applied
// This mask will make the area masked “transparent” to the viewer.
Map.addLayer(maskedComposite, equivalentWaterThicknessCsrVis, 'masked');
```

4.4.4. Printing an Image to Console

```
// Printing an Image to Console
//
// Import the Image Collection you wish to view and the land mask you want to apply
// The area of India is also defined

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

var landmask =
ee.Image("users/howard_csr_utexas/CSR_GRACE_RL06_Mascons_v02_LandMask"),
    India =
    /* color: #98ff00 */
    /* shown: false */
    /* displayProperties: [
      {
        "type": "rectangle"
      },
      {
        "type": "marker"
      }
    ] */
    ee.Geometry({
      "type": "GeometryCollection",
      "geometries": [
        {
          "type": "Polygon",
          "coordinates": [
            [
              [
                67.80230058730974,
                36.49755997729118
              ],
              [
                67.80230058730974,
                4.878971715253764
```

```
            ],
            [
              99.35503496230974,
              4.878971715253764
            ],
            [
              99.35503496230974,
              36.49755997729118
            ]
          ]
        ],
        "geodesic": false,
        "evenOdd": true
      },
      {
        "type": "Point",
        "coordinates": [
          84.017829803191,
          27.259170221135655
        ]
      }
    ],
    "coordinates": []
  }),
  geometry = /* color: #d63000 */ee.Geometry.Point([84.01791563387948,
27.26092507652443]);

// Import a Feature Collection with the outlines of countries
var worldcountries = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017');
// Filter the collection to show the natural boundaries of India
var filterCountry = ee.Filter.eq('country_na', 'India');
var bounds = worldcountries.filter(filterCountry);

// Import the collection you want to see
var filtered_grace = ee.ImageCollection('users/Rmjacobs28/GRACE/Land-Mask')
.filterDate('2009-07-01', '2009-07-31')


var grace_img = filtered_grace.toBands()
print(grace_img)

// Update the grace image to show only land
var maskedLand = grace_img.updateMask(landmask);
var clippedLand = maskedLand.clip(bounds)

// Define palette of colors for specific lwe thickness values
```

```
var sld_ramp_land =
  '<RasterSymbolizer>' +
    '<ColorMap type="ramp" extended="false" >' +
      '<ColorMapEntry color="#8c2d04" quantity="-30.0" label="lt -30 cm"/>' +
      '<ColorMapEntry color="#cc4c02" quantity="-25.0" label="-30 to -25" />' +
      '<ColorMapEntry color="#ec7014" quantity="-20.0" label="-25 to -20" />' +
      '<ColorMapEntry color="#fe9929" quantity="-17.0" label="-20 to -17" />' +
      '<ColorMapEntry color="#fec44f" quantity="-14.0" label="-17 to -14" />' +
      '<ColorMapEntry color="#fee391" quantity="-11.0" label="-14 to -11" />' +
      '<ColorMapEntry color="#fff7bc" quantity="-8.0" label="-11 to -8" />' +
      '<ColorMapEntry color="#ffffd4" quantity="-5.0" label="-8 to -5" />' +
      '<ColorMapEntry color="#f0f0f0" quantity="5.0" label="-5 to 5" />' +
      '<ColorMapEntry color="#edf8b1" quantity="8.0" label="5 to 8" />' +
      '<ColorMapEntry color="#c7e9b4" quantity="11.0" label="8 to 11" />' +
      '<ColorMapEntry color="#7fcdbb" quantity="14.0" label="11 to 14" />' +
      '<ColorMapEntry color="#41b6c4" quantity="17.0" label="14 to 17" />' +
      '<ColorMapEntry color="#1d91c0" quantity="20.0" label="17 to 20" />' +
      '<ColorMapEntry color="#225ea8" quantity="25.0" label="20 to 25" />' +
      '<ColorMapEntry color="#0c2c84" quantity="30.0" label="25 to 30" />' +
      '<ColorMapEntry color="#081d58" quantity="50.0" label="gt 30 cm" />' +
    '</ColorMap>' +
  '</RasterSymbolizer>';

// Add the source GRACE data from October to assist with map queries
Map.addLayer(clippedLand.sldStyle(sld_ramp_land), {}, 'SLD ramp Land');

// Create an empty image just specifically looking at India
var empty = ee.Image().byte()
// Paint the feature collection to show the border of India
var regionVis = empty.paint({
  featureCollection: bounds,
  width: 1
})
Map.addLayer(regionVis,{palette:'black'})

Map.centerObject(bounds, 6);

// Also add a terrain layer for user viewing
// Define the visualization arguments for terrain
var visArgs = {
  min: -40.0,
  max: 35.0,
};

// Update the visualization with data from Image Collection
var lweColVis = filtered_grace.map(function(img) {
```

```
  return img.visualize(visArgs);
});

var hillshade = ee.Terrain.hillshade(ee.Image('USGS/SRTMGL1_003')
  .multiply(1050))
  .clipToCollection(bounds);

// Creates a layer for terrain on the interactive map
var finalVisCol = lweColVis.map(function(img) {
  return hillshade
    .blend(img.clipToCollection(bounds).visualize({opacity: 0.6}))
});

Map.addLayer(finalVisCol)

// Update the GRACE lwe thickness image with the boundary of India for easy viewing
var image = regionVis.blend(clippedLand.sldStyle(sld_ramp_land))

// Print an image to console
print(ui.Thumbnail({
  image: image,
  params: {
    dimensions: '256x256',
    region: India,
    format: 'png'
  },
  style: {height: '300px', width: '300px'}
}));
```

Such a script will produce an output of figure 16 below in the console.

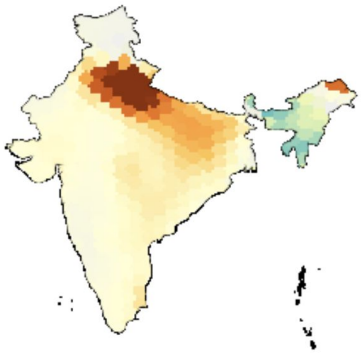


Fig 16. A printed image of India (September 2009) in console

4.4.5. Combining with other Satellite Data

One of the most important applications of Google Earth Engine, more than any other visualization platform, is its ability to map data from different satellites onto one map. From climate and weather, imagery to geophysical, GEE has created an extensive network of datasets. Searching through the collection is quick and easy by utilizing the “Search places and datasets” search bar at the top of the GEE code editor. From there the same principles of importing a dataset can be applied to any dataset available in GEE. Therefore, if it exists in the GEE platform, the user may import it into a script with the GRACE data to quickly compare between new sources of various information.

5. Troubleshooting and Support

5.1. Error Messages

As the user switches between various software platforms, error messages will naturally arise. The two error messages most commonly found when uploading TIFF files into GEE arise from poor file conversion.

The first error most commonly encountered is: “ValueError: time data ‘2002-01-01T00:00:00Z’ does not match format ‘%Y-%m-%d %H:%M:%S.” This error will most often occur during the conversion process of the NetCDF files to TIFF files. This is a result of the .nc file of the RL06 Mascon solutions Version 2 data. The error has two solutions: a python script that utilizes a new date and time stamp in this iso date format or utilizing ArcGIS. It is recommended to utilize the latter as ArcGIS contains a software package known as Arcpy that easily works within this iso date time format. Therefore, the user is not required to address this manually unless they are utilizing their own script.

The second error most often encountered is the following error in Google Earth Engine: “Projection for GRACE_LWE could not be determined. Make sure both CRS and affine transport are present.” If an error such as this is presented it demonstrates that a Coordinate Reference System in the NetCDF file has not been set. The CSR GRACE mascons are not represented on a standard coordinate system so can sometimes encounter this problem. The CRS transform that is required to properly upload into GEE is the coordinate system known as EPSG:4326, which is the WGS84, or World Geodetic System 1984, coordinate system. This error can easily be solved by once again utilizing ArcGIS. One of the more helpful applications is the use of applying or manipulating coordinate systems in ArcGIS. When uploading the NetCDF file into GEE, ArcGIS will automatically apply a coordinate system onto the GRACE metadata. In doing this, when the user properly exports from ArcGIS software the correct affine coordinate system will be applied without user correction.

Error messages addressing both ArcGIS and GEE are heavily documented. If other errors occur, consult both help guides and debugging tips at each respective website.

Appendix A: Bibliography

Save, H., S. Bettadpur, and B.D. Tapley (2016), High resolution CSR GRACE RL05 mascons, J. Geophys. Res. Solid Earth, 121, doi:10.1002/2016JB013007.

Save, Himanshu, 2020, "CSR GRACE and GRACE-FO RL06 Mascon Solutions v02", doi: 10.15781/cgq9-nh24.

“ArcGIS Pro Help.” *ArcGIS Pro Help-ArcGIS Pro | Documentation*, pro.arcgis.com/en/pro-app/help/main/welcome-to-the-arcgis-pro-app-help.htm.

“Get Started with Earth Engine | Google Earth Engine.” *Google*, Google, developers.google.com/earth-engine/guides/getstarted.

Appendix B: ArcGIS Pro Python Code

```
#Import system modules
import arcpy                    # the main ArcGIS library
import os                       # python module for manipulation of files and directories
# from datetime import datetime as dt # the submodule of the datetime module needed in this script
arcpy.env.overwriteOutput = True    # arcpy setting permits output files to be overwritten


# source NetCDF file
# change if named differently
nc_mascons = "CSR_GRACE_GRACE-FO_RL06_Mascons_all-corrections_v02.nc"

# You may need to change paths in next code block to reflect your own directory structure and filenames
# input and output directories
# note use of r before string to indicate how backlashes should be interpreted as raw string
nc_dir = r"E:\Projects\GRACEFO_Riley\NetCDF"
tiff_dir = r"E:\Projects\GRACEFO_Riley\GeoTIFFs"

# Next we want to construct full path names for the input NetCDF files.
# full paths
src_mascons = os.path.join(nc_dir, nc_mascons)
src_landmask = os.path.join(nc_dir, nc_landmask)
src_watermask = os.path.join(nc_dir, nc_watermask)

# determine NetCDF time series time step
# Description: Create a table view from a netCDF file.
# Set local variables
variable = "time_bounds;time" # NetCDF variables
outTableView = "GRACE_RL06_Mascons_TimeTable" # in memory table view
rowDimension = "time"
dimensionValue = None
valueSelectionMethod = "BY_VALUE"

# Execute MakeNetCDFTableView. The output exists in memory only.
arcpy.md.MakeNetCDFTableView(src_mascons, variable, outTableView, rowDimension, dimensionValue,
valueSelectionMethod)

# Extract time steps from table view
# access data in table view by constructing a cursor
cursor = arcpy.da.SearchCursor(outTableView, ["time", "time_bounds"])

# build two lists - one of time in form of datetime, the other of time step counts
grace_dates = list()
grace_time_steps = list()

# step through the rows in the cursor and append values to appropriate list
for row in cursor:
    grace_dates.append(row[0])
    grace_time_steps.append(row[1])
# check the last element in each list
print(grace_dates[-1]) # note that datatype is datetime.datetime
print(grace_time_steps[-1]) # note that data type is float
```

```
# Generate raster layers from NetCDF mascon file and export to GeoTIFF
'''
Armed with date information, our next step is to create a raster layer representing each time step in the GRACE mascon
NetCDF file. The syntax for the layer creation command is:

arcpy.MakeNetCDFRasterLayer(in_netCDF_file, variable, x_dimension, y_dimension, out_raster_layer,
{band_dimension}, {dimension_values}, {value_selection_method}, {cell_registration}) # optional parameters in {}

We had tested geoprocessing tool output in ArcGIS Pro and know that we need to reference all variables accept for
{band_dimension} which is not required since each raster output consists of a single band.

We also know that we are going to loop through the datetime values to construct output file name and dimension values.
'''

# Set local 'universal' variables
nc_variable = "lwe_thickness"
XDimension = "lon"
YDimension = "lat"
bandDimension = "" # not pertinent to our output
valueSelectionMethod = "BY_VALUE"
cellRegistration = "CENTER"

# create a spatial reference to the World Geodetic System 1984 coordinate system
sr = arcpy.SpatialReference("WGS 1984")

# prepare empty lists for output file names
# Not used in this script but useful in the case of land and water mask creation & application
raster_layer_list = list()
tiff_list = list()

# loop through grace datetime.datetime values to construct output Raster layer names and
# dimension values in the proper format
cnt = 0 # counter variable set for testing purposes
for grace_date in grace_dates:
    cnt+=1
    # print(cnt, grace_date.strftime("%Y%m%d")) # test print
    # strftime is method on datetime.datetime object used to stringify date
    outRasterLayer = "lwe_thickness_{}".format(grace_date.strftime("%Y%m%d"))
    raster_layer_list.append(outRasterLayer)
    # print(outRasterLayer) # test print

    tiff_name = "lwe_thickness_{}.tif".format(grace_date.strftime("%Y%m%d"))
    out_tiff = os.path.join(tiff_dir, tiff_name)
    # print(out_tiff) # test print
    tiff_list.append(out_tiff)

    # sample format goal "time '04/18/2002 12:00:00 AM'"
    dimensionValues = "time '{}'".format(grace_date.strftime("%m/%d/%Y %H:%M:%S %p"))
    #if cnt % 20 == 0:
    #    print(outRasterLayer, dimensionValues) # test print
    try:
        # Execute MakeNetCDFRasterLayer
        arcpy.MakeNetCDFRasterLayer_md(src_mascons, nc_variable, XDimension,
```

```
                        YDimension, outRasterLayer, bandDimension,
                        dimensionValues, valueSelectionMethod, cellRegistration)
    # Export raster layer to GeoTiff
     with arcpy.EnvManager(outputCoordinateSystem= sr): # set coordinate system in tool environment
        arcpy.management.CopyRaster(outRasterLayer, out_tiff, '', None, "-3.402823e+38",
                                    "NONE", "NONE", "32_BIT_FLOAT", "NONE", "NONE", "TIFF",
                                    "NONE", "CURRENT_SLICE", "NO_TRANSPOSE")
  except Exception:
     e = sys.exc_info()[1]
     print(e.args[0])

print("DONE!. Last time step is {}, at "{}").format(outRasterLayer, dimensionValues))
```