

Project title: *Fortification of the hyperSpec R Package*

Short title: *hyperSpec Fortification*

URL of project idea page: <https://github.com/rstats-gsoc/gsoc2020/wiki/hyperSpec>

Fortification of the hyperSpec R Package

Student Applicant: **Erick Oduniyi (eoduniyi@gmail.com)**

Project Mentors: Bryan Hanson (hanson@depauw.edu),

Claudia Beleites (claudia.beleites@chemometrix.gmbh),

Roman Kiselev (roman.kiselev@stjude.org)

Student Biography

Overview

Contributing to important and novel technologies is my passion. Everything from how technology is designed to interact with users, the manufacturing innovations that technology requires to reach multiple users, and the physical and/or digital forms technology can take over generations continues to fascinate me. Now, as senior electrical engineering and computer science (EECS) student I continue to develop my passion for technology and for applying mathematical, computational, and digital tools from EECS to the study and augmentation of biological, cognitive, cultural, and electro-mechanical systems. Concretely, my background is rooted in my scientific and research experiences: employing cognitive and software-defined radios for mobile spectrum sensing, modeling the impact of media engagement on infectious diseases, utilizing sentiment analysis and physiological sensing (e.g., skin conductance, facial electromyography) for storytelling and cognition, and the design of speech recognition systems as a representation of early language acquisition. Unifying these diverse research interests and experiences is Systems theory – *the interdisciplinary study of natural and man-made systems*. Ultimately, through the application of systems theory (and systems thinking), my research and technology development aims to reveal the broad complex communication paradigms within and between animal, machine, plant, and environment.

Qualifying Background

My education as an undergraduate EECS student has provided me with foundations in programming, data structures, and software engineering, all of which are critical to successful open-source development. Beyond the classroom, I have been fortunate enough to undertake numerous research experiences and internships. In particular, this past summer I was a research intern at Harvard University’s Long-Term Ecological Research site (LTER), Harvard-Forest (HF). At HF, I was working under the **end-to-end-provenance** (E2E) team where I had the opportunity to be mentored by computer science and ecology professors and graduate students in R and open-source software development. During my time as an intern, I wrote a proposal for a new R package called **provBookR**, and collected informal user requirements from HF scientists and students. From there, I processed the user requirements to generate use cases and flow diagrams (a common practice in software engineering and product design). After generating user requirements I spent the rest of the summer developing the provBookR software and learning about the open-source R data provenance tools and packages created by the HF E2E team. In the end, this experience was extremely valuable because it introduced me to data provenance, open-source development (working with in-person and remote teams), R package development, and being comfortable with going through code repositories and documentation until my eyes fall out. Additionally, during previous internships¹ (i.e., Santa Fe

¹For more information about me please see a current [CV](#) and [writing sample](#)

Institute, University of Kansas Ecology & Evolutionary Biology Department) I primarily used R for data analysis and synthesis. In short, I believe I am qualified to learn and contribute to the hyperSpec project for R Google Summer of Code 2020.

Contact Information

Student name: Erick Oduniyi
Student postal address: 15502 E Lynnwood St, Wichita, KS 67230
Telephone(s): +1 316-990-1410
Email(s): eoduniyi@gmail.com
Other communications channels:

- Skype: Erick Oduniyi (+1 316-990-1410)
- Hangouts: eoduniyi@gmail.com

Student Affiliation

Institution: Wichita State University
Program:

- Degree: Undergraduate
- College: College of Engineering
- Major: Computer Engineering

Stage of completion: Senior Undergraduate
Contact to verify: Neal Hoelting

- Email: neal.hoelting@wichita.edu
- Phone: +1 316-669-3580

Schedule Conflicts:

If accepted, I will have no other schedule conflicts this summer and will treat R GSoC 2020 as a full time (40+hrs/week) opportunity.

Mentors

Evaluating mentor name and email: Bryan Hanson (hanson@depauw.edu)
Co-mentor name(s) and email(s):

- Claudia Beleites (claudia.beleites@chemometrix.gmbh)
- Roman Kiselev (roman.kiselev@stjude.org)

I have been in touch with all three mentors since March, 24, 2020 through GitHub and email.

Coding Plan and Methods

Project Background

The **hyperSpec** package allows R users a suite of utilities for manipulating spectroscopic data [1]. These utilities currently include functions for importing spectroscopic data (e.g., Raman, IR, NIR, UV/VIS), plotting, pre-processing and wrangling spectroscopic objects (i.e., hyperSpec objects) so that they can be easily integrated with commonly used statistical analysis packages (e.g., **plsR** and **MASS**). On the one hand, the usefulness of the hyperSpec package for spectroscopic research is clear and immediately useful to R programmers and practitioners of bioinformatics, chemoinformatics, and medical statistics. On the other hand, the infrastructure afforded by hyperSpec has led to various dependencies on other R packages, large amounts of test data, and more than a hundred .R files. Accordingly, the current state of the package has become difficult to maintain and vulnerable to fixes related to dependencies on deprecated packages. The project *Fortification of the hyperSpec R Package* aims to tackle these issues directly by accomplishing three goals.

Project Goal

At a high-level, the three goals of the proposed project are the following: *distilling* the original hyperSpec package, *shielding* against package dependencies, and *bridging* hyperSpec with other relevant packages. Accomplishing all three goals will result in the fortification of hyperSpec's software architecture and the creation of a new ecosystem of R spectroscopic tools.

Goal 1: Distilling hyperSpec

The first goal works to reduce the package footprint by distilling and retaining core computationally inexpensive functionality within the original hyperSpec package and outsourcing computationally expensive functionality (i.e., file import functions) across smaller specialized packages. The result of this goal will make the original and subsequent hyperSpec packages easier to maintain.

Goal 2: Shielding hyperSpec

The second goal works to protect the package by striving for a balanced approach to package dependencies. This goal is less about making sure hyperSpec is completely decoupled from other packages and more about ensuring the features and procedures the hyperSpec packages depends on come from packages that are actively maintained on CRAN (or non-CRAN packages that are themselves reliable). For instance, if hyperSpec depends on two packages that provide the same functionality, then hyperSpec should only rely on one of those packages (assuming there is not a straightforward base R solution); ideally the package itself has few dependencies and/or has a longer life history on CRAN (i.e., fewer updates) and success within the R developer community (see figure 1). Because CRAN package errors are commonly the result of updates in package dependencies, the result of this goal will bring control back to hyperSpec and make the hyperSpec project anti-fragile.

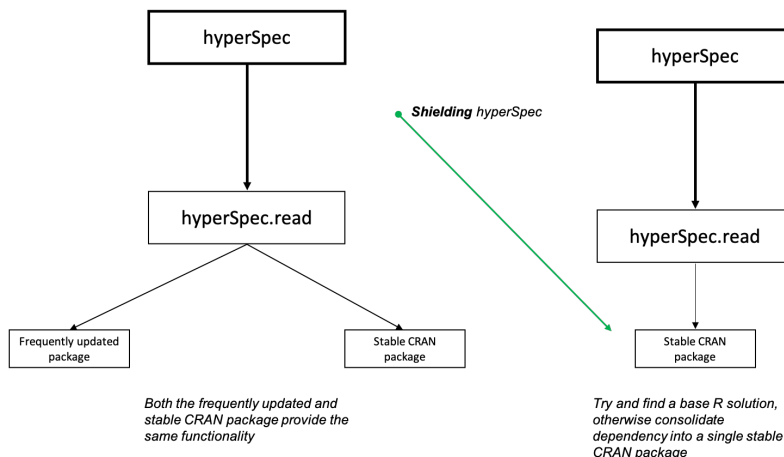


Figure 1: Shielding hyperSpec from unnecessary package dependencies

Goal 3: Bridging hyperSpec

The third goal works to build packages that make hyperSpec and hyperSpec objects easier to integrate with relevant data cleaning, spectroscopic, and statistical analysis packages. For example, integrating hyperSpec with spectra pre-processing packages like **baseline** and **EMC**, data wrangling and plotting packages like **dplyr** and **magrittr**, and **ggplot2** (see figure 2); file import functions like the **readJDX** package. The result of this goal will generate several small hyperSpec companion packages (e.g., **hyperSpec.read**, **hyperSpec.tidyverse**, etc.).

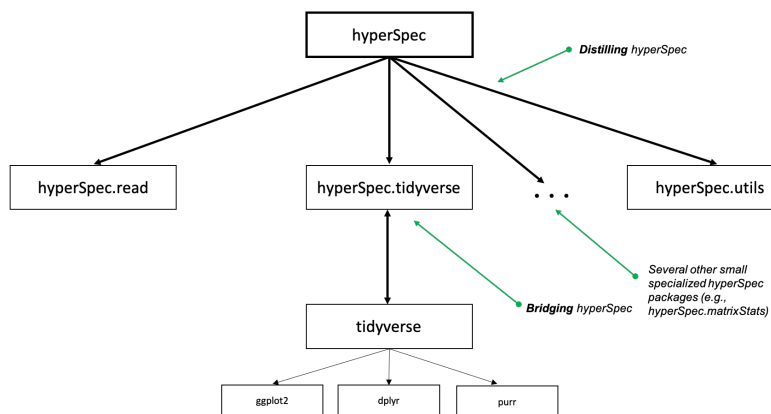


Figure 2: Software fortification of the hyperSpec R package

Project Planning

Whether developing big or small software projects, robust software development is no easy task. Software development requires extensive design and planning, clear implementation, verification, and testing; all of which should be done iteratively. In fact, software projects regularly require a team of developers that span these tasks: designing, implementing, reviewing, testing, and documenting.

However, because of the nature of open-source projects, one, more, or all of these tasks can fall on the shoulders of a single developer. As a result, the coverage of these tasks can vary widely. The beauty of open-source is that through engagement and strong communication, solo endeavors turn into community projects that support contributors from around the world that work together to develop robust software systems.

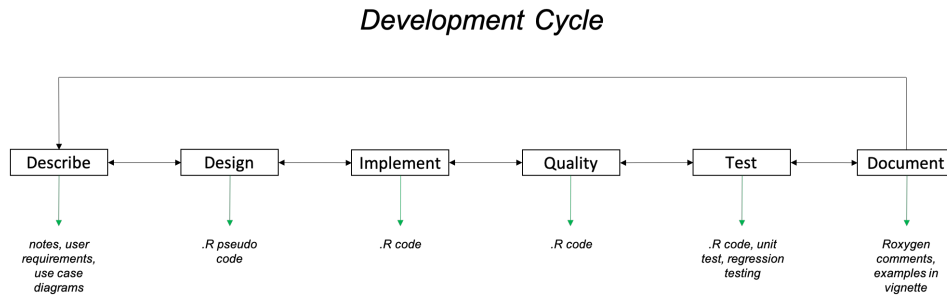


Figure 3: Software development plan for making progress on coding project

Coding Plan

From the three high-level goals, it is clear that my coding project is package fortification and lean package development. A plan that proposes to execute this coding project and as a result the goals of the project is the following development cycle: *generate software description, function design, function implementation, quality assurance, testing and verification, documentation, deliverables* (see figure 3). At the heart of successful communities is planning and clear communication. These are the guiding principles of my coding project.

Software Description

I will always first “sketch” out the description of the task that need to be completed as specified by, and agreed upon with my mentors. I expect these tasks to often come in the form of GitHub issues or through other communication channels (i.e., weekly meetings). Depending on the complexity and formality of the task at hand, these description sketches will take the form of briefly typed notes or more formal mock-ups of user requirements and use case diagrams (as shown in figure 4).

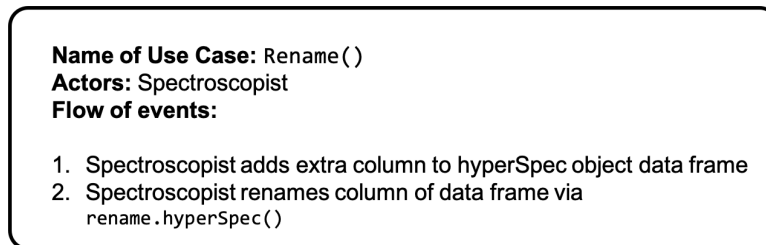


Figure 4: Example use case diagram from `dplyr::rename()`

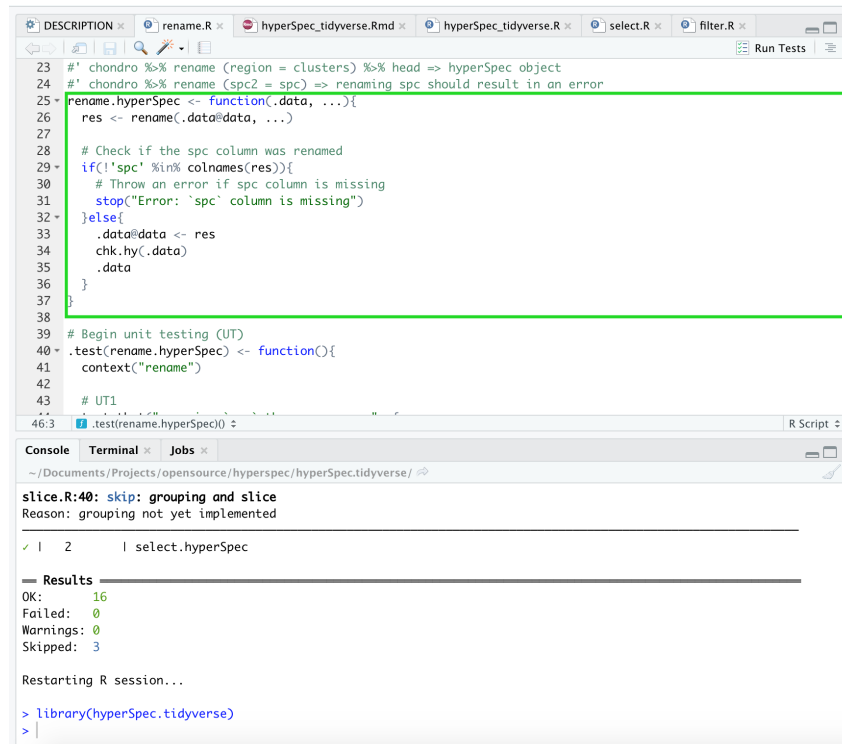
Function Design

Depending on the complexity of the software description, I will develop R pseudo-code specifying function assumptions, inputs, outputs, and side effects. I find it helpful to generate software descriptions and function designs in R Markdown first.

```
# Psuedo code for hyperSpec.rename
rename.hyperSpec <- function(.data){
  res <- dplyr::rename(.data@data)
  return(res)
}
```

Function Implementation

For function implementation I will follow Hadley Wickham's R code workflow as discussed in his [R packages book](#): *create .R file giving it an appropriate and clear name, explore the code in the console, "rinse and repeat" until the function is implemented as specified (see figure 5).*



The screenshot shows the RStudio interface with several open files. The main editor displays the following R code:

```
23 #' chondro %>% rename (region = clusters) %>% head => hyperSpec object
24 #' chondro %>% rename (spc2 = spc) => renaming spc should result in an error
25 rename.hyperSpec <- function(.data, ...){
26   res <- rename(.data@data, ...)
27
28   # Check if the spc column was renamed
29   if(!'spc' %in% colnames(res)){
30     # Throw an error if spc column is missing
31     stop("Error: 'spc' column is missing")
32   }else{
33     .data@data <- res
34     chk.hy(.data)
35     .data
36   }
37 }
38
39 # Begin unit testing (UT)
40 .test(rename.hyperSpec) <- function(){
41   context("rename")
42
43   # UT1
44   .test(rename.hyperSpec())
```

The console output shows the following results:

```
slice.R:40: skip: grouping and slice
Reason: grouping not yet implemented

✔ | 2 | | select.hyperSpec

--- Results ---
OK:      16
Failed:   0
Warnings: 0
Skipped:  3

Restarting R session...
> library(hyperSpec.tidyverse)
> |
```

Figure 5: Implementing dplyr::rename() in RStudio

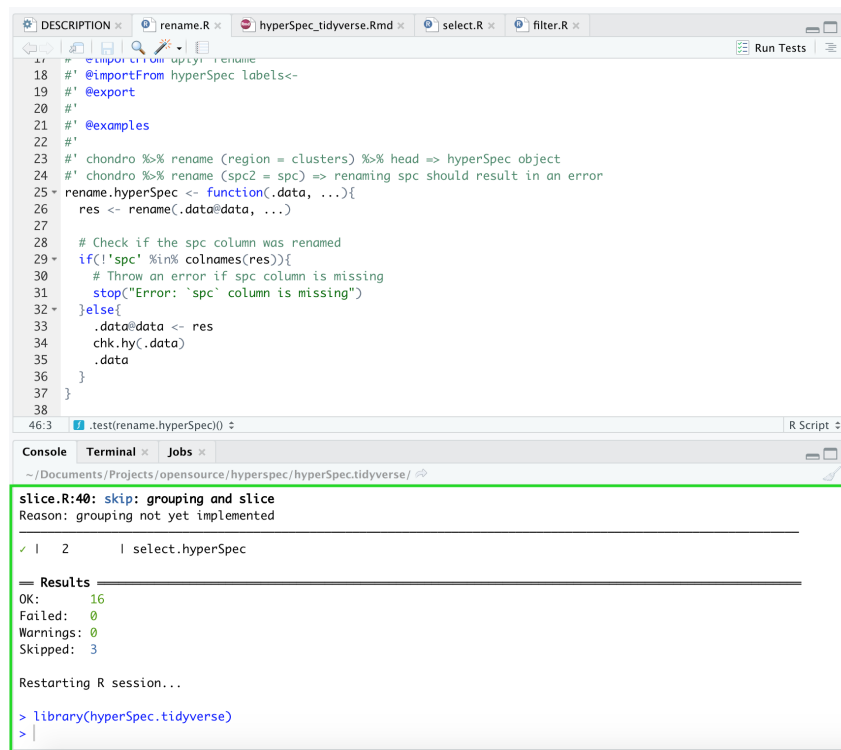
Quality Assurance

I will write anti-fragile code that can be refactored, extended, and tested easily. At a high-level, this means the code will not be unnecessarily clever or confusing to my mentors, other students, and

especially my future self. Any code that is written will adhere to the style/standard of the hyperSpec code base.

Testing and Verification

For each function and its associated code, I will make sure to develop comprehensive unit test. Depending on the complexity of the function (simple functions versus functions with side effects like file import/reading functions) prepare tests that check for points of failure and potentials bugs (see figure 6).



```
DESCRIPTION x rename.R x hyperSpec_tidyverse.Rmd x select.R x filter.R x
18 #' @importFrom hyperSpec labels<-
19 #' @export
20 #'
21 #' @examples
22 #'
23 #' chondro %>% rename (region = clusters) %>% head => hyperSpec object
24 #' chondro %>% rename (spc2 = spc) => renaming spc should result in an error
25 rename.hyperSpec <- function(.data, ...){
26   res <- rename(.data@data, ...)
27
28   # Check if the spc column was renamed
29   if(!'spc' %in% colnames(res)){
30     # Throw an error if spc column is missing
31     stop("Error: `spc` column is missing")
32   }else{
33     .data@data <- res
34     chk.hy(.data)
35     .data
36   }
37 }
46:3 test(rename.hyperSpec)

Console Terminal Jobs x
~/Documents/Projects/opensource/hyverspec/hyperSpec.tidyverse/

slice.R:40: skip: grouping and slice
Reason: grouping not yet implemented

✓ | 2 | select.hyperSpec

== Results ==
OK:      16
Failed:  0
Warnings: 0
Skipped: 3

Restarting R session...

> library(hyperSpec.tidyverse)
>
```

Figure 6: Testing implementation of dplyr::rename() in RStudio

Documentation

I will make sure each .R file is appropriately documented (i.e., parameters, examples, etc.) using **roxygen** and **knitr**. I will also make sure to comment sections of the code that highlight assumptions or any outstanding nuances. Finally, for each function and feature I will make sure there is a brief explanation and example in the package vignette (see figure 7).


```

123 as.hyperSpec()
124
125
126 The resulting `hyperSpec` object has 0 wavelengths: its `$spc` column contains a spectra matrix with 0
127 columns, and its wavelength vector is also of length 0.
128 Behind the scenes, the `data.frame` returned by `select()` gets an attribute `labels` which stores the
129 labels of the `hyperSpec` object.
130 `as.hyperSpec()` restores the labels from this attribute (if available).
131 Therefore, the "back-converted" `hyperSpec` object has its labels preserved.
132
133 ## Renaming extra data columns: `rename()`
134 `rename()` renames extra data columns from the `hyperSpec` object. Unlike `select()`, `rename()` keeps all
135 the variables of the data frame intact.
136 ```{r, results='show'}
137 chondro %>%
138   rename(region = clusters)
139 ```
140
141 ## Selecting wavelength ranges
142
143 TODO
144
145 137:1 Renaming extra data columns: `rename()`

```

Figure 7: Documenting rename.hyperSpec in hyperSpec.tidyverse.Rmd

Deliverables

The highest level deliverable is a well documented and tested R package that distills, shields, and bridges hyperSpec with relevant R packages and tools. Of course, this high-level deliverable depends on smaller deliverables (i.e., well tested and documented functions and code; see figure 8). In fact, the nature of the coding project is to deliver usable results as quickly as possible. As soon as finer pieces of the project are delivered, they will work as scaffolding; encourage team development and project growth throughout the summer.

```

~/Documents/Projects/opensource/hyverspec/hyperSpec.tidyverse/
> library(hyperSpec.tidyverse)
> load_all()
Loading hyperSpec.tidyverse
> chondro %>% rename.hyperSpec(region = clusters)
hyperSpec object
  875 spectra
  5 data columns
  300 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 602 606 ... 1798
data: (875 rows x 5 columns)
  1. y: y [numeric] -4.77 -4.77 ... 19.23
  2. x: x [numeric] -11.55 -10.55 ... 22.45
  3. filename: filename [character] rawdata/chondro.txt rawdata/chondro.txt ... rawdata/chondro.txt
  4. region: [factor] matrix matrix ... lacuna + NA
  5. spc: I / a.u. [matrix300] 501.8194 500.4552 ... 169.2942
> chondro %>% rename(spc.newcolumnname = spc)
Error in rename.hyperSpec(., spc.newcolumnname = spc) :
  Error: `spc` column is missing

```

Figure 8: Renaming extra data column using rename.hyperSpec()

Project Obstacles

Unfortunately, there is no such thing as perfect and no plan is ever problem-free, so I foresee a couple of obstacles: misunderstanding of software descriptions and deliverables from mentors, lack of time to implement deliverables or spending too much time on a deliverable, and forgetting about unresolved errors, maintaining consistent code quality. To overcome these obstacles I will always keep mentors up to date with progress and set expectations early and adjust them accordingly. In particular, my expectations for a successful R GSoC experience consist of keeping myself enthusiastic and accountable for completing work and associated deliverables throughout the summer, as

well as, considerate of my mentors and other participating students time. Finally, I expect myself to be appropriately open and remain cooperative across disciplines and people in addition to actively listening to concerns, ideas, and issues, so that cross-pollinating conversations are encouraged, and assumptions, biases, and intentions are not misplaced.

Timeline

Rough Timeline

Date	Event
March 31 - May 4	Pre-GSoC Period
May 5 - May 31	Community Bonding Period
June 1 - June 28	Coding Period 1
June 29 - July 3	Phase 1 Evaluations
July 4 - July 26	Coding Period 2
July 27 - July 31	Phase 2 Evaluations
August 1 - August 31	Coding Period 3
August 31 - September 7	Final Evaluations
September 8	Final Results

Detailed Timeline

March 31 - May 4: Pre-GSoC Period

Week 1

Continue to spend time understanding the codebase and solving open issues. Start reading all R documentation related to [hyperSpec](#). (*Make sure I have all of the development tools I need to contribute efficiently and effectively*)

Week 2

Continue to spend time understanding the codebase and solving any open issues. Continue reading documentation related to hyperSpec. Start reading documentation on other [spectroscopy packages](#)

Week 3

Continue to spend time understanding the codebase and solving any open issues and reading documentation related to hyperSpec. Continue reading documentation on other spectroscopy packages. Start reading documentation on spectra pre-processing packages (baseline and EMSC) packages.

Week 4

Continue to spend time understanding the codebase and solving any open issues and reading documentation related to hyperSpec. Continue reading the documentation on other spectroscopy packages. Start reading documentation on data wrangling and visualization packages (ggplot2/tidiverse).

Week 5

Continue to spend time understanding the codebase and solving any open issues and reading documentation related to hyperSpec. Continue reading documentation on other spectroscopy packages.

Start reading documentation on tools related to the importing of spectroscopy data (readJDX).

Week 6

Brush up on R programming. Find good resources for learning base R and **advance R** techniques. Brush up on git and git-lfs.

Week 7

Continue brushing up on R programming skills and on git and git-lfs. *(Make sure I have all of the development tools I need to contribute efficiently and effectively)*

May 5 - May 31: Community Bonding Period

Week 1

Create a clear communication schedule for weekly meetings, protocols for check-ins, and a high-level road map for successful contribution to the hyperSpec project (e.g., forking, merging, pulling, etc.).

Week 2

Finalize development tools and workflow for contributing to the project over the summer. Inquiry about helpful R programming and spectroscopy resources from mentors and digest them.

Week 3

Continue reading resources from my mentors. Start working on software development descriptions for goals, proposed features, functions, and other tasks related to development.

Week 4

Continue reading resources from my mentors. Continue working on software development descriptions for goals, proposed features, functions, and other tasks related to development.

June 1 - June 28: Coding Period 1

Week 1

Let the coding begin! Start making progress on Goal 1. Stick to the development cycle for all weekly tasks (i.e., describe, design, implement, quality check, test, document, deliver, and iterate). Compile a weekly report of progress made. Meet with mentors. *(check-in with mentors as necessary)*

Week 2

Let the coding continue! Continue making progress on Goal 1. Compile a weekly report of progress made. Meet with mentors. *(check-in with mentors as necessary)*

Week 3

Let the coding continue! Continue making progress on Goal 1. Compile a weekly report of progress made. Meet with mentors. *(check-in with mentors as necessary)*

Week 4

Let the coding continue! Wrap up progress on Goal 1. Compile a weekly report of progress made.

Meet with mentors. (*check-in with mentors as necessary*)

June 29 - July 3: Phase 1 Evaluations

This period will be used to write a detailed report on the work done in Coding Period 1. All work completed will be uploaded and documented.

Deliverables:

- Distilled hyperSpec package
- New specialized hyperSpec packages for file I/O
- New implemented import filters for new file formats

July 4 - July 26: Coding Period 2

Week 1

Let the coding continue! Start making progress on Goals 2 and 3. Stick to the development cycle for all weekly tasks. Compile a weekly report of progress made. Meet with mentors. (*check-in with mentors as necessary*)

Week 2

Let the coding continue! Continue making progress on Goals 2 and 3. Compile a weekly report of progress made. Meet with mentors. (*check-in with mentors as necessary*)

Week 3

Let the coding continue! Continue making progress on Goals 2 and 3. Compile a weekly report of progress made. Meet with mentors. (*check-in with mentors as necessary*)

Week 4

Let the coding continue! Continue making progress on Goals 2 and 3. Compile a weekly report of progress made. Meet with mentors. (*check-in with mentors as necessary*)

July 27 - July 31: Phase 2 Evaluations

This period will be used to write a detailed report on the work done in Coding Period 2. All work completed will be uploaded and documented.

Deliverables:

- Shielded hyperSpec and associated hyperSpec packages
- Fortified hyperSpec for tidyverse

August 1 - August 31: Coding Period 3

Week 1

Let the coding continue! Start making progress on Goal 3. Stick to the development cycle for all weekly tasks. Compile a weekly report of progress made. Meet with mentors. (*check-in with mentors as necessary*)

Week 2

Let the coding continue! Continue making progress on Goal 3. Compile a weekly report of progress made. Meet with mentors. (*check-in with mentors as necessary*)

Week 3

Let the coding continue! Continue making progress on Goal 3. Compile a weekly report of progress made. Meet with mentors. (*check-in with mentors as necessary*)

Week 4

Let the coding continue! Wrap up progress on Goal 3. Compile a weekly report of progress made. Meet with mentors. (*check-in with mentors as necessary*)

August 31 - September 7: Final Evaluations

This period will be used to write a detailed report on the work done in Coding Period 3. All work completed will be uploaded and documented.

Deliverables:

- Fortify hyperSpec for baseline with bridge packages
- Fortify hyperSpec for EMSC with bridge packages
- Fortify hyperSpec for matrixStats with bridge packages

September 8: Final Results

All documentation, modules, and tests will be uploaded and Travis CI will be integrated into the project Github page. All the deliverables promised for R GSoC 2020 will be provided by this stage.

Contingency Plan

Again, there is no such thing as perfect and no plan is ever problem-free. Also, life does not wait for coding projects or internships to finish. So, if there is a decline in health or other personal and family issues arise, I will take a deep breath and let mentors know what is going on without TMI². If I foresee a setback in development, I will make sure to communicate my difficulties and get suggestions on how to pivot from mentors so that milestones and deliverables can still be met in a timely manner.

²The nature of the project is “fine grained”, so we expect usable results soon and resilience to possible Covid-19 disruptions.

Additional Information

There are currently no other summer commitments that interfere with R GSoC 2020, so I expect to be working 40+hrs per week. Additionally, I will be using the seven weeks before the coding period to get a running start with development and consistent communication between mentors.

Management of Coding Project

In addition to writing weekly reports, I will maintain a blog that goes over my development progress to ensure I am following my schedule and getting feedback from my mentors. I will be committing multiple times a day and any change in that behavior will be expressed to my mentors beforehand. Testing will be done throughout the summer and during the summer I will be working on setting up Travis for regression testing.

Test

To get prepared for my coding project I have worked on test (tasks) that are easy, medium, hard, and very hard as outlined by my mentors. The tasks ranged from installing the hyperSpec packages to creating GitHub repositories and related skeleton hyperSpec packages. I have submitted solutions for the easy and medium tasks and will continue to work on the hard and very hard task (solutions can be viewed [here](#)) while developing familiarity with the hyperSpec team and codebase. Additionally, while at Harvard Forest I worked on solving similar coding problems as part of an onboarding process. A document that details these problems and output can be viewed [here](#).

Anything Else

Working Environment And Schedule

Monday to Saturday I will be working full-time on the development cycle (i.e., describe, design, implement, quality check, test, document, deliver, and iterate) in a distraction free environment. I typically like to reserve Sundays for periods of rest. My timezone follows Central Daylight Time and will make sure to be aware of my mentor's timezones. I do have a habit of working late at night and early in the morning. However, I am very flexible with my schedule and can adjust accordingly. As for the channels of communication, I am comfortable with any form of communication that suits my mentors. Additionally, If I plan on traveling during the summer I will make sure my mentors are aware of such plans.

References

[1] Claudia Beleites and Maintainer Claudia Beleites. Package 'hyperspec', 2015.