simVizMap: spatial visualization of simulation results

This mini tutorial shows how to install and use the package simVizMap to obtain graphics that spatially visualize results from simulations. For more information on the methodology, please refer to Arribas-Bel et al. (2011)[1].

Dependencies

Before starting to use simVizMap, you will need to have Python installed on your machine, as well as the following libraries:

- Numpy
- Scipy
- Matplotlib

If you are not familiar with the Python language, we recommend you install the Enthought Python Distribution (EPD), which is an easy-to-install bundle of several Python libraries for scientific computing. All the required dependencies, as well as many others, are included in the EPD.

Installation

First of all, go to the package website and download a copy of it. Because simVizMap is a very simple and light package contained in one simple file (simVizMap.py), the easiest way to run it is to place it in the same folder as the script that will call it or, in the case of an interactive session, in the working directory. To bring it into one of your scripts or interactive sessions, simply include the following line:

```
from simVizMap import SimVizMap
```

Input files

The main objective of simVizMap is to make it easy for non-programmers to create maps from simulation results. For that reason, we designed a universal structure of the files that the package will read to generate the graphics. The file structure should meet the following criteria:

- It has to be a plain text comma separated values (csv) file.
- The file should not have a header.

[1] A working paper version is available as a free download at http://geodacenter.asu.edu/category/public/improving-multi

- The values you wish to represent as colors need to be placed in the same location they will be in the map you want to create. Basically, this means you have to create the map yourself *in numbers* and simVizMap will convert it to colors.

- If you want to divide the map into separate blocks or regions with different lines, you can specify where the lines should go by filling a whole row or column of the line with either the character `'b'` (for black line) or `'w'` (for white line).

  IMPORTANT: neither the first nor the last row or column of the file should be a line. They should only contain numbers.

As a simple example, if we want to create a map to display four values in a square with no lines, we would have to create a csv file containing the following scheme:

```
2,3
7,1
```

If we then want to add a vertical white line and a horizontal black one, the file would look like this:

```
2,w,3
b,b,b
7,w,1
```

It does not matter which letter is set to the crossing, the result will be equivalent. Also, note that since the csv is a common format, most spreadsheet programs (like OpenOffice Calc, MS Excel or Apple Numbers) allow you to export a sheet as a csv file, which means you can prepare it in your program of choice and save it as a csv for `simVizMap`.

Usage

The usage is very similar to any traditional command-line tool, so if you are familiar with *R*, Matlab or STATA, it should be easy to use. Basically, after importing the package, you have to call the the main function and pass the appropriate arguments.

If you start an interactive Python session, create the basic map:

```
from simVizMap import SimVizMap
graphical_object = SimVizMap('path/to/file.csv')
```

You can tweak the output by passing the following commands:

- `cmap`: color scheme to be used (default='Blues').

- `cb_orientation`: orientation of the color bar, which can take the values of 'horizontal' (default), 'vertical' or `None`. If None, no color bar is added.

The object is a standard Matplotlib pcolor, so if you are familiar with it, you can customize it. The following Section exemplifies how to add labels by using two functions included in SimVizMap. Once you have the object in the final format and do not want to add any extra features (like labels or titles), you can display it:

```
graphical_object.show()
```

or save if to a png file:

```
graphical_object.save('path/to/output_file.png')
```

Finally, if you type help(SimVizMap), the interpreter will show the detailed list of arguments and attributes.

Example

In this section we present the example provided with the package that allows you to create a more elaborate figure. The original input file (simVizMap_example.csv) with the values as recquired is also provided with the package. All the code in this section is also provided in simVizMap_example.py.

As mentioned before, the first step is to import the class and the functions from the package:

```
from simVizMap import SimVizMap, set_h_tags, set_v_tags
```

Next, we create the object:

```
plot_object = SimVizMap(csv_link, cb_orientation='vertical')
```

At this point, if all we wanted was to generate the basic map, we could already save it to a file. But we will use two functions provided with the package (set_v_tags and set_h_tags) to add labels for the dimensions displayed. This will create Fig. 2 in Arribas-Bel et al. (2011).

We have to specify the labels on each dimension as a separate list:

```
sizes = ['225', ' 169', '121', '81', '49', '25']
models = ['LM', 'Moran'] * 6
sp = ['-.8', '-.4', '0', '.4', '8'] * 2
```

Note, that since the elements in models have to be repeated six times, we multiply the list; we proceed similarly with sp because it has to be repeated twice. Finally, we use those lists to pass them as arguments to the functions that will add them to the plot:

```
set_v_tags(-0.15, sizes, plot_object.p, \
        weight='bold', rotation=90, fontsize=15)
set_v_tags(-0.05, models, plot_object.p, fontsize=10)
set_h_tags(-0.075, ['Rho', 'Lambda'], plot_object.p, \
        weight='bold', fontsize=15)
set_h_tags(-0.025, sp, plot_object.p, fontsize=10)
set_h_tags(1.05, ['Sp. Lag', 'Sp. Error'], plot_object.p, \
        weight='bold', fontsize=15)
```

In first place, we pass a parameter that determines the location along the vertical (`set_v_tags`) or horizontal axis (`set_h_tags`) of the set of labels. The second argument is the list of labels we created before. The rest of the arguments are options to customize the results. If you type `help(set_v_tags)` or `help(set_h_tags)`, you will get a full list and description.

Now we are ready to either visualize or save the output. Respectively, we would then type:

```
plot_object.show()
```

to display it or:

```
plot_object.save('simVizMap_example.png')
```

to save it to a local file named `simVizMap_example.png`.

References

Arribas-Bel, D., Koschinsky, J., and Amaral, P. V. (2011). Improving the multi-dimensional comparison of simulation results: A spatial visualization approach. *Letters in Spatial and Resource Sciences*.