Received: 29 June 2022 | Revised: 22 December 2022 | Accepted: 26 January 2023

DOI: 10.1111/tgis.13031

Transactions in GIS WILEY

RESEARCH ARTICLE

Integrating GRASS GIS and Jupyter Notebooks to facilitate advanced geospatial modeling education

Caitlin Haedrich[1] | Vaclav Petras[1] | Anna Petrasova[1] | Stefan Blumentrath[2] | Helena Mitasova[1,3]

[1]Center for Geospatial Analytics, North Carolina State University, Raleigh, North Carolina, USA

[2]Norwegian Institute for Nature Research, Trondheim, Norway

[3]Department of Marine, Earth, and Atmospheric Sciences, North Carolina State University, Raleigh, North Carolina, USA

Correspondence
Caitlin Haedrich, Center for Geospatial Analytics, North Carolina State University, Raleigh, NC 27695, USA.
Email: caitlin.haedrich@gmail.com

Funding information
Google Summer of Code; North Carolina State University; OSGeo GRASS GIS

Abstract

Open education materials are critical for the advancement of open science and the development of open-source software. These accessible and transparent materials provide an important pathway for sharing both standard geospatial analysis workflows and advanced research methods. Computational notebooks allow users to share live code with in-line visualizations and narrative text, making them a powerful interactive teaching tool for geospatial analytics. Specifically, Jupyter Notebooks are quickly becoming a standard format in open education. In this article, we introduce a new GRASS GIS package, `grass.jupyter`, that enhances the existing GRASS Python API to allow Jupyter Notebook users to easily manage and visualize GRASS data including spatiotemporal datasets. While there are many Python-based geospatial libraries available for use in Jupyter Notebooks, GRASS GIS has extensive geospatial functionality including support for multi-temporal analysis and dynamic simulations, making it a powerful teaching tool for advanced geospatial analytics. We discuss the development of `grass.jupyter` and demonstrate how the package facilitates teaching open-source geospatial modeling with a collection of Jupyter Notebooks designed for a graduate-level geospatial modeling course. The open education notebooks feature spatiotemporal data visualizations, hydrologic modeling, and spread simulations such as the spread of invasive species and urban growth.

1 | INTRODUCTION

The open science movement argues that science should be transparent and involve the free sharing of knowledge, methods, data, and code, thereby making science accessible to all (Petras et al., 2015; Rocchini & Neteler, 2012; Watson, 2015). There are six accepted pillars of open science: open data, open access, open methodology, open source, open peer review and open education (Watson, 2015). In many ways, these are related; increasing open education resources increases the accessibility of open-source software, which in turn, supports open research (McKiernan et al., 2016; Rey, 2018). Open science developed in response to calls within the scientific community for greater transparency and reproducibility (Morin et al., 2012; Rocchini & Neteler, 2012; Stodden et al., 2013; Watson, 2015). These calls for transparency and reproducibility in the sciences are not new; the advent of the scientific journal was, after all, in response to those needs (Páez, 2021). However, centuries later, science still struggles to accomplish this. A study of 204 computational articles in Science from 2011 to 2012 found that only 26% had reproducible results (Stodden et al., 2018).

Open education, one of the pillars of open science, refers to open education resources that are freely available online and have open content licensing, allowing the materials to be used, replicated and modified (Belgiu et al., 2015; Weller et al., 2018). These materials play an important role in broader open education practices (Cronin, 2017), documenting open-source software, increasing the accessibility of science and providing resources for instructors to build curriculums that teach open science practices.

In the geospatial sciences, there have been several recent studies presenting open education resources using open-source software and demonstrating their effectiveness in the classroom. For example, Ciolli et al. (2017) present an open-source geospatial curriculum developed and employed at Italian universities for over two decades. Petras et al. (2015) discuss materials developed at North Carolina State University for teaching advanced geospatial modeling with open-source software. Beyond the classroom, there are also many examples of general open education resources that have become extremely popular. For example, users of QGIS and Python may be familiar with the tutorials created by Anita Graser (2022), Klas Karlsson (2022) or Hans van der Kwast (2022a). Geospatial data scientists may have encountered University of Colorado Boulder's Earth Lab Python and R courses and tutorials (2022) or the courses available on GIS Open CourseWare (2022).

As the open science movement gathers momentum and open education resources multiply, computational notebooks, and Jupyter Notebooks in particular, have become popular across many disciplines (Pimentel et al., 2019; Rowe et al., 2020), including data science (Lasser et al., 2021; Perkel, 2018), bio-informatics (Engelberger et al., 2021) and earth science (Committee on Earth Observation Satellites, 2021; Wagemann et al., 2022). This format mixes live code with in-line code output and narrative text in a single document. Their rise reflects their ability to support open science: notebooks are a powerful teaching tool and an effective way to share scientific methods, supporting open research, open software (Rowe et al., 2020) and open education (Reades, 2020). For sharing scientific methods, notebooks can be a standalone presentation of work (Rey, 2018; Rowe et al., 2020): introduction, live code for methods, in-line results and narrative text discussing the results. Presenting work through computational notebooks also offers a transparent explanation of methods where users can see the inner workings of the analysis, test processing, and modify the inputs. In educational settings, notebooks offer live code with in-line results and narrative text to illustrate and document the code.

In geospatial science, scripting has become an essential skill in the field (Anbaroğlu, 2021; Brunsdon & Comber, 2021; Etherington, 2016; Palomino et al., 2017). Recent open education resources reflect this new demand for coding expertise: scripting and coding are increasingly taught in geospatial courses (Anbaroğlu, 2021; Bowlick et al., 2017; Petras et al., 2015; Reades & Rey, 2021). With the rise of computational notebooks and the demand for scripting in the geospatial sciences, it is unsurprising that Jupyter Notebooks are becoming popular for teaching geospatial science. Several studies have already presented open educational resources that teach geospatial analysis through computational notebooks. For example, Reades (2020) present an extensive open-source collection of Jupyter Notebooks for teaching spatial analysis at the undergraduate level. Arribas-Bel (2019) presents a Jupyter-based Geospatial Data Science course that is offered at the University of Liverpool.

There is a large ecosystem of Python packages for geospatial analysis that are commonly used in Jupyter Notebooks, such as PySAL (Rey & Anselin, 2007), GeoPandas (Jordahl et al., 2020), Rasterio (Gillies et al., 2013), and GDAL (Rouault et al., 2022), including many open-source GIS software with Python APIs. While these Python packages can be used in Jupyter Notebooks as well as in plain scripts, additional extensions of the projects are specifically developed for the Jupyter Notebook environment to simplify setup or to take advantage of its interactive features. For example, Whitebox and QGIS both have Python APIs (giswqs/whitebox-python, 2022; QGIS Development Team, 2022) and third-party extensions for use in Jupyter Notebooks (giswqs/whiteboxgui, 2022; QGIS, 2022).

GRASS GIS, an open-source software, has an extensive geospatial toolset including support for multi-temporal analysis and dynamic simulations, making it a powerful teaching tool for advanced geospatial analytics. Because of its long history and large contributing community, numerous advanced analytical and modeling tools have been implemented and continue to be implemented (Bornaetxea & Marchesini, 2022; Cimburova & Blumentrath, 2022). GRASS GIS also has a well-documented and extensively used Python API, making it well-suited for geospatial programming. Finally, GRASS GIS has already been used for teaching advanced geospatial modeling and simulation by means of step-by-step tutorials that let students run GRASS tools with either the GRASS Graphical User Interface (GUI), Python API, GRASS console or Bash (Petras et al., 2015). Migrating these tutorials to Jupyter Notebooks would make them interactive, allowing students to run, modify, and explore the code while reading explanatory text.

However, several barriers exist to using GRASS GIS easily in Jupyter Notebooks. Although GRASS GIS has a Python API, GRASS session management and data visualization are not intuitive in Jupyter Notebooks. Launching GRASS GIS and accessing and visualizing data are complicated, requiring users to modify several environment variables. The static visualizations created by the GRASS Python API do not offer the interactivity that many users expect when working with geospatial data: the images do not allow users to zoom in or out or toggle between layers.

In this article, we introduce a new GRASS package, `grass.jupyter`, that enhances the existing GRASS Python API to allow Jupyter Notebook users to easily manage and visualize data including spatiotemporal datasets. The package also allows GRASS users to take advantage of Jupyter Notebooks interactivity and portability. We show how `grass.jupyter` could be used to facilitate teaching open-source geospatial modeling with a collection of Jupyter Notebooks designed for a graduate-level geospatial modeling course. The open education notebooks feature spatiotemporal data visualizations, hydrologic modeling, and spread simulations including the spread of invasive species and urban growth.

2 | BACKGROUND

2.1 | GRASS GIS

GRASS GIS is an open-source Geographic Information System (GIS) software that provides powerful raster, vector, and geospatial processing tools, accessible through a GUI, Python API, command-line API, and integration with QGIS and R (Landa et al., 2022; Neteler & Mitasova, 2013). Tools, which are referred to as modules in the GRASS GIS documentation, include advanced terrain and hydrological modeling (Harmon et al., 2019; Jasiewicz & Stepinski, 2013; Mitasova et al., 2004), satellite and aerial imagery processing (Rocchini, Petras, Petrasova, Chemin, et al., 2017; Rocchini, Petras, Petrasova, Horning, et al., 2017), point cloud processing (Petras et al., 2017), visualization of raster and vector data and, clustering and classification algorithms including machine-learning methods (Flasse et al., 2021;

Koreň et al., 2021; Silver et al., 2019). For multi-temporal raster and vector data, GRASS GIS provides a temporal framework with associated processing and visualization tools (Gebbert & Pebesma, 2017). It is also optimized for efficient computing in high-powered computing environments with many cores, making it an ideal software for large geospatial analysis (Metz et al., 2017).

After its initial development at the US Army Corps of Engineers from 1982 to 1995 (Neteler & Mitasova, 2013), GRASS GIS development has become an international collaborative effort between private business, academia, and individuals. Several mailing lists provide forums for developers and users alike to discuss technical solutions for geospatial problems, future software features, software releases, bugs and to troubleshoot issues. The source code is hosted on GitHub and licensed under the GNU General Public License. Users can install GRASS GIS from the OSGeo-hosted website (GRASS GIS, 2022a).

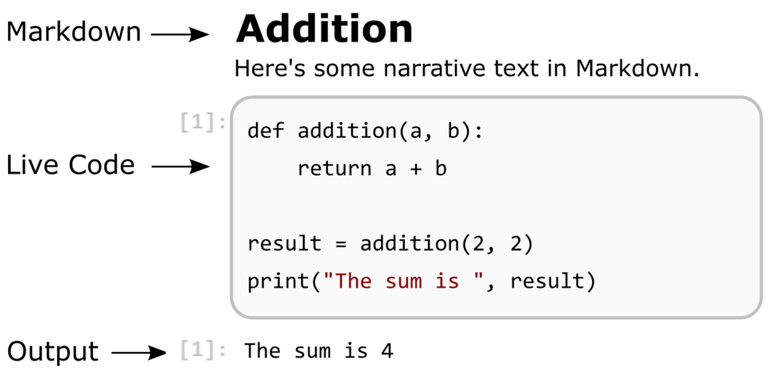
The core of GRASS GIS processing is a library of tools, also called modules, that are divided into families by purpose. The general nomenclature is a single letter declaring a tool's family membership followed by a dot and the name of the tool. For example, `r.mapcalc` belongs to the raster family and is a raster calculator. Other families include g for general tools, v for vectors, d for display, i for image processing, t for temporal framework tools, db for database commands, and r3 for 3D raster tools.

To achieve accurate and consistent results, GRASS GIS requires data to be imported into its native data storage format and reprojected to a common Coordinate Reference System (CRS) for each project. This approach ensures all data are in the same CRS. By doing this up front, users avoid CRS mismatch issues later on in their workflow. Projects, also called locations, are defined by their CRS and contain sub-projects, called mapsets, that organize sub-regions, different workflows or other aspects of the project. Users also set a computational region defined as cell resolution and the geographic extent over which raster tools are applied. This is particularly useful when testing workflows: users can run an analysis on a small subset of data without cropping the data first.

2.2 | Jupyter Notebooks

Project Jupyter provides open-source software for interactive computing (Project Jupyter, 2022). The most prominent product of Project Jupyter is the Jupyter Notebook, a shareable web document that combines live code with output and narrative text. In 2014, Project Jupyter began as a spin-off of the IPython Project which had its own IPython Notebooks. Project Jupyter formally introduced the Jupyter Notebook in 2016 (Kluyver et al., 2016; Project Jupyter, 2022). Since then, it has quickly become the format of choice for data scientists to explore data and share analyses (Perkel, 2018). Even in the first 2 years since its introduction, there have been over 1.4 million notebooks added to GitHub (Pimentel et al., 2019). Jupyter Notebook is not the first computational notebook introduced; R Markdown along with mathematics software Mathematica, SageMath, and Maple all offer interactive notebooks that mix text with live code (Kluyver et al., 2016). However, one of the big advantages of the Jupyter Notebook is that it supports a plethora of programming languages beyond its original three: Julia, Python, and R (in fact, this was the original naming inspiration JU-PYT-R) (Kluyver et al., 2016). As of May 2022, there are over 100 kernels available for Jupyter Kernels - jupyter/jupyter Wiki (2022). The default kernel installed with a basic installation of Jupyter Notebook remains the IPython kernel.

Fundamentally, Jupyter Notebooks are JSON documents with `.ipynb` file extension. The notebooks consist of cells that can contain either code, Markdown, or unformatted/raw text (see Figure 1). Users select the cell type then run, modify, and re-run code cell-by-cell. The notebook is accessed through a web browser, making the interface consistent whether run locally or remotely (Kluyver et al., 2016). Notably, the notebooks do not compute or run within the browser application, just the user interface is in the browser; the computation occurs in a Jupyter kernel, running separately. With this configuration, notebooks can access other software as long as it is accessible where the kernel is running. Notebooks can be converted to a variety of formats including PDFs, HTML and LaTeX. The saved or exported notebooks can contain code outputs, making them an ideal format for sharing scientific methods and results.



Addition

Here's some narrative text in Markdown.

```
[1]: def addition(a, b):
         return a + b

     result = addition(2, 2)
     print("The sum is ", result)
```

```
[1]: The sum is 4
```

FIGURE 1 Illustration of Jupyter Notebook contents. Jupyter Notebooks contain a mix of markdown text, live code and code output.

Since the introduction of Jupyter Notebooks, a whole ecosystem of tools has been developed to support the sharing capabilities of Jupyter Notebooks. JupyterLab expands the original notebook viewer to include file browsing, multiple windows, and an extension library (Project Jupyter, 2022). Binder is an open-source service that hosts GitHub repositories containing notebooks as executable notebooks running in the cloud, allowing users to share their analyses with anyone through a single link (Jupyter et al., 2018). Similarly, Google Colab, Code Ocean and CoCalc allow users to edit and execute the notebooks in the cloud (CoCalc – Run Python Online, 2022; Code Ocean, 2022; Google LLC, 2022). BinderHub and JupyterHub let users configure their own JupyterLab services (Project Jupyter, 2022).

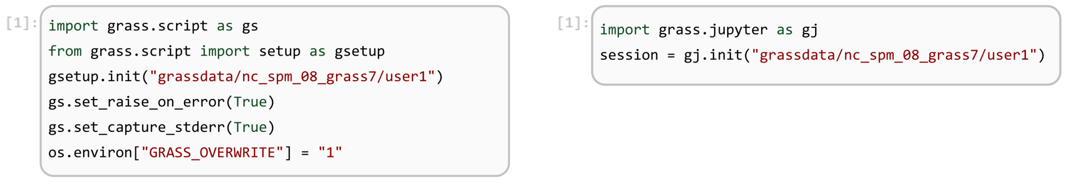
3 | SOFTWARE ARCHITECTURE

`grass.jupyter` is a Python package. It is a subpackage of the GRASS Python library which is a component of the GRASS GIS software. `grass.jupyter` depends on several additional packages: folium (Filipe et al., 2022), ipywidgets (2022), IPython (Pérez & Granger, 2007), and Pillow (Murray et al., 2022). The `grass.jupyter` package was developed with significant collaboration and feedback from the GRASS user community, facilitated through the GRASS developer mailing lists and Google Summer of Code mentorship. The package was also tested in a classroom setting where student feedback was solicited, especially when refining API naming.

To use `grass.jupyter`, GRASS GIS is launched within a notebook by adding the GRASS Python library (containing `grass.script` and `grass.jupyter`) on the path, importing it, and then starting a GRASS session in a specific mapset. The `grass.jupyter` package was released with GRASS GIS version 8.2 (Landa et al., 2022) and contains two primary components, discussed in detail below: GRASS session management tools and data visualization tools (`Map`, `Map3D`, `InteractiveMap`, and `TimeSeriesMap`).

3.1 | GRASS GIS session management

While GRASS sessions can be created from a notebook in the same way as from a Python script, there are differences between the expected behavior of a notebook and a script that require additional customizations (Figure 2). First, when calling GRASS tools from a notebook, the notebook should raise a Python exception when an error occurs instead of exiting the program, as would be appropriate for scripts. Second, standard error output of a subprocess is not visible in the notebook by default. The `grass.script` library needs to capture standard error output from GRASS tools that are called subprocesses and raise the error messages as Python exceptions. Third, it is generally expected that the whole notebook, as well as individual cells, can be executed repeatedly. However, by default



```
[1]: import grass.script as gs
     from grass.script import setup as gsetup
     gsetup.init("grassdata/nc_spm_08_grass7/user1")
     gs.set_raise_on_error(True)
     gs.set_capture_stderr(True)
     os.environ["GRASS_OVERWRITE"] = "1"
```

```
[1]: import grass.jupyter as gj
     session = gj.init("grassdata/nc_spm_08_grass7/user1")
```

FIGURE 2 Comparison of GRASS session initiation with `grass.script` package (left) and with `grass.jupyter` package (right).

GRASS tools do not overwrite existing data. To allow repeated cell execution an environmental variable is set globally to allow overwriting by all GRASS tools.

To prevent users from having to configure these behaviors, we wrapped the configuration in the `grass.jupyter.init` function, simplifying the launch as shown in Figure 2. Additionally, the `init` function returns a session object which allows users to switch between mapsets using the switch method. The session is automatically terminated when the notebook is closed or the kernel is ended.

3.2 | Visualization classes

The primary functionality of `grass.jupyter`, besides session management, is easy visualization of GRASS data. The `grass.jupyter` package provides four different visualization classes: `Map`, `Map3D`, `TimeSeriesMap`, and `InteractiveMap`. In each case, a visualization is created by instantiating the class, adding one or more datasets, and calling the class' `show` method that returns a map in the form of a static image, animated series of images or an interactive HTML widget to be rendered in the notebook. An overview of the package architecture is shown in Figure 3. Each of the visualization classes uses a region manager to control the extent rendered, as described in Section 3.2.1.

3.2.1 | Region managers

Region managers are internal objects of `grass.jupyter` visualization classes that provide a more fine-grained control of the displayed geographic extent and raster resolution using the concept of computational regions (Section 2.1). By default, the rendered extent matches the extent of the first layer or first space–time dataset added to a map. Alternatively, the `use_region=True` option passed into the visualization class' constructor allows users to set the rendered extent to the current computational region and `saved_region="myregion"` to a region previously saved by `g.region`. Visualization classes that return an image or a series of images adjust the default output image size of 600 × 400 pixels to match the aspect ratio of the given geographic extent. Different image sizes can be set by `width` and `height` parameters. The described region manager default behavior was designed to deliver results with expected extent and size suited for a notebook environment.

3.2.2 | Map

Map provides static, PNG image renderings of GRASS data (Figure 4). Although this is possible using the `grass.script` package, `grass.jupyter` encapsulates the GRASS rendering system so it is more intuitive and requires no knowledge of the rendering system (Figure 5). Without `grass.jupyter`, users must manually modify several environmental variables, including the filename and file path of the output image. If the user creates multiple renderings, the image file path has to be changed in the environmental variable with each new image. Alternatively, users could overwrite the file with each new visualization; in practice, this requires users to run `d.erase` before creating a new

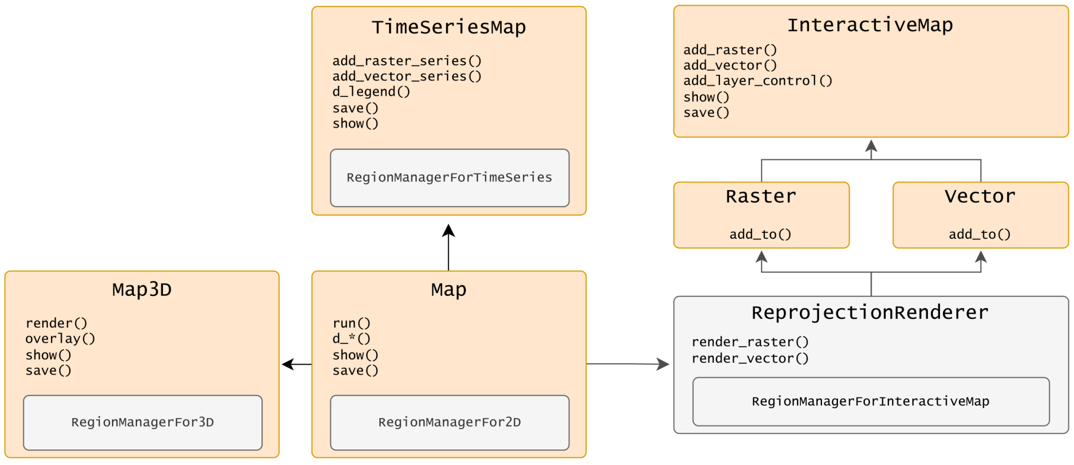
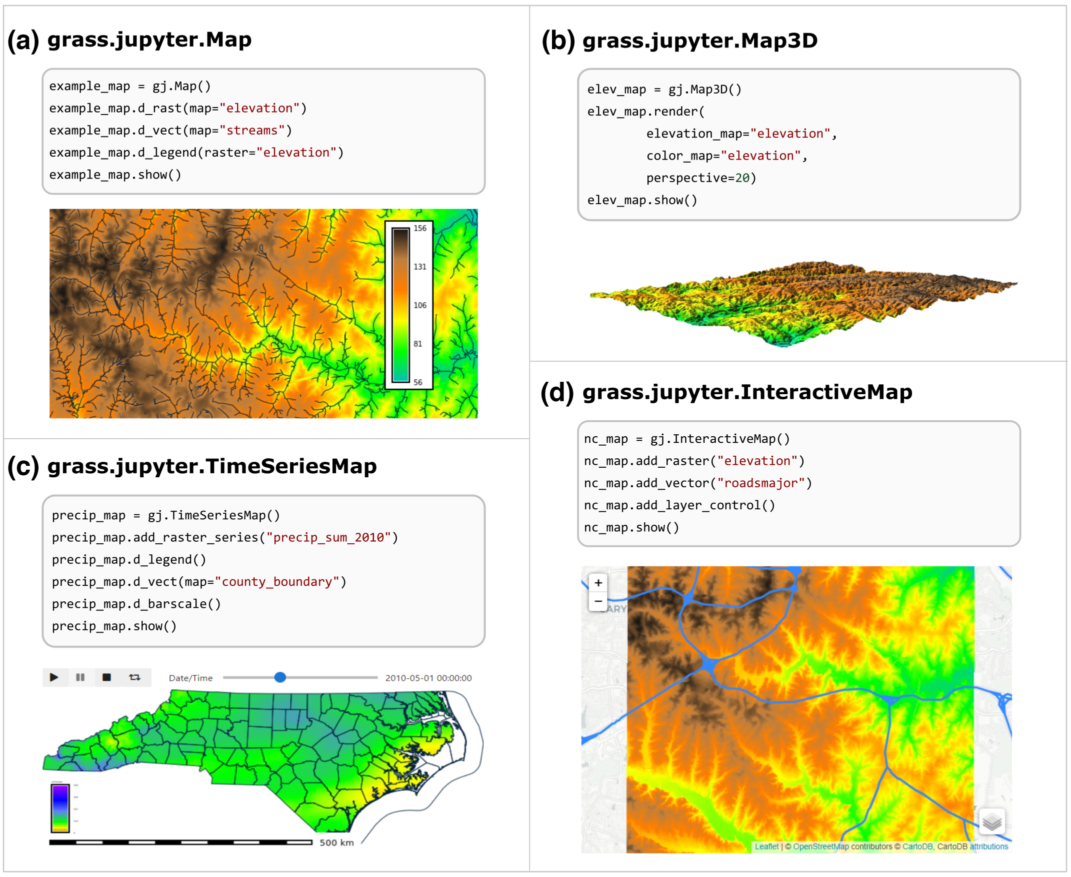


FIGURE 3 Diagram of `grass.jupyter` software showing private classes in gray and public classes and functions in orange. Arrows indicate composition for public classes and classes that are used by more than one class. Common class methods are listed within each box.


156
131
106
81
56
Date/Time
2010-05-01 00:00:00
500 km
+
−
Leaflet | © OpenStreetMap contributors © CartoDB, CartoDB attributions

(a) grass.jupyter.Map

```
example_map = gj.Map()
example_map.d_rast(map="elevation")
example_map.d_vect(map="streams")
example_map.d_legend(raster="elevation")
example_map.show()
```

(b) grass.jupyter.Map3D

```
elev_map = gj.Map3D()
elev_map.render(
        elevation_map="elevation",
        color_map="elevation",
        perspective=20)
elev_map.show()
```

(c) grass.jupyter.TimeSeriesMap

```
precip_map = gj.TimeSeriesMap()
precip_map.add_raster_series("precip_sum_2010")
precip_map.d_legend()
precip_map.d_vect(map="county_boundary")
precip_map.d_barscale()
precip_map.show()
```

(d) grass.jupyter.InteractiveMap

```
nc_map = gj.InteractiveMap()
nc_map.add_raster("elevation")
nc_map.add_vector("roadsmajor")
nc_map.add_layer_control()
nc_map.show()
```

FIGURE 4 Visualization classes available in `grass.jupyter`: `Map` for PNG image renderings (a), `Map3D` for 3D PNG image renderings (b), `TimeSeriesMap` for creating animations of space-time datasets (c), and `InteractiveMap` for creating interactive HTML maps with folium (d).

[1]:
```
os.environ["GRASS_FONT"] = "sans"
os.environ["GRASS_RENDER_IMMEDIATE"] = "cairo"
os.environ["GRASS_RENDER_FILE_READ"] = "TRUE"
os.environ["GRASS_LEGEND_FILE"] = "legend.txt"
os.environ["GRASS_RENDER_FILE"] = "map.png"

gs.run_command("d.erase")
gs.run_command("d.rast", map="lakes_buff")
gs.run_command("d.vect", map="streams")
gs.run_command("d.legend", raster="elevation")
Image(filename="map.png")
```

[1]:
```
example_map = gj.Map()
example_map.d_rast(map="elevation")
example_map.d_vect(map="streams")
example_map.d_legend(raster="elevation")
example_map.show()
```

FIGURE 5 Code comparison for creating an in-line PNG image rendering with `grass.script` package (left) and with `grass.jupyter` package (right).

rendering. Similar problems exist for legend creation. The `grass.jupyter` package solves these issues by creating a temporary directory for each instance of `Map` where the image file and legend text file are stored, thus creating an independent image for each instance of `Map`.

The second advantage of `grass.jupyter.Map` over the bare GRASS rendering system, accessed through `grass.script`, is the intuitive and consistent naming and API; all of the `grass.jupyter` visualization classes use a similar syntax for methods (`show`, `save`, and shortcuts to display family tools). After initiating an instance of the class, users call GRASS display tools to add elements to the rendering. This is done by calling the tool as a method of the instance and replacing the dot with the underscore as shown in Figure 4 (e.g., `Map.d_rast` would call the d.rast tool). `Map` does not implement a method for each display tool, rather it implements the Python magic method `_ _getattr_ _` to parse the GRASS tool from the intercepted attribute and run the tool with `grass.script`. We chose this strategy because it avoids the need to wrap tools separately and will require less maintenance as the GRASS display library evolves.

By default, image rendering uses the resolution of the first raster added and resamples the raster to fit the given image size. As a result, high-resolution rasters can be rendered very quickly and use less computer memory.

3.2.3 | Map3D

`Map3D` renders PNG images of 3D perspective views (Figure 4). The `Map3D` has two main components: the `render` method, which creates the 3D surface image, and the overlay attribute, which renders overlays, such as a legend or title. The render method calls the `m.nviz.image` tool that wraps GRASS's 3D rendering engine to write static images. The overlay attribute accesses an instance of `Map`, allowing users to add elements to the image file using any command from the display family of GRASS tools. `Map3D`'s syntax is consistent with other `grass.jupyter` visualization classes, allowing `grass.jupyter` users to create 3D perspective views without having to use the `m.nviz.image` tool directly.

3.2.4 | TimeSeriesMap

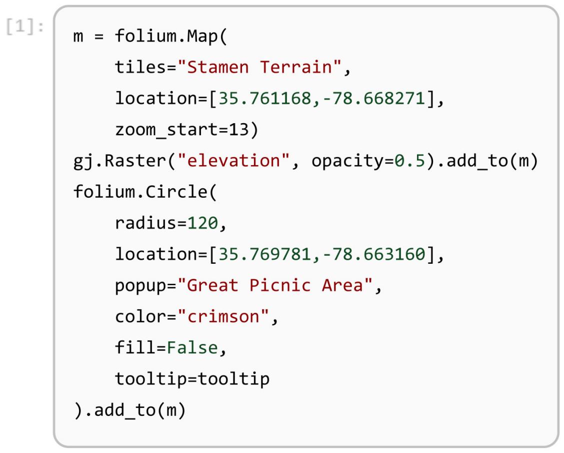
GRASS GIS offers a library of tools that support the analysis and visualization of space–time datasets (Gebbert & Pebesma, 2017). These space–time datasets contain a series of raster or vector layers with associated timestamps. Although GRASS GIS does have a dedicated GUI tool for creating animations of space–time datasets and exploring the data with a time slider, the tool is inaccessible from Jupyter Notebooks. `grass.jupyter` provides this functionality with `TimeSeriesMap`, a class that allows users to build interactive animations of space–time datasets. Using ipywidgets, an interactive HTML widget library designed for Jupyter and the IPython Kernel, `TimeSeriesMap` renders each time step in the space–time dataset using the `grass.jupyter.Map` class then displays them with a Play widget (play, pause, loop button) and a `SelectionSlider` widget for the timestamp. Users add space–time datasets to the `TimeSeriesMap` instance with `add_raster_series` and `add_vector_series` for raster and vector datasets, respectively. An example is shown in Figure 4.

One of the design challenges we faced in creating animations for space–time datasets is handling variable time steps: these datasets require layers to be shown at irregular intervals along the selection slider and it is unclear what the expected behavior would be during long temporal gaps in data. To handle this, we use the `gran` method of the `t.rast.list` and `t.vect.list` tools to sample the data at the temporal granularity of the dataset, returning a list with evenly spaced timestamps and the name of the layer corresponding to each timestamp and `NULL` for steps without data. Then, the `gap_fill` Boolean parameter allows users to either fill the `NULL` steps with the previous layer (`gap_fill=True`) or render an empty image (`gap_fill=False`). This approach ensures that distances on the slider bar are proportional to time, even in the case of an irregularly-spaced time series. Rendering an image for each step in the time series can be quite time-consuming so `TimeSeriesMap` renders base layers once and then copies the file for each step, avoiding the need to render the base layers repeatedly.

The syntax for `TimeSeriesMap` was designed to be consistent with `Map` and `Map3D`: GRASS tools are called via class attribute parsing and the order the tools are called reflects the order in which they are rendered. For example, to create an animation of an inundation flood over a static elevation raster, users would first call `d_rast` for the elevation then `add_raster_series` to add the inundation space–time dataset. The `show` method displays the `TimeSeriesMap` and save writes a GIF animation of the `TimeSeriesMap` instance.

3.2.5 | GRASS-folium integration and InteractiveMap

Interactivity, such as zooming in and out or toggling between layers, is particularly useful for exploring geographic data. Users, especially those switching to Jupyter Notebooks from the GRASS GUI, may expect interactivity when visualizing geographic data. The `grass.jupyter` package provides this through direct integration with folium, a Leaflet library for Python (Figure 6), and the `InteractiveMap` class (Figure 4). Although there are several open-source geospatial mapping libraries that can be integrated into Jupyter Notebooks, we selected folium, a widely popular Python library built on top of Leaflet.js that covers all our mapping needs. folium creates interactive HTML maps that can be embed-


```
[1]:
m = folium.Map(
    tiles="Stamen Terrain",
    location=[35.761168,-78.668271],
    zoom_start=13)
gj.Raster("elevation", opacity=0.5).add_to(m)
folium.Circle(
    radius=120,
    location=[35.769781,-78.663160],
    popup="Great Picnic Area",
    color="crimson",
    fill=False,
    tooltip=tooltip
).add_to(m)
```

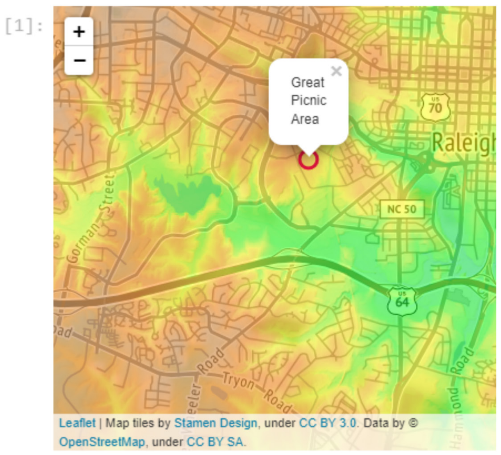

[1]:
+
−
Great Picnic Area
70
Raleigh
NC 50
64
Gorman Street
Tryon Road
Leaflet | Map tiles by Stamen Design, under CC BY 3.0. Data by © OpenStreetMap, under CC BY SA.

FIGURE 6 Demonstration of GRASS-folium integration showing the creation of a folium map with `grass.jupyter.Raster`.

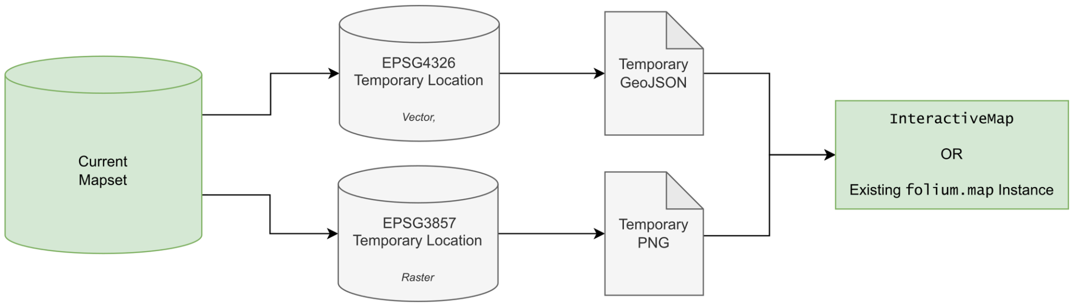


FIGURE 7 Diagram of GRASS-folium integration process.

ded in Jupyter Notebooks. In addition to zooming in and out, users can add a layer control menu to toggle between layers and custom tile base layers, selected either from a list of built-in tilesets or with a URL to a custom tileset. Unlike Map, which resamples data to fit the rendered image size, `InteractiveMap` by default renders rasters in their native resolution so that the rasters can be inspected by zooming in. For large, high-resolution rasters, the default behavior for `InteractiveMap` can be computationally-intensive and may require a problematically large amount of memory; in this scenario, the user can use the region manager to set a lower-resolution computational region or smaller extent.

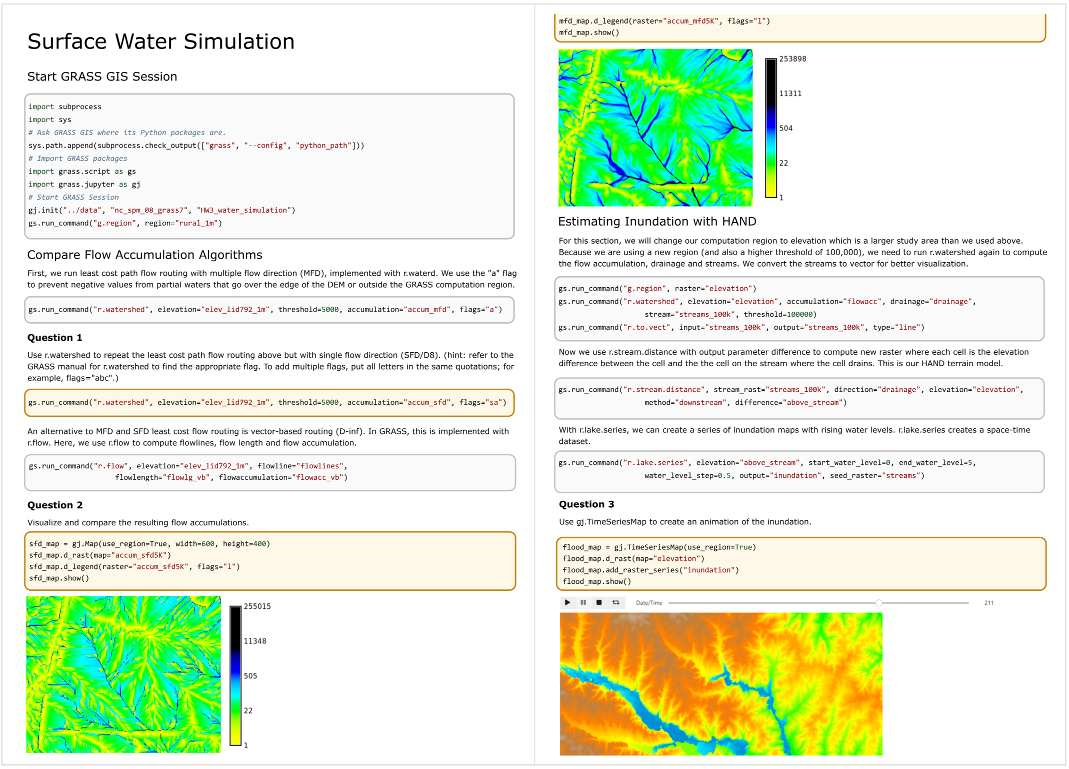
Compared to the other visualization classes which use the GRASS rendering library, GRASS-folium integration and `InteractiveMap` are more complex and computationally intensive because they require data to be reprojected. For visualization, folium and the underlying Leaflet.js package use Web Mercator projection (EPSG:3857), the de facto standard for web mapping applications. However, any coordinates, such as vector data in a GeoJSON file and bounding box of raster image overlays, need to be specified in degrees of latitude and longitude (WGS84, EPSG:4326); folium reprojects latitude and longitude to Web Mercator internally. Therefore, assuming the source and target CRS do not match, to render vector data, the `grass.jupyter` package first reprojects vector data to WGS84 and then exports it to GeoJSON. To render raster data, `grass.jupyter` reprojects a raster to Web Mercator and exports it as a PNG file. Additionally, the coordinates of the raster bounding box are reprojected to WGS84 so that folium fits the PNG image to specific bounds.

As shown in Figure 7, data are passed from GRASS GIS to folium by: (1) creating temporary locations in EPSG:3857 and EPSG:4326; (2) reprojecting data to the temporary locations, then (3) exporting the results as temporary GeoJSON or PNG files before (4) importing the data to a folium map. Raster bounds are passed as a variable. The reprojection and GeoJSON and PNG export is handled by the `ReprojectionRenderer` class, which also stores the name of the temporary files and raster boundaries. The `Raster` and `Vector` classes use `ReprojectionRenderer` to create the PNG and GeoJSON files and add them to an existing `folium.map` instance. Finally, the `InteractiveMap` class uses `ReprojectionRenderer`, `Raster` and `Vector` to create folium maps.

`InteractiveMap` wraps folium completely creating an intuitive API that is consistent with other `grass.jupyter` visualization classes. The simplicity of `InteractiveMap` combined with its consistent syntax with other `grass.jupyter` classes makes it a more accessible option, allowing users to avoid learning the folium package in addition to `grass.jupyter`. However, folium offers extensive customization options and additional functionality like heatmaps and tooltip pop-ups which cannot yet be accessed with the `InteractiveMap` class. Therefore, the `add_to` method in the `Raster` and `Vector` classes add data to existing `folium.map` instances, allowing users to take full advantage of folium's library (see Figure 6) and to allow folium users access to GRASS data.

4 | EXAMPLE APPLICATION

In this section, we show how `grass.jupyter` can be used to facilitate geospatial education and discuss an example workflow from a tutorial notebook. We employed `grass.jupyter` to teach a graduate-level course on Geospatial Computing and Simulation (GIS714) at North Carolina State University in Raleigh, North Carolina, USA. Previously,



Surface Water Simulation

Start GRASS GIS Session

```
import subprocess
import sys
# Ask GRASS GIS where its Python packages are.
sys.path.append(subprocess.check_output(["grass", "--config", "python_path"]))
# Import GRASS packages
import grass.script as gs
import grass.jupyter as gj
# Start GRASS Session
gj.init("../data", "nc_spm_08_grass7", "HW3_water_simulation")
gs.run_command("g.region", region="rural_1m")
```

Compare Flow Accumulation Algorithms

First, we run least cost path flow routing with multiple flow direction (MFD), implemented with r.waterd. We use the "a" flag to prevent negative values from partial waters that go over the edge of the DEM or outside the GRASS computation region.

```
gs.run_command("r.watershed", elevation="elev_lid792_1m", threshold=5000, accumulation="accum_mfd", flags="a")
```

Question 1

Use r.watershed to repeat the least cost path flow routing above but with single flow direction (SFD/D8). (hint: refer to the GRASS manual for r.watershed to find the appropriate flag. To add multiple flags, put all letters in the same quotations; for example, flags="abc".)

```
gs.run_command("r.watershed", elevation="elev_lid792_1m", threshold=5000, accumulation="accum_sfd", flags="sa")
```

An alternative to MFD and SFD least cost flow routing is vector-based routing (D-inf). In GRASS, this is implemented with r.flow. Here, we use r.flow to compute flowlines, flow length and flow accumulation.

```
gs.run_command("r.flow", elevation="elev_lid792_1m", flowline="flowlines",
                flowlength="flowlg_vb", flowaccumulation="flowacc_vb")
```

Question 2

Visualize and compare the resulting flow accumulations.

```
sfd_map = gj.Map(use_region=True, width=600, height=400)
sfd_map.d_rast(map="accum_sfd5K")
sfd_map.d_legend(raster="accum_sfd5K", flags="l")
sfd_map.show()
```

```
mfd_map.d_legend(raster="accum_mfd5K", flags="l")
mfd_map.show()
```

Estimating Inundation with HAND

For this section, we will change our computation region to elevation which is a larger study area than we used above. Because we are using a new region (and also a higher threshold of 100,000), we need to run r.watershed again to compute the flow accumulation, drainage and streams. We convert the streams to vector for better visualization.

```
gs.run_command("g.region", raster="elevation")
gs.run_command("r.watershed", elevation="elevation", accumulation="flowacc", drainage="drainage",
                stream="streams_100k", threshold=100000)
gs.run_command("r.to.vect", input="streams_100k", output="streams_100k", type="line")
```

Now we use r.stream.distance with output parameter difference to compute new raster where each cell is the elevation difference between the cell and the the cell on the stream where the cell drains. This is our HAND terrain model.

```
gs.run_command("r.stream.distance", stream_rast="streams_100k", direction="drainage", elevation="elevation",
                method="downstream", difference="above_stream")
```

With r.lake.series, we can create a series of inundation maps with rising water levels. r.lake.series creates a space-time dataset.

```
gs.run_command("r.lake.series", elevation="above_stream", start_water_level=0, end_water_level=5,
                water_level_step=0.5, output="inundation", seed_raster="streams")
```

Question 3

Use gj.TimeSeriesMap to create an animation of the inundation.

```
flood_map = gj.TimeSeriesMap(use_region=True)
flood_map.d_rast(map="elevation")
flood_map.add_raster_series("inundation")
flood_map.show()
```

FIGURE 8 Illustration of an excerpt from the Surface Water Simulation notebook. Tutorial cells are in gray and answers to questions are in orange.

the course was taught with static, HTML-based tutorials described in Petras et al. (2015). With the completion of `grass.jupyter`, we were able to translate these tutorials to interactive Jupyter Notebooks that mix tutorial material and code with assignment questions. The questions require students to write their own code and add text aimed to demonstrate comprehension of the material and the ability to implement their own computations and simulations. In addition to managing the GRASS session, we used `grass.jupyter` to visualize the input data and computed results.

The notebooks are hosted on GitHub in the public course repository (GIS714-Assignments, 2022) and students can choose to either run the notebooks remotely using Binder (Jupyter et al., 2018) or install GRASS GIS and Jupyter to run the notebooks locally. The notebooks cover material from 5 different topics in Geospatial Computing and Simulation: Introduction to GRASS GIS, Temporal Analyses, Data Simulation, Surface Water Simulations, and Spread Models.

In the Surface Water Processes notebook (Figure 8), students learn several different methods in GRASS GIS for computing flow accumulation: routing with multiple flow direction, single flow direction and vector-based routing. Then, they implement the HAND (Height Above Nearest Drainage) method (Nobre et al., 2011) to model flooding along a stream. The notebook begins by launching GRASS GIS: after importing the `grass.jupyter` package as `gj`, we initiate the GRASS session with `gj.init`. On the Windows operating system, this process contains extra steps; we provided the students with more detailed instructions for working with GRASS GIS and Jupyter Notebook in the Windows operating system (Haedrich et al., 2022). We use the GRASS North Carolina example dataset (GRASS GIS, 2022a) with a small computational region (700 × 750 m) and a resolution of 1 m.

In the first half of the notebook, we compute and compare several flow accumulation algorithms. The first two are least cost path algorithms: single and multiple flow direction, both implemented in `r.watershed`. The notebook gives a demonstration of the single flow direction computation and then asks students to read the GRASS manual and change parameters to use the multiple flow direction algorithm instead. The third algorithm is a vector-based method implemented in `r.flow`. After computing flow accumulation with each of the three methods, students are asked to compare the results by visualizing the outputs with `gj.Map` and writing a brief description.

In the second half of the notebook, we model a flood event where the water level rises to 5 m above the channel banks. We use the HAND method along a small network of streams. We also use a larger computational region for this section and therefore, a higher threshold when computing drainage basins with `r.watershed`. After computing the streams, drainage basins, and flow accumulation rasters with `r.watershed`, we compute the HAND terrain raster with `r.stream.distance`. Finally, we create a time series of the water rising from its banks to 5 m above at 0.5-m intervals with `r.lake.series`. The last question of the notebook asks students to visualize the resulting space–time dataset with the `TimeSeriesMap` class, creating an animation of the flood caused by the rising water levels.

The focus of our work was on the quick development and implementation of `grass.jupyter` to support notebook tutorials, so we did not examine the impact on the students directly. We were still able to make some general observations in the classroom. Several students noted they enjoyed being able to experiment with GRASS tools by varying parameters and visually observing the effect while working through the tutorial sections of the notebooks. By hosting the notebooks on GitHub and including a link to Binder, students could easily access and run the notebooks without installing and configuring software on their personal computers. This also gave the students exposure to working with Git and GitHub, important tools for open science and collaborative research. However, running the notebooks locally was difficult for many students. Due to the required configuration of path variables on the Windows operating system and installation of required packages, many students switched to Binder right away. For students that did get the notebooks running locally, we often encountered issues that were specific to their version of GRASS GIS or operating system.

5 | DISCUSSION

In this aricle, we presented a novel Python package, `grass.jupyter`, that integrates GRASS GIS with Jupyter Notebooks by providing convenient visualization and session management tools. The package, a product of community collaboration and discussion, extends the existing GRASS Python API so that GRASS GIS is more accessible to Jupyter Notebook users and provides a user-friendly way to share GRASS workflows. In particular, the package removes the need for users to modify environmental variables when launching GRASS GIS or creating renderings. It also provides interactive visualization tools through integration with folium and ipywidgets, allowing users to explore data by zooming in and out and creating animations of temporal data. All of the visualizations can be easily saved outside the notebook, giving users another method for sharing their work. We also describe our main application where `grass.jupyter` was used to facilitate teaching a graduate-level course on advanced geospatial modeling and discuss a specific example notebook on surface water processes.

By streamlining the usage of GRASS GIS in Jupyter Notebooks, the package allows Jupyter users easier access to GRASS's broad variety of tools, from basic GIS processing tools to remote sensing and hydrology algorithms. Although these workflows were possible without `grass.jupyter`, they either required using and integrating many different Python libraries with extensive custom code or overcoming the barriers with the existing GRASS Python API. Additionally, using `grass.jupyter` can significantly shorten the number of code lines required. For example, a recent Jupyter Notebook tutorial on urban development modeling was recently re-written using `grass.jupyter` and the code supporting the notebooks was shortened by 61% (Petrasova & Petras, 2022).

In addition to session management, `grass.jupyter` enables easy visualization of data in GRASS GIS. There are several packages that are particularly popular for data visualization in Jupyter Notebooks, such as Matplotlib (Hunter, 2007), seaborn (Waskom, 2021), imageio (Klein et al., 2022) and geospatial-specific libraries like folium (Filipe et al., 2022) and GeoPandas (Jordahl et al., 2020). Visualization workflows using these libraries often require users to manipulate and manage data closely: re-projecting data, explicitly handling extent plotted or using separate libraries for raster and vector data. With the `grass.jupyter` package and GRASS approach to data management, users do not need to clip data to matching extents, handle projection mismatches or re-sample data to matching resolutions to visualize it. The package also provides a unified interface for visualizing raster and vector data.

While open-source GIS projects such as Whitebox and QGIS use these popular visualization packages for displaying data, they additionally have specialized extensions for Jupyter Notebooks to provide more interactivity or simplify setup. Whitebox has a third-party package for Jupyter Notebooks that creates an interactive, inline GUI for users to browse and execute Whitebox tools (giswqs/whiteboxgui, 2022). QGIS can also be used from within a notebook but, similarly to GRASS GIS, it requires some environment configuration to access QGIS tools (van der Kwast, 2022b). The `qgis-nbextension` extension helps shorten this environmental setup for QGIS (2022). GRASS' Jupyter-specific addition, the `grass.jupyter` package, currently focuses on management of environment configuration and visualization but could be extended to take advantage of other Jupyter Notebooks' interactivity features in the future.

The `grass.jupyter` package opens new opportunities in developing open educational resources for geospatial analytics, from short tutorials to advanced semester-long courses, allowing Jupyter-based courses to expand and include workflows that use GRASS's extensive tool library. Already, the package has been used in three conference workshops for teaching GRASS GIS in Python and for remote sensing (Andreo et al., 2022; GRASS GIS, 2022b; Neteler, 2022). One of the workshops was online, and two were in-person. Use of notebooks and the package together with Binder (Jupyter et al., 2018) facilitated easier interaction with the users and workshop leaders by reducing the amount of time used for software installation and environment setup, thereby allowing participants to focus on the main concepts of the workshop. In line with the work of Reades (2020) and Arribas-Bel (2019), we presented open education notebooks used to teach a graduate-level course in geospatial sciences. Reades (2020) and Arribas-Bel (2019) present materials on point pattern analysis, clustering, classification and data visualization using common geospatial libraries like GeoPandas (Jordahl et al., 2020), Shapely (Gillies et al., 2022), GDAL (Rouault et al., 2022), PySAL (Rey & Anselin, 2007), and folium (Filipe et al., 2022). The notebooks presented here leverage GRASS's extensive toolbox for several modeling workflows, ranging from hydrological modeling to urban development modeling.

With feedback from the graduate course students and the GRASS user community, we have accumulated a list of features that would improve the integration of GRASS GIS and Jupyter Notebooks. In particular, we see several opportunities for future work on improving the handling of large datasets and offering increased interactivity with `InteractiveMap`. As mentioned in Section 3.2.5, rendering large rasters at their native resolution in `InteractiveMap` can be computationally intensive and may require a problematically large amount of memory; offering a tiling method using GDAL tools could be a solution. Similarly, for large time series, parallelization of `TimeSeriesMap` could further reduce the rendering time. For increasing the interactivity with `InteractiveMap`, we hope to add vector and raster pixel querying. Raster pixel querying would likely require integration with ipyleaflet, another Python library for Leaflet.js that allows for front-end to back-end communication, thereby capturing user input.

In the future, we also hope to expand the curriculum of GIS714 at North Carolina State University to include additional geocomputation domains such as machine-learning algorithms (MLAs), clustering and classification and the processing and analysis of remote sensing data. For example, the `r.learn` and `v.class` tool families both include several machine learning classification algorithms and they have been applied in several studies (Flasse et al., 2021; Koreň et al., 2021; Silver et al., 2019). GRASS GIS can also be integrated with third-party MLAs like MaxEnt (Andreo et al., 2021) which could be demonstrated using a notebook tutorial. For classification and clustering in remote sensing, there are already several tutorials demonstrating typical workflows using `grass.jupyter` (Andreo et al., 2022; Neteler, 2022).

The package presented here and the accompanying notebooks are one piece in a mosaic of efforts that support and advance open science. By integrating GRASS GIS and Jupyter Notebooks, `grass.jupyter` supports the creation of more open education resources for learning geospatial programming, helping to meet the rising demand for programming knowledge in the geospatial sciences. Since Jupyter Notebooks are readily shareable and operate in web browsers, the `grass.jupyter` also facilitates remote, hybrid and asynchronous instruction. The authors of this article hope this work is a catalyst towards expanding the capabilities of `grass.jupyter` and increasing the quantity and quality of open education resources for advanced geospatial modeling.

ACKNOWLEDGMENTS

Funding for this work was provided by Google Summer of Code 2021, a GRASS GIS Mini Project Grant on `grass.jupyter`, and the Center for Geospatial Analytics at North Carolina State University. The authors would also like to acknowledge the support, advice and guidance provided by the GRASS GIS development community, in particular Veronica Andreo, and the graduate students in the NC State University GIS714 course offered in spring 2022.

CONFLICT OF INTEREST STATEMENT

The authors have no conflict of interest to declare.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in GRASS GIS at https://doi.org/10.5281/zenodo.6612307.

ORCID

Caitlin Haedrich https://orcid.org/0000-0003-4373-5691
Vaclav Petras https://orcid.org/0000-0001-5566-9236
Anna Petrasova https://orcid.org/0000-0002-5120-5538
Stefan Blumentrath https://orcid.org/0000-0001-6675-1331
Helena Mitasova https://orcid.org/0000-0002-6906-3398

REFERENCES

Anbaroğlu, B. (2021). A collaborative GIS programming course using GitHub classroom. *Transactions in GIS*, *25*(6), 3132–3158. https://doi.org/10.1111/tgis.12810

Andreo, V., Cuervo, P. F., Porcasi, X., Lopez, L., Guzman, C., & Scavuzzo, C. M. (2021). Towards a workflow for operational mapping of *Aedes aegypti* at urban scale based on remote sensing. *Remote Sensing Applications: Society and Environment*, *23*, 100554. https://doi.org/10.1016/j.rsase.2021.100554

Andreo, V., Neteler, M., & Nartiss, M. (2022). *Notebook for the GRASS GIS for remote sensing data processing and analysis workshop at FOSS4G 2022 in Florence*. https://github.com/veroandreo/foss4g2022_grass4rs

Arribas-Bel, D. (2019). A course on geographic data science. *Journal of Open Source Education*, *2*(16), 42. https://doi.org/10.21105/jose.00042

Belgiu, M., Strobl, J., & Wallentin, G. (2015). Open geospatial education. *ISPRS International Journal of Geo-Information*, *4*(2), 697–710. https://doi.org/10.3390/ijgi4020697

Bornaetxea, T., & Marchesini, I. (2022). r.survey: A tool for calculating visibility of variable-size objects based on orientation. *International Journal of Geographical Information Science*, *36*(3), 429–452. https://doi.org/10.1080/13658816.2021.1942476

Bowlick, F. J., Goldberg, D. W., & Bednarz, S. W. (2017). Computer science and programming courses in geography departments in the United States. *The Professional Geographer*, *69*(1), 138–150. https://doi.org/10.1080/00330124.2016.1184984

Brunsdon, C., & Comber, A. (2021). Opening practice: Supporting reproducibility and critical spatial data science. *Journal of Geographical Systems*, *23*(4), 477–496. https://doi.org/10.1007/s10109-020-00334-2

Cimburova, Z., & Blumentrath, S. (2022). Viewshed-based modelling of visual exposure to urban greenery—An efficient GIS tool for practical planning applications. *Landscape and Urban Planning*, *222*, 104395. https://doi.org/10.1016/j.landurbplan.2022.104395

Ciolli, M., Federici, B., Ferrando, I., Marzocchi, R., Sguerso, D., Tattoni, C., Vitti, A., & Zatelli, P. (2017). FOSS tools and applications for education in geospatial sciences. *ISPRS International Journal of Geo-Information*, *6*(7), 225. https://doi.org/10.3390/ijgi6070225

CoCalc – Run Python Online. (2022). https://cocalc.com/features/python

Code Ocean. (2022). https://codeocean.com/

Committee on Earth Observation Satellites. (2021). *Jupyter Notebooks for capacity development webinar | CEOS*. https://ceos.org/meetings/jupyter-notebooks-for-capacity-development-webinar/

Colorado University Boulder Earth Lab. (2022). *Earth lab: Free, online courses, tutorials and tools*. https://www.earthdatascience.org/

Cronin, C. (2017). Openness and praxis: Exploring the use of open educational practices in higher education. *International Review of Research in Open and Distributed Learning*, *18*(5), 15–34. https://doi.org/10.19173/irrodl.v18i5.3096, https://www.irrodl.org/index.php/irrodl/article/view/3096/4301

Engelberger, F., Galaz-Davison, P., Bravo, G., Rivera, M., & Ramírez-Sarmiento, C. A. (2021). Developing and implementing cloud-based tutorials that combine bioinformatics software, interactive coding, and visualization exercises for distance learning on structural bioinformatics. *Journal of Chemical Education*, *98*(5), 1801–1807. https://doi.org/10.1021/acs.jchemed.1c00022

Etherington, T. R. (2016). Teaching introductory GIS programming to geographers using an open source python approach. *Journal of Geography in Higher Education*, *40*(1), 117–130. https://doi.org/10.1080/03098265.2015.1086981

Filipe, F. A., Story, R., Gardiner, J., Journois, M., Rump, H., Bird, A., Lima, A., Cano, J., Oefelein, M., Leonel, J., Baker, J., Sampson, T., Reades, J., Welsh, B., Kong, Q., Komarov, O., Crosby, A., Harris, G., Dumas, R., … Duke, J. (2022). *Python-visualization/folium: v0.14.0*.

Flasse, C., Grippa, T., & Fennia, S. (2021). A tool for machine learning based dasymetric mapping approaches in GRASS GIS. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *46*, 55–62. https://doi.org/10.5194/isprs-archives-XLVI-4-W2-2021-55-2021

Gebbert, S., & Pebesma, E. (2017). The GRASS GIS temporal framework. *International Journal of Geographical Information Science*, *31*(7), 1273–1292. https://doi.org/10.1080/13658816.2017.1306862

Gillies, S., Ward, B., & Petersen, A. S. (2013). *Rasterio: Geospatial raster I/O for Python programmers*. https://github.com/rasterio/rasterio

Gillies, S., van der Wel, C., Van den Bossche, J., Taves, M. W., Arnott,J., Ward, B. C., & others. (2023). Shapely (2.0.1) [Python]. https://doi.org/10.5281/zenodo.5597138

GIS Open Courseware. (2022) GIS 714: Assignments. https://github.com/ncsu-geoforall-lab/GIS714-assignments

giswqs/whiteboxgui. (2022). *An interactive GUI for WhiteboxTools in a Jupyter-based environment*. https://github.com/giswqs/whiteboxgui

giswqs/whitebox-python. (2022). https://github.com/giswqs/whitebox-python

Google LLC. (2022). *Google colaboratory*. https://colab.research.google.com/

Graser, A. (2022). *Free and open source GIS ramblings*. https://anitagraser.com

GRASS GIS. (2022a). *Bringing advanced geospatial technologies to the world*. https://grass.osgeo.org/

GRASS GIS. (2022b). *From beginner to power user (FOSS4G 2021 workshop)*. https://github.com/ncsu-geoforall-lab/grass-gis-workshop-FOSS4G-2021

Haedrich, C., Petras, V., Petrasova, A., & Mitasova, H. (2022). *GIS714 assignments*.

Harmon, B. A., Mitasova, H., Petrasova, A., & Petras, V. (2019). r.sim.terrain 1.0: A landscape evolution model with dynamic hydrology. *Geoscientific Model Development*, *12*(7), 2837–2854. https://doi.org/10.5194/gmd-12-2837-2019

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

ipywidgets: Interactive HTML widgets. (2022). https://github.com/jupyter-widgets/ipywidgets

Jasiewicz, J., & Stepinski, T. F. (2013). Geomorphons—A pattern recognition approach to classification and mapping of landforms. *Geomorphology*, *182*, 147–156. https://doi.org/10.1016/j.geomorph.2012.11.005

Jordahl, K., Van den Bossche, J., Fleischmann, M., Wasserman, J., McBride, J., Gerard, J., Tratner, J., Perry, M., Badaracco, A. G., Farmer, C., Hjelle, G. A., Snow, A. D., Cochran, M., Gillies, S., Culbertson, L., Bartos, M., Eubank, N., Albert, M., Bilogur, A., … Leblanc, F. (2020). *Geopandas/geopandas: v0.8.1*.

Jupyter Kernels · jupyter/jupyter Wiki. (2022). https://github.com/jupyter/jupyter

Jupyter, P., Bussonnier, M., Forde, J., Freeman, J., Granger, B., Head, T., Holdgraf, C., Kelley, K., Nalvarte, G., Osheroff, A., Pacer, M., Panda, Y., Perez, F., Ragan-Kelley, B., & Willing, C. (2018). Binder 2.0—Reproducible, interactive, sharable environments for science at scale. In F. Akici, D. Lippa, D. Niederhut, & M. Pacer (Eds.), *17th Python in Science Conference* (pp. 113–120). https://doi.org/10.25080/Majora-4af1f417-011

Karlsson, K. (2022). https://www.youtube.com/c/KlasKarlsson

Klein, A., Wallkötter, S., Silvester, S., Tanbakuchi, A., Müller, P., Nunez-Iglesias, J., Harfouche, M., Dennis, A. L., McCormick, M., Rai, A., Ladegaard, A., Smith, T. D., van Kemenade, H., Vaillant, G., Nises, J., Komarčević, M., Schambach, M., Andrew, C. D., Gohlke, C., … Inggs, G. (2022). *Imageio/imageio: v2.23.0*.

Kluyver, T., Ragan-Kelley, B., Rez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., & Jupyter Development Team. (2016). Jupyter Notebooks—A publishing format for reproducible computational workflows. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, *2016*, 87–90. https://doi.org/10.3233/978-1-61499-649-1-87

Koreň, M., Jakuš, R., Zápotocký, M., Barka, I., Holuša, J., Ďuračiová, R., & Blaženec, M. (2021). Assessment of machine learning algorithms for modeling the spatial distribution of bark beetle infestation. *Forests*, *12*(4), 395. https://doi.org/10.3390/f12040395

Landa , M. , Neteler , M. , Metz , M. , Petrasova , A. , Clements , G. M. , Bowman , H. , Petras , V. , Gebbert , S. , Cho , H. , Delucchi , L. , Zambelli , P. , Barton , M. , Zigo , T. , Chemin , Y. , Nartišs , M. , Turek , Š. , Kudrnovsky , H. , Larsson , N. , Lennert , M. , … Tawalika , C. (2022). *OSGeo/grass: GRASS GIS 8.2.0* .

Lasser, J., Manik, D., Silbersdorff, A., Säfken, B., & Kneib, T. (2021). Introductory data science across disciplines, using python, case studies, and industry consulting projects. *Teaching Statistics*, *43*(Suppl. 1), S190–S200. https://doi.org/10.1111/test.12243

McKiernan, E. C., Bourne, P. E., Brown, C. T., Buck, S., Kenall, A., Lin, J., McDougall, D., Nosek, B. A., Ram, K., Soderberg, C. K., Spies, J. R., Thaney, K., Updegrove, A., Woo, K. H., & Yarkoni, T. (2016). How open science helps researchers succeed. *eLife*, *5*, e16800. https://doi.org/10.7554/eLife.16800

Metz, M., Andreo, V., & Neteler, M. (2017). A new fully gap-free time series of land surface temperature from MODIS LST data. *Remote Sensing*, *9*(12), 1333. https://doi.org/10.3390/rs9121333

Mitasova, H., Thaxton, C., Hofierka, J., McLaughlin, R., Moore, A., & Mitas, L. (2004). Path sampling method for modeling overland water flow, sediment transport, and short term terrain evolution in open source GIS. In C. T. Miller & G. F. Pinder (Eds.), *Developments in water science* (Vol. 55, pp. 1479–1490). Elsevier. https://doi.org/10.1016/S0167-5648(04)80159-X

Morin, A., Urban, J., Adams, P. D., Foster, I., Sali, A., Baker, D., & Sliz, P. (2012). Shining light into black boxes. *Science*, *336*(6078), 159–160. https://doi.org/10.1126/science.1218263

Murray, A., van Kemenade, H., Wiredfool, Clark (Alex), J. A., Karpinsky, A., Baranovič, O., Gohlke, C., Dufresne, J., DWesl, Schmidt, D., Kopachev, K., Houghton, A., Mani, S., Landey, S., Vashek, J. W., Piolie, J. D., Stanislau, T., Caro, D., Martinez, U., … Base, M. (2022). *Python-pillow/pillow: 9.3.0*.

Neteler, M. (2022). https://github.com/neteler/jupyter_sentinel2_grass_gis

Neteler, M., & Mitasova, H. (2013). *Open source GIS: A GRASS GIS approach*. Springer Science & Business Media.

Nobre, A. D., Cuartas, L. A., Hodnett, M., Rennó, C. D., Rodrigues, G., Silveira, A., Waterloo, M., & Saleska, S. (2011). Height above the nearest drainage—A hydrologically relevant new terrain model. *Journal of Hydrology*, *404*(1), 13–29. https://doi.org/10.1016/j.jhydrol.2011.03.051

Open Courseware for GIS. (2022). https://courses.gisopencourseware.org

Páez, A. (2021). Open spatial sciences: An introduction. *Journal of Geographical Systems*, *23*(4), 467–476. https://doi.org/10.1007/s10109-021-00364-4

Palomino, J., Muellerklein, O. C., & Kelly, M. (2017). A review of the emergent ecosystem of collaborative geospatial tools for addressing environmental challenges. *Computers, Environment and Urban Systems*, *65*, 79–92. https://doi.org/10.1016/j.compenvurbsys.2017.05.003

Pérez, F., & Granger, B. E. (2007). IPython: A system for interactive scientific computing. *Computing in Science and Engineering*, *9*(3), 21–29. https://doi.org/10.1109/MCSE.2007.53, https://ipython.org

Perkel, J. M. (2018). Why Jupyter is data scientists' computational notebook of choice. *Nature*, *563*(7729), 145–146. https://doi.org/10.1038/d41586-018-07196-1

Petras, V., Newcomb, D. J., & Mitasova, H. (2017). Generalized 3D fragmentation index derived from lidar point clouds. *Open Geospatial Data, Software and Standards*, *2*(1), 9. https://doi.org/10.1186/s40965-017-0021-8

Petras, V., Petrasova, A., Harmon, B., Meentemeyer, R., & Mitasova, H. (2015). Integrating free and open source solutions into geospatial science education. *ISPRS International Journal of Geo-Information*, *4*(2), 942–956. https://doi.org/10.3390/ijgi4020942, http://www.mdpi.com/2220-9964/4/2/942

Petrasova, A., & Petras, V. (2022). *Futures model introduction using Jupyter Notebook*. https://github.com/ncsu-landscape-dynamics/futures-model-intro-notebook

Pimentel, J. F., Murta, L., Braganholo, V., & Freire, J. (2019). A large-scale study about quality and reproducibility of Jupyter Notebooks. *16th IEEE/ACM International Conference on Mining Software Repositories, Montreal, QC, Canada* (pp. 507–517). IEEE. https://doi.org/10.1109/MSR.2019.00077

Project Jupyter. (2022). https://jupyter.org

QGIS. (2022). Jupyter Notebook extension: Use qgis in jupyter notebook. https://github.com/3liz/qgis-nbextension

QGIS Development Team. (2022). *QGIS geographic information system*. QGIS Association. https://www.qgis.org

Reades, J. (2020). Teaching on Jupyter: Using notebooks to accelerate learning and curriculum development. *Region*, *7*(3), 21–34. https://doi.org/10.18335/region.v7i1.282

Reades, J., & Rey, S. J. (2021). Geographical python teaching resources: Geopyter. *Journal of Geographical Systems*, *23*(4), 579–597. https://doi.org/10.1007/s10109-021-00346-6

Rey, S. J. (2018). Code as text: Open source lessons for geospatial research and education. In J.-C. Thill & S. Dragicevic (Eds.), *GeoComputational analysis and modeling of regional systems, advances in geographic information science* (pp. 7–21). Springer. https://doi.org/10.1007/978-3-319-59511-5_2

Rey, S. J., & Anselin, L. (2007). PySAL: A python library of spatial analytical methods. *The Review of Regional Studies*, *37*(1), 5–27. https://doi.org/10.52324/001c.8285

Rocchini, D., & Neteler, M. (2012). Let the four freedoms paradigm apply to ecology. *Trends in Ecology & Evolution*, *27*(6), 310–311. https://doi.org/10.1016/j.tree.2012.03.009

Rocchini, D., Petras, V., Petrasova, A., Chemin, Y., Ricotta, C., Frigeri, A., Landa, M., Marcantonio, M., Bastin, L., Metz, M., Delucchi, L., & Neteler, M. (2017). Spatio-ecological complexity measures in GRASS GIS. *Computers & Geosciences*, *104*, 166–176. https://doi.org/10.1016/j.cageo.2016.05.006

Rocchini, D., Petras, V., Petrasova, A., Horning, N., Furtkevicova, L., Neteler, M., Leutner, B., & Wegmann, M. (2017). Open data and open source for remote sensing training in ecology. *Ecological Informatics*, *40*, 57–61. https://doi.org/10.1016/j.ecoinf.2017.05.004

Rouault, E., Warmerdam, F., Schwehr, K., Kiselev, A., Butler, H., Łoskot, M., Szekeres, T., Tourigny, E., Landa, M., Miara, I., Elliston, B., Kumar, C., Plesea, L., Morissette, D., Jolma, A., & Dawson, N. (2022). *Gdal*.

Rowe, F., Maier, G., Arribas-Bel, D., & Rey, S. (2020). The potential of notebooks for scientific publication, reproducibility and dissemination. *Region*, *7*(3), E1–E5. https://doi.org/10.18335/region.v7i3.357

Silver, M., Tiwari, A., & Karnieli, A. (2019). Identifying vegetation in arid regions using object-based image analysis with RGB-only aerial imagery. *Remote Sensing*, *11*(19), 2308. https://doi.org/10.3390/rs11192308

Stodden, V., Guo, P., & Ma, Z. (2013). Toward reproducible computational research: An empirical analysis of data and code policy adoption by journals. *PLoS ONE*, *8*(6), e67111. https://doi.org/10.1371/journal.pone.0067111

Stodden, V., Seiler, J., & Ma, Z. (2018). An empirical analysis of journal policy effectiveness for computational reproducibility. *Proceedings of the National Academy of Sciences of the United States of America*, *115*(11), 2584–2589. https://doi.org/10.1073/pnas.1708290115

van der Kwast, H. (2022a). https://www.youtube.com/c/HansvanderKwast

van der Kwast, H. (2022b). *Jvdkwast/PyQGISTutorials: Jupyter Notebooks with PyQGIS tutorials*. https://github.com/jvdkwast/PyQGISTutorials

Wagemann, J., Fierli, F., Mantovani, S., Siemen, S., Seeger, B., & Bendix, J. (2022). Five guiding principles to make jupyter notebooks fit for earth observation data education. *Remote Sensing*, *14*(14), 3359. https://doi.org/10.3390/rs14143359

Waskom, M. L. (2021). Seaborn: Statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021

Watson, M. (2015). When will 'open science' become simply 'science'? *Genome Biology*, *16*(1), 101. https://doi.org/10.1186/s13059-015-0669-2

Weller, M., Jordan, K., DeVries, I., & Rolfe, V. (2018). Mapping the open education landscape: Citation network analysis of historical open and distance education research. *Open Praxis*, *10*(2), 109–126. https://doi.org/10.5944/openpraxis.10.2.822

How to cite this article: Haedrich, C., Petras, V., Petrasova, A., Blumentrath, S., & Mitasova, H. (2023). Integrating GRASS GIS and Jupyter Notebooks to facilitate advanced geospatial modeling education. *Transactions in GIS*, *00*, 1–17. https://doi.org/10.1111/tgis.13031