Extracting more data: Instructions

Team Jupyter

This document describes the necessary steps to expand our code in the provided jupyter notebook in order to extract more fields from Stellarium. It will include an example for a specific additional field, but the process can be generalized for any other field. Also, the procedure of adding more celestial bodies of interest will be provided.

Determine an additional field of interest

Suppose we are interested in adding the rise and set times to our data retrieval process. Opening Stellarium and clicking on a celestial body, reveals the format of those measures, which is as follows:

```
Rise: 22h32m
...
Set: 6h58m
```

Make necessary changes

In Stellarium_API.ipynb, we will have to make the following changes in order to extract those fields as well.

Changes in the get_location function

Some preliminary information

The function returns a python list which is then stored as a numpy array. Since numpy arrays can only have a particular data type, the output of get_location is split in two such arrays, one containing floats and the other containing strings. In our particular use case, the one containing strings only stored constellation information, thus it was named "constellations", but in case the additional data you would like to extract are textual, you have the option to rename it and increase its size or use additional numpy arrays. There is no need to know how to manage html requests, or how to use the bs4 package to parse html! This part is already done and you can just use the locate_line function, and perhaps do some basic string manipulation and data conversion (similar to what is done in already defined functions) to add more fields.

Necessary changes

Back to the get_location function, as done for all the other fields, we would have to use the locate_line function, passing 'Rise:' and 'Set:' as parameters. The function would then return the strings '22h32m' and '6h58m'. We could then decide whether we want to store that as text or further manipulate it (e.g. converting it to a float corresponding to hours, similarly to what is done in the hrmnsec_to_hr function). We would then have to pass the value inside the list after the return call. Considering the information provided in the previous section, it would be good practice to have all numeric data first and textual data second in the list that is being returned.

The following code blocks show those changes, where some previous / next lines have been retained from the original function.

```
...
const = locate_line(soup,'IAU Constellation:')
rise = locate_line(soup,'Rise:')
set = locate_line(soup,'Set:')
# Parse times, angles and convert to float
...
```

And also our return call should now be:

```
return [ra,dec,az,alt,distEarth,distSun,rise,set,const]
```

Changes in the main script

The main script written for our particular use case consists of a loop that iterates over Julian date values and extracts the parameters for a list of celestial bodies. Assuming this process remains unchanged, we will have to make some alterations to manage to store the two new fields that we added.

First we will have to change the size of the numpy arrays that are pre-allocated. In this example we assume that we added two textual fields. We will rename the constellations array to strings and update its size.

```
floats = np.full((n_rows,6),0.00)
# constelations = np.full((n_rows),'',dtype=object) <- Removed
strings = np.full((n_rows,3),'',dtype=object)
```

Then, we will update the loop part.

```
for JD in tqdm(JD_array):
    update_time(JD)
    for body in bodies:
        tmp = get_location(body)
        # floats[i,:] = tmp[0:-1] <- update to new size
        floats[i,:] = tmp[0:-3]
        # constelations[i] = tmp[-1] <- rename, update size
        strings[i,:] = tmp[-3]
        i += 1
```

Finally, when all data are gathered in those arrays, the dataframe creation process will have to be the following:

```
df = pd.DataFrame(data = {'JD' : JD_array.repeat(len(bodies)),
                          'Body'     : body_name_array,
                          'RA'       : floats[:,0],
                          'Dec'      : floats[:,1],
                          'Az'       : floats[:,2],
                          'Alt'      : floats[:,3],
                          'Dist'     : floats[:,4],
                          'Dist_Sun' : floats[:,5],
                          'Rise'     : strings[:,0],
                          'Set'      : strings[:,1],
                          'Const'    : strings[:,2]
})
```

where we have removed the line

```
'Const'    : constelations
```

Running the script after making those changes will result in an expanded dataset with the new fiels. Hooray!

Adding more celestial bodies

Adding more celestial bodies is much easier to do, because of the long-format of the dataset (i.e. having a column that includes the name of the celestial body and repeating Julian date values contrary to having columns for all attributes of all bodies and no repetitions in the Julian date column). We just need to include the new celestial body of interest in the corresponding python list, like so:

```
bodies = ['Mercury','Venus','Mars',
'Jupiter','Saturn','Uranus',
'Neptune','Moon','Sun','Pluto']
```

Possible issues

We might encounter an issue if we are interested in the distance from earth and the sun of objects like 'Sirius', that do not belong in our solar system (which would be somewhat silly to try to track over time in our opinion), since in this case it is reported in light years. We would then have to accommodate for that in get_location with an if statement and provide a different string manipulation sequence, or avoid gathering that info based on the object name.

Final remarks

This document is provided in response to feedback received during the peer reviewing session, and will be added to our GitHub repository. For any comments, concerns or clarifications, feel free to email us.