GSoC 2019 Proposal

Starkit: Auto-generate Filter Curves & Photometry

Proposed by: Jaladh Singhal (jaladh-singhal)

1. Introduction

For imaging of stars using various filters, the wsynphot package of starkit provides set of filters and related functions like plot() to visualize a filter curve which shows the transmission of light for each wavelength value. Currently, to plot a filter curve, user has to pass its filter ID to the plot() function which makes it mandatory that fiter ID is known to the user. So first, the user has to find the filters of the criteria they want (like observation facility, instrument, wavelength range, etc.) from the dataset, and then they have to pass their filter IDs (which they might need to copy or remember) to the function, before finally getting the filter curves.

To simplify this cumbersome way of accessing the filters, I propose to develop several interfaces which will ease the navigation and selection of filters by providing suitable visualizations of the filters dataset and will auto-generate the filter curves for the selected filters. Most importantly, these interfaces will be completely interactive which will capture the user responses by means of UI elements (like dropdowns, sliders, buttons, etc.) and will respond to the same by providing them the information & filter curves, they want.

For calculating photometry of a combination of filter set and spectrum, the code used by the user requires them to specify lots of values including set of filter IDs, path of spectral grid and associated characteristics (teff, logg, mh, distance, etc.) of the spectrum. Currently, when they need to calculate the photometry for an arbitrary combination, they have to manually change all of these values in the code and have to re-run it.

To save the user from this hassle, I will develop an interface to automate this calculation of photometry from any arbitrary combination of filter curves and spectrum. This will be achieved through UI controls which will allow user to make selection of any arbitrary values of spectrum characteristics and filter set within moments. After which the whole code will run at the back end to provide photometry on the screen.

The central idea of this project is to leverage the adequate visualizations by enabling interactivity and UI controls, such that they serve as analytical web interfaces through which user can access what they need in a couple of clicks! These interfaces will be made accessible to the user by integrating them within living Sphinx docs, such that they will be auto generated when building the docs.

I also aim to well document the wsynphot package after integrating the developed interfaces in it, ultimately reshaping the entire package.

2. Project Goals

Following is the brief list of **deliverables** which I aim to achieve within the three-month period of Summer:

2.1. Primary Goals

2.1.1. Develop following interfaces to access Filter Curves:

- By selecting arbitrary combination of Observation Facility and Instrument.
- By specifying the wavelength range of filters required.
- By comparing no. of filters distributed among Observation Facilities and Instruments.
- By comparing wavelength range of the filters of a specified Observation Facility.

2.1.2. Develop interface to calculate Photometry

The interface that will calculate the photometry by allowing user to select arbitrary combination of spectrum and filter set, to visualize curves of both, and also to carry out filter transformations.

2.1.3. Deploy the above interfaces

All developed interfaces will be integrated within online docs (webpages) of wsynphot & starkit to make them available for user, and a brief demo for using each interface will also be documented.

2.2. Secondary Goals

2.2.1. Development of additional interfaces which:

- Shows distribution of filters across the wavelengths in EM spectrum.
- Fulfills any identified requirement as suggested by mentors.

2.2.2. Linking of photometry interface with the interfaces to access filter curves

2.2.3. Refinement of the developed interfaces

2.2.4. Testing of all the interfaces and fixing errors (if any)

2.2.5. Building auto-generating Sphinx documentation of the modules & functions of wsynphot

2.2.6. Updating documentation & wrap up work

2.3. Future Developments

If above goals are completed before time, I will start working on the following tasks, else I will do them post GSoC:

- Including animations in above interfaces like creating buttons to auto move sliders, etc.
- Creating function(s) in the wsynphot or starkit package, for any of the functionalities implemented by interfaces.
- Enhancement of the look of docs using custom CSS/HTML like building a custom introductory page for wsynphot or starkit docs, repositioning elements in docs, etc.

3. Implementation Details

First and foremost, what do I mean by an **interface**?
Interface is actually a combination of interactive visualizations and UI controls at the front end, that is tied with analytical python code using functions and dataset of the wsynphot or starkit package to serve as the back end. Also the code to develop front end will be written completely in Python by using:

- Plotly.py for coding customizable interactive visualizations.
- Dash or Jupyter Notebooks for coding the UI controls and logics of interactivity.

Plotly.py is open source Python graphing library that makes beautiful interactive graphs, embeddable on web. The best feature of this library is that the data labels are automatically shown on hovering over graph components (like bars, lines, etc.) which makes it very easy to explore data at any point in the graph. Besides, Plotly provides a wide variety of customizable features which makes it possible to leverage our visualizations to do anything we need.

Choice of development tool:

Plotly can make interactive visualizations but to make those visualizations interact with each other & with the UI controls and to serve together as an app, we need to use any one of the following:

1. **Dash** which is a Python framework (developed by the Plotly organisation) for building analytical web applications. To enable interactivity, it uses callbacks that trigger updates in other components of app when one component is changed.
2. **Jupyter Notebook** can also be used to serve as an analytical app by using ipywidgets of notebook with the figurewidgets of Plotly library.

After developing the proposed interfaces by using any of these two, it is possible to auto generate these interfaces while building the Sphinx documentation, as they will be integrated within the pages of docs. But we need to make a choice that which one of the two will we use to develop interfaces?

I propose to use Dash over Jupyter Notebooks due to following reasons:

- The interface developed in Dash can be run directly within HTML page of docs whereas in Jupyter Notebook, the interface will require a Python kernel to run so it can't be run in docs but will redirect user from docs to a platform that can run it (see deployment of interfaces for details about each case).
- Unlike Dash, code will be visible to user in the Jupyter Notebook which will not have the appeal of a proper GUI.

This choice of development tool will be discussed with the mentors in Community bonding period and will be finalized before the Summer of Code begins. It does not affect the logic of interfaces' implementation (i.e. discussed in subsequent sections - 3.1 & 3.2) but will only affect the way code will be written (in the back end) and the way deployment will be carried out (in section 3.3).

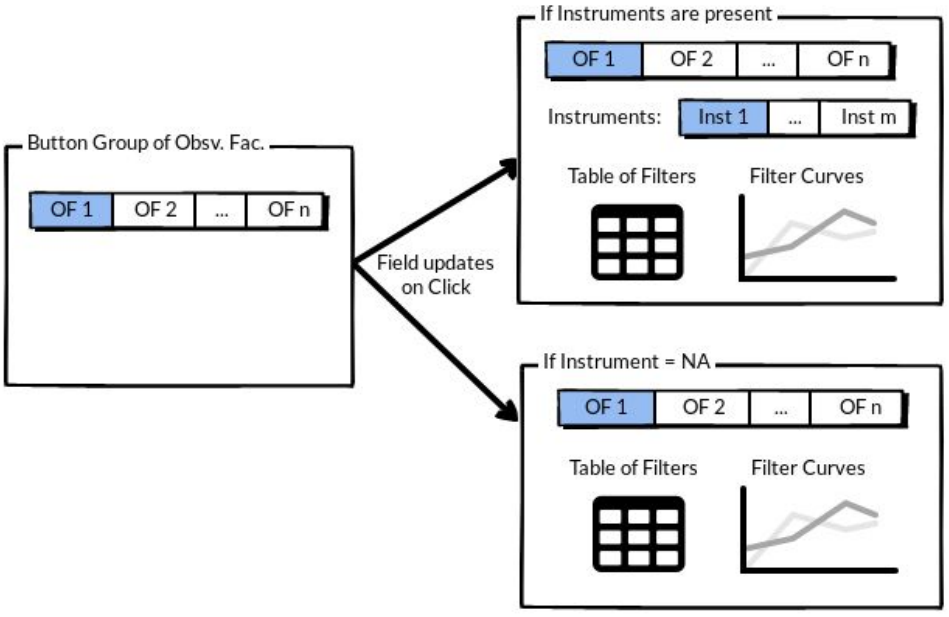
The interfaces mentioned in project goals can be developed as follows:

3.1. Interface to access the Filter Curves

The main aim of these interfaces is to provide several visualizations of the filters dataset (library of filter curves) such that user can interact with them, to easily access the filter curves and information they want. Each of these interfaces differ from each other in the way they let user to access the auto-generated filter curves.

3.1.1. By selecting arbitrary combination of Observation Facility and Instrument

It allows user to access the filters on the basis of Observation Facility and Instrument they belong to. The aim is to develop an interface similar to this as suggested by mentors.

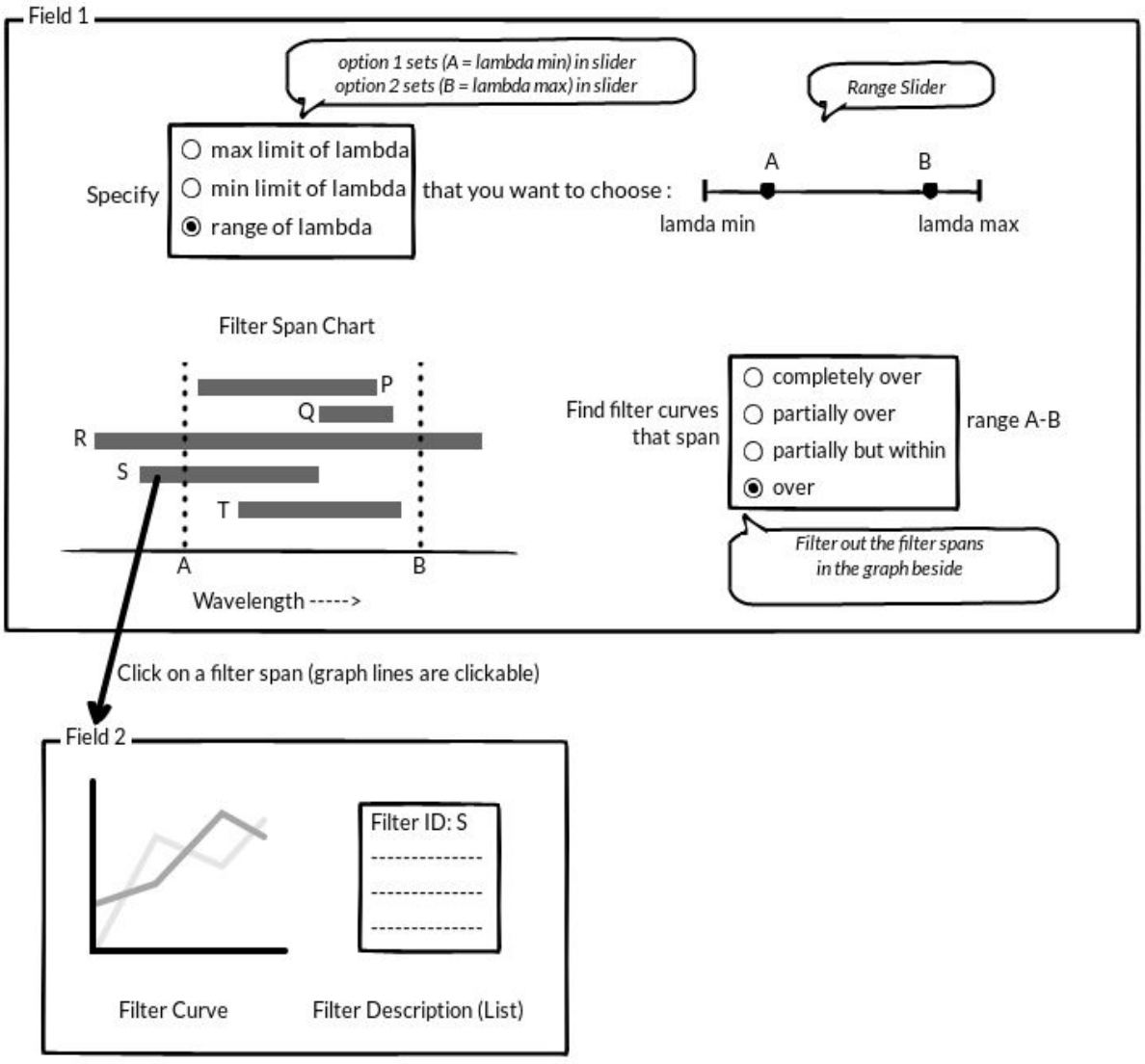


Wireframe 3.1.1

By clicking on an Observation facility, the Instruments of that facility opens (if present) and then on selecting one of the Instrument, the Filter curves & table appear.

3.1.2. By specifying the wavelength range of filter required

It allows user to access any of the filters that fall under a wavelength range they want to use. To achieve this, the interface visualizes the wavelength ranges of all filters (i.e. filter spans) which are overlapping with the specified range, by using a span chart (that can also be stacked as the no. of filter spans plotted will be large).



Wireframe 3.1.2

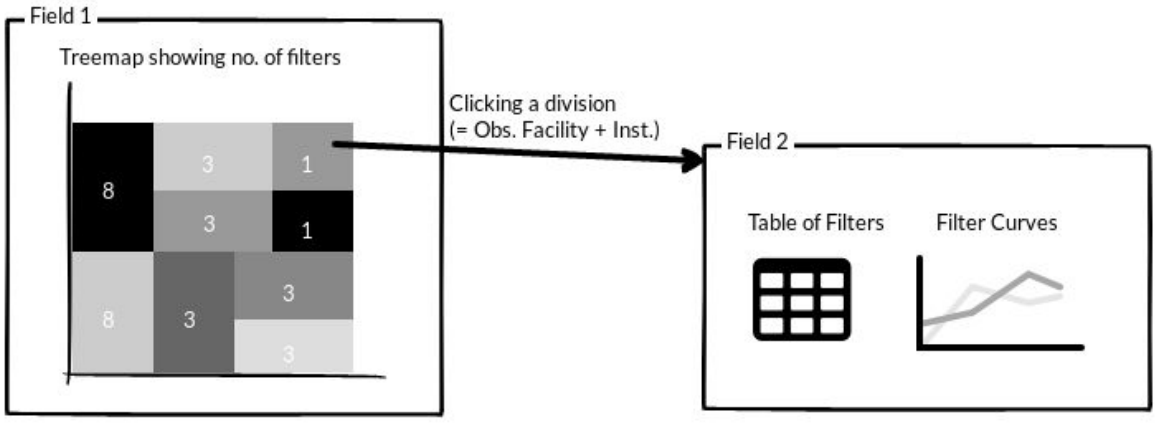
User can enter the full range or just the maximum or minimum limit of the range by using a range slider. Then the filter spans are visualized which can be clicked (or hovered over) to see the curve and information of the selected filter. The way this interface will work interactively is somewhat similar to this drug discovery app.

There is also an option to further filter out the filter spans (as shown in radio buttons group in the right side of span chart):

- Option 1: To get such filters which alone can cover entire specified range (like filter R completely overlapping with range A-B)
- Option 2: To get set of filters which together can cover specified range (like filters P,Q,S,T partially overlapping with A-B)
- Option 3: Same as above but filters are restricted to extend outside the specified range (like filters P,Q,T partially overlapping but strictly within A-B)
- Option 4: All filters that together or alone cover the specified range (like all filters P,Q,R,S,T overlapping with A-B)

3.1.3. By comparing no. of filters distributed among Observation Facilities and Instruments

It allows user to explore the hierarchy of filters (Obsv. Facility -> Instrument -> Filter) and to compare the distribution of no. of filters among them. The treemap visualization is used in which each colour represents Observation facility and each division represents an Instrument which will be labelled with no. of filters. By using this visualization, user can get glimpse of all observation facilities and instruments present in the dataset, at once.



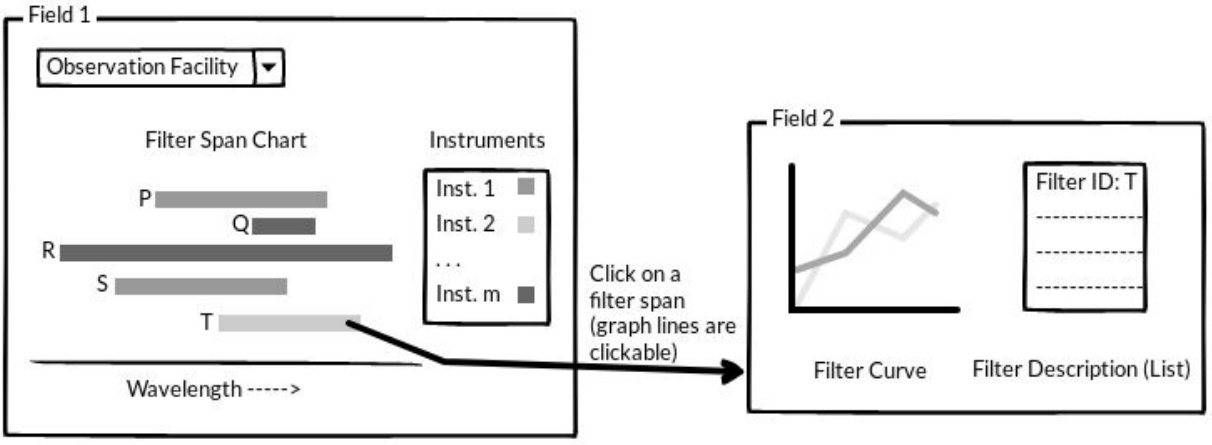
Wireframe 3.1.3

User can click or hover over a division (i.e. either an instrument of Obsv. facility, or Obsv. facility with no instruments) to see the filter curves and table belonging to it.

Alternatively, two interlinked bar charts - one showing no. of filters per Obsv. facility and other showing per Instrument can be used in place of treemap if mentors don't prefer it.

3.1.4. By comparing wavelength range of the filters of a specified Observation Facility

This interface allows user to compare the wavelength range of all the filters (filter spans) falling under a specified Observation facility. It uses span chart to visualize filter spans where each color indicates the instruments they belong to (i.e. grouping data points by color).

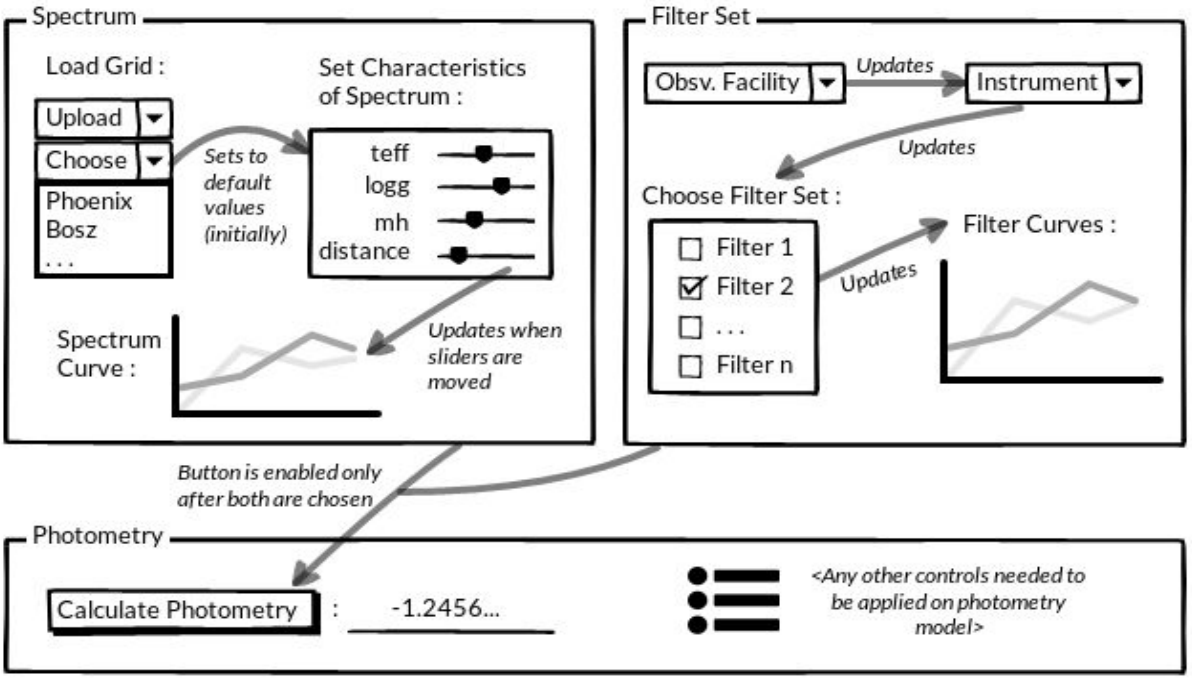


Wireframe 3.1.4

User first selects the observation facility from the dropdown, then filter span chart for the selected facility appears. After which they can click or hover over a filter span to see the information and curve of the chosen filter.

3.2. Interface to calculate Photometry

The main aim of this interface is to allow the user to select an arbitrary combination of filters and spectrum to calculate the photometry; by providing UI controls to quickly input the required values of characteristics. The interface will also visualize the curves of both chosen spectrum & filter set, besides offering options to choose. Once both of these are chosen, a code (in back end) similar to this example will be executed to automatically calculate the photometry.



Wireframe 3.2

User can either upload the spectral grid (from their local computer) or choose the grids like Phoenix (already stored at backend). The default values of spectrum characteristics get set after grid is loaded which can be changed by the user. For each combination of these values spectrum curve is also shown.

The filters can be added to filter set after making selections from dropdowns as shown in the wireframe. For all the filters selected, the interface also shows the filter curves. The code at backend will make sure that *those filters which do not lie in the wavelength range spanned by the chosen spectral grid*, are automatically disabled. Besides using observation facility & instrument dropdowns, the other ways to access filter curves can be integrated by linking it with the interfaces developed earlier, which will be achieved under goal 2.2.2.

After selecting both spectrum and filter set, a button is clicked to calculate photometry. Further, any controls that need to be applied in calculation of photometry (like setting characteristics of photometry model, etc.) will be developed & added to the interface (under goal 2.2.3).

Filter transformations will also be enabled within this interface for which user will be allowed to input the magnitude (Photometry value) for the selected spectrum under a certain filter. Then by using this, scaling factor will be calculated (at backend) which will be used to predict the photometry of spectrum under another chosen filter. For implementing this at front end, a tab named “Filter Transformations” can be added in Photometry field, in which user will first input the magnitude for the selected combination of spectrum & filter, then they will be asked to re-choose the filter for predicting the magnitude of spectrum under it.

3.3. Deployment of the developed interfaces

Depending on the tool chosen for development, the interfaces will be integrated in the docs as follows:

- **Dash:** First, the interfaces will be deployed to a server (free cloud platform like heroku or pythonanywhere) and then they will be embedded within docs using <iframe> element pointing to the url of deployed interface. This will make the interactive Dash interfaces run directly on the static HTML pages of docs.

- **Jupyter Notebook:** To make ipywidgets run, a Python kernel is necessary thus the notebooks (interfaces) will be uploaded to a GitHub repository with a dependency file. Then using Binder the repository contents will be hosted and shareable link to a live binder repository that can run notebooks online, will be produced. This link (along with a launch binder badge) will be embedded in the docs and a GIF will also be placed to show user the working of interface, therein the docs (since the Binder link will redirect them). The user may also opt to download the notebook from repo and run it locally.

After integrating interfaces, proper content will be added to the docs to illustrate the working of each interface. The aim will be to put together all the developed interfaces in a presentable form in the docs.

3.4. Additional Interfaces

While developing above interfaces, many other requirements (i.e. functionalities that are not provided by these interfaces) may emerge. Thus I have accounted time to develop those functionalities as additional interfaces under secondary goal. One such functionality which should be developed is:

Distribution of filters across the wavelengths of EM spectrum:

An interface can be developed that visualizes a histogram showing the no. of filters for different wavelength intervals in EM spectrum. Each band of EM spectrum (considering from UV to Infrared) will be represented in the histogram by means of corresponding colors (7 colours of the visible band, light red for infra-red and dark violet for UV).

Besides this, interface for other functionalities as suggested by mentors will be developed. Another noteworthy aspect of this project is:

Flexibility of the Project:

The way of implementation of each proposed interface is completely flexible to changes. Since there are uncountable ways to tell a story through data, the visualizations used in each interface can be modified as suggested by mentors. The changes can be anything from adding a UI control to changing the entire visualization technique. A perfectly suitable visualization is devised after trials & errors, therefore enough scope is accounted in timeline to refine the visualizations - simultaneously within the week of interface development (likely on weekends) and dedicatedly under the refinement phase (goal 2.2.3).

4. Timeline

4.1. Weekly Timeline

Following is an indicative schedule of how I have planned to achieve project goals with the milestones:

Review Period:

Apr 10 - May 5 (3.5 Weeks)

- Fixing the Travis CI in wsynphot
- Fixing the installation error in wsynphot (working on PR#6)
- Making the environment file of wsynphot similar to that of starkit (working on PR#7)
- Putting installation steps in readme of wsynphot
- Putting installation steps of both starkit & wsynphot in their respective Sphinx docs

Milestone #0 : Set up wsynphot package for robust user installation

Community Bonding Period:

May 6 - May 26 (3 Weeks)

- Fixing the error in loading certain filters (issue#13)
- Discussing & finalizing the choice of development tool
- Getting familiar with Plotly library & chosen tool
- Getting familiar with relevant topics from Astronomy
- Understanding community working culture of Starkit

Coding Period:

Week 1: May 27 - Jun 2

Writing code to create plotly plots for filter curves which will be used in developing the following interfaces; Adopting the best way to visualize the spaghetti plot of curves when plotted together; Setting up environment of wsynphot package for plotly.

Week 2: Jun 3 - Jun 9

Getting familiar with either Dash or figurewidgets (for Jupyter Notebook); Development of the interface described in section 3.1.1; Identifying the suitable way to plot wavelength span chart of filters and writing the code for same (part of interface 3.1.2).

Week 3: Jun 10 - Jun 16

Finishing development of the interface as described in 3.1.2; Development of the interface described in section 3.1.3.

Week 4: Jun 17 - Jun 23

Development of the interface described in section 3.1.4; Polishing off the development of all 4 interfaces and finding a way to present them together.
Milestone #1 : The interfaces to access filter curves are developed (Goal 2.1.1 achieved).

Week 5 (Evaluation week): Jun 24 - Jun 30

Understanding the setting up of grid, spectrum & concerned procedure to calculate photometry; Initiating discussion on implementation details about same; Finalizing the complete implementation logic; Writing up dev blog for work until now.

Week 6: Jul 1 - Jul 7

Development of the interface as per decided logic & as described in section 3.2 (except the filter transformations); Tuning all developed functionalities to work together in the interface.

Week 7: Jul 8 - Jul 14

Development of the “Filter Transformations” functionality & finishing up the interface 3.2; Investigating the chosen method of deploying interfaces and finalizing the choice of server.
Milestone #2 : The interface to calculate Photometry is developed (Goal 2.1.2 achieved).

Week 8: Jul 15 - Jul 21

Deployment of the interfaces by following the procedure described in section 3.3; Writing brief user guide for each interface; Polishing off the integration of deployed interfaces in docs.
Milestone #3 : The developed interfaces are deployed (Goal 2.1.3 achieved).

Week 9 (Evaluation week): Jul 22 - Jul 28

Building docs of the modules & functions of wsynphot using autodoc (goal 2.2.5); Writing up dev blog; Initiating discussion to devise the need of additional interfaces.

Week 10: Jul 29 - Aug 4

Development of the decided additional interface(s), one can be as described in this section (goal 2.2.1); Initiating discussion to identify the refinements required in any of the interface (modifications like any additional UI control, enhancing look, etc).

Week 11: Aug 5 - Aug 11

Investigating suitable way to link the interfaces to access filter curves with the interface to calculate photometry and implementing the same (goal 2.2.2); Accomplishing the identified refinements required in the interface(s) (goal 2.2.3).

Week 12: Aug 12 - Aug 18

Investigating the suitable way to write tests for developed interfaces; Writing the tests & debugging the interfaces for any errors encountered (goal 2.2.4).

Final Evaluation Week: Aug 19 - Aug 26

Updating the documentation & final wrap up of any loose ends in the work (goal 2.2.6); submission of code.
Milestone #4 : All the secondary goals are accomplished (Goal 2.2 achieved).

Note: The updated timeline is to be considered from the most recent version of this proposal at http://bit.ly/jaladh-singhal_gsoc19_starkit.

4.2. Other commitments

I will be having my end term exams for 5 days in April end (that will be during Review period). Since the exam period is short so I can manage to finish the scheduled work on time. Also, I will have my semester examinations for about 15 days after mid May (actual date will be released in May beginning). Hence, if they overlap with the 1st week of Coding period (or 2nd week in worst case), I will complete the scheduled work earlier in the Community bonding period itself. Since I will be having summer vacations immediately after the exams, I can complete the delayed work (if any) by spending more hours per day.

Except these I have no other commitments for the Summer. I have planned to devote 35-40 hours per week to the project and upto 50 hours per week in vacations (possibly from mid June to mid July).

5. About me

5.1. Personal Information

- **Name:** Jaladh Singhal
- **E-mail address:** jaladhsinghal@gmail.com
- **Github/Gitter username:** jaladh-singhal
- **Location:** Jaipur, Rajasthan, India
- **Time Zone:** India Standard Time (GMT+5:30)

5.2. Links to contributions

1. PR#10: Fixed the error in building Sphinx docs of wsynphot (merged).
2. PR#8: Fixed the label error in plot() function of wsynphot (y-axis label was not visible).
3. PR#9: Created a notebook to plot Filter curves based on selection of any Observation facility & Instrument, and included it in docs.
4. PR#12: Integrating the list of filters in living Sphinx docs i.e auto generated by using Jinja templating (work in progress - won't be necessary after interface 3.1.1 is developed).
5. PR#6: Tried to fix setup install error in wsynphot (will finish it under Milestone#0).
6. PR#7: Tried to fix Travis CI & install error by making the environment file of wsynphot similar to that of starkit (will also finish it under Milestone#0).
7. Issue#13: Identified certain filters which throw error in loading (will be fixed during the community bonding period after discussing with mentors).
8. PR#37: Fixed the error in readme of starkit due to changes in Anaconda disto (fixed broken link & added alternative command to activate conda environment).

5.3. Background Info

I am a Computer Science Engineering sophomore, pursuing my graduation at Swami Keshvanand Institute of Technology, Management & Gramothan (SKITM&G). I learned C as my first programming language, which helped me to build a strong base in programming. I've developed a creative fun project in C named Frequency Composer which creates music by varying the frequency values of sound beeps. Then I started learning & using Python for writing automation scripts which made me fell in love with this language. To practically apply my Python knowledge, I completed two internships last year - one of Database Building and other of building Web Crawlers, which strengthened my grip over Python. The details of my experience can be found in my resume. Being amazed by the beauty of Data Visualization, I further started learning the same this year. And I experienced the real power of Python which I am enjoying to harness in this project of Starkit.

I am also a great fan of science & astronomy and spend hours on internet, reading countless fascinating articles on the same. I am a keen stargazer who always remains ready to stay on rooftop overnight if there is an astronomical event, even when it is a chilling winter night! Due

to this passion for astronomy, I decided to contribute to an Org of same domain this year, under GSoC. At the same time, I also wanted to continue learning data visualization by practically applying my knowledge in a GSoC project. And fortunately, I found the right organisation - Starkit with the right project and now I am more than happy working for it.

Lastly, I would like to mention that I am a curiosity-driven learner. All the programming knowledge I have gained so far, is the result of my endless exploration & self-learning capabilities. I do not hesitate in learning new stuff if it piques my interest; the same resulted in development of a Running Game (in Processing.js) entirely from scratch, in which runner can become the superhero Flash. Although it is my first time in Open Source contribution, but what I have learned in mere one month was unimaginable for me. I have learned about virtual environments, Anaconda, Ipywidgets, Python package structure, Sphinx docs, Jinja templating, multiple Data Visualization techniques, Plotly, Dash and what not! I am thrilled to take this learning journey farther by not only contributing to Starkit but to entire Open Source world.

I have been contributing solely to Starkit since past month, and with the constant support of mentors, I have engaged long enough with the project that I'm crystal clear about the objectives of this project. After coming up with well-thought ideas and preparing an actionable plan to achieve them, I am confident that I can finish the project within proposed timeline by the support of community. I will be extremely happy to be a part of Starkit in this Summer (and even later) to evolve not only the wsynphot package, but also myself as a developer.