PFalExpTools

Gabriel Foster

2020-03-23

0.1 Introduction

My thesis is entirely based on the generation and analysis of transcriptional data in *Plasmodium falciparum*. This package encompasses many of the computational tools I have built over the years to deeply interrogate this data. There are a large number of functions present in the package; they are mostly used in a small number of analytical pipelines, which I will describe and demonstrate here.

0.2 Curation of Time Course Data

0.2.1 Introduction to Method

Apicomplexan parasites, including *P. falciparum*, have an unusual largely cyclical expression pattern for the majority of genes in their intraerythrocytic development cycle; the details are beyond the scope of this document. Time courses are incredibly valuable for analysis. However they are often sparesly sampled and missing data, due to experimental complications. Having a robust imputed time course is necessary for further analysis. For the coherent analysis of time course data, I've assembled several functions that take full advantage of this cyclicality.

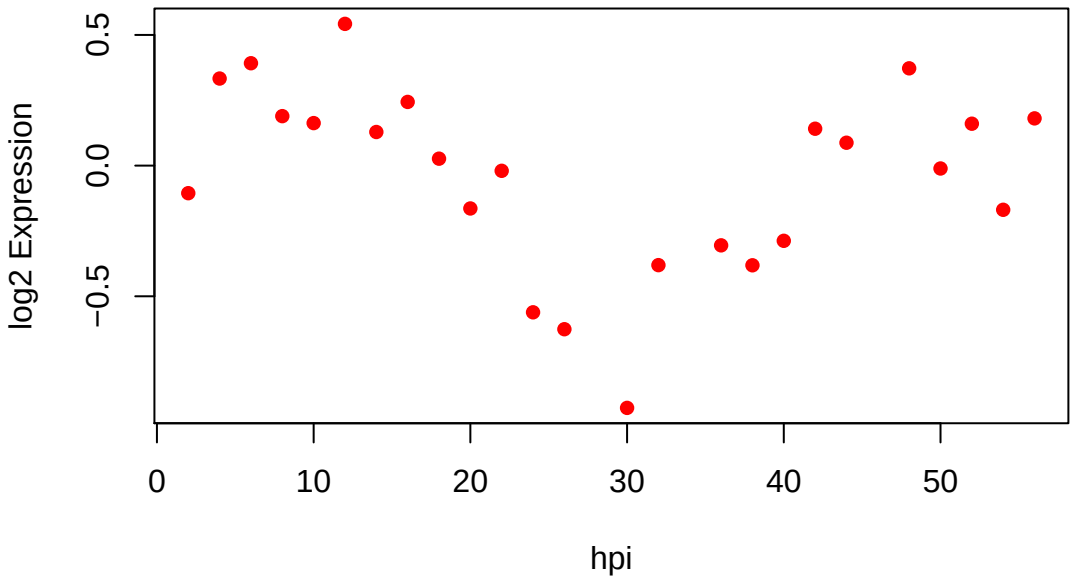
Here, we will work through the processing and analysis of a transcriptional time course.

The time course `NF54` is provided; this is a transcriptional time course of the NF54 parasite, sampled every 2 hours from 2 to 56hpi. The platform for this was the Agilent HD Exon Array (10.1371/journal.pone.0187595). This data is saved as a matrix with rows as genes and columns as samples; note that 3 sample times are present with no data, representing failed samples. Data has already been quantile normalized and summarized to gene level.

Our microarrays were not cohybridized- for comparison to other experiments, it is useful to log2 transform this data. The function `LogTransform` will do this for you. We do not have an average expression for NF54 across the time course, so we'll simply average the rows and use that.

```
library(PFExpTools)

#Get gene average across time course
NF54Avg <- as.data.frame(apply(NF54, 1, function(x){mean(x,na.rm=T)}))

#Get log2 transformed data
NF54log2 <- LogTransform(newdata = NF54, refavg = NF54Avg)

#We had Time as a row; it was log transformed. Let's fix that.

NF54log2["Time",] <- NF54["Time",]

#Example plot
plot(NF54log2["Time",which(!is.na(NF54log2["PF3D7_1343700",]))],
     NF54log2["PF3D7_1343700",which(!is.na(NF54log2["PF3D7_1343700",]))],
```

```
xlab = "hpi", ylab = "log2 Expression", pch = 16, col = "red")
```



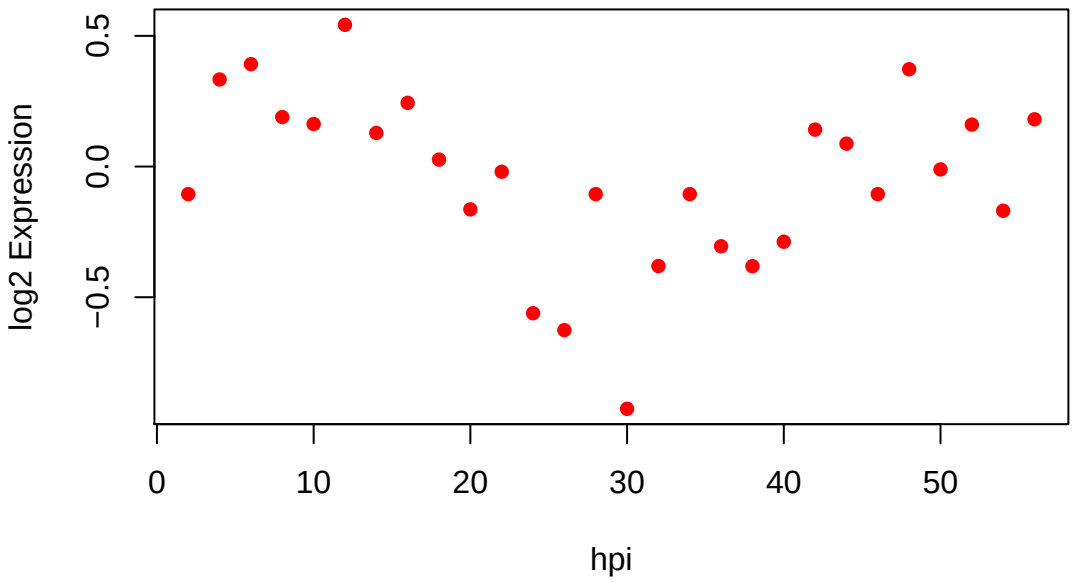
This returns a matrix of log2 transformed data.

If your data already exists as log2 data, you can skip the previous step.

0.2.2 Original Methodology

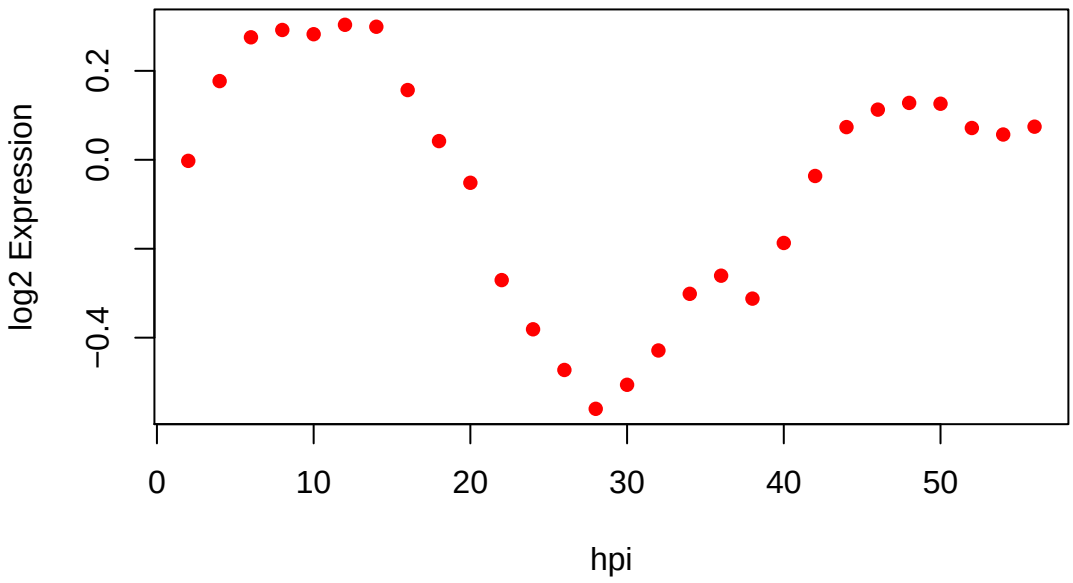
Initially, I used the Fourier transform to evenly impute between time points to generate a higher resolution time course for reference. For that, I need an evenly sampled time course, with no missing time points. The function `tcImpute` handles this for you, using k-nearest neighbor imputation, with the default number of neighbors set to 5.

```
#impute missing time points, using default number of neighbors
NF54noNAlog2 <- tcImpute(NF54log2)

plot(NF54noNAlog2["Time",], NF54noNAlog2["PF3D7_1343700",],
     xlab = "hpi", ylab = "log2 Expression", pch = 16, col = "red")
```



The apicomplexan transcription community has largely hewed to the assumption of cyclicality of expression for the vast majority of genes, with a period being equal to one intraerythrocytic development cycle (approximately 48h for *P. falciparum*). Under this assumption, it is largely assumed that minor variations from this broad cyclicality are assay noise and this is often smoothed out. I've got a function for that, too- `lSmooth` will take care of this for us. This function uses the `loess` algorithm in base R; the default smoothing value is 0.3, and can be set by the user.

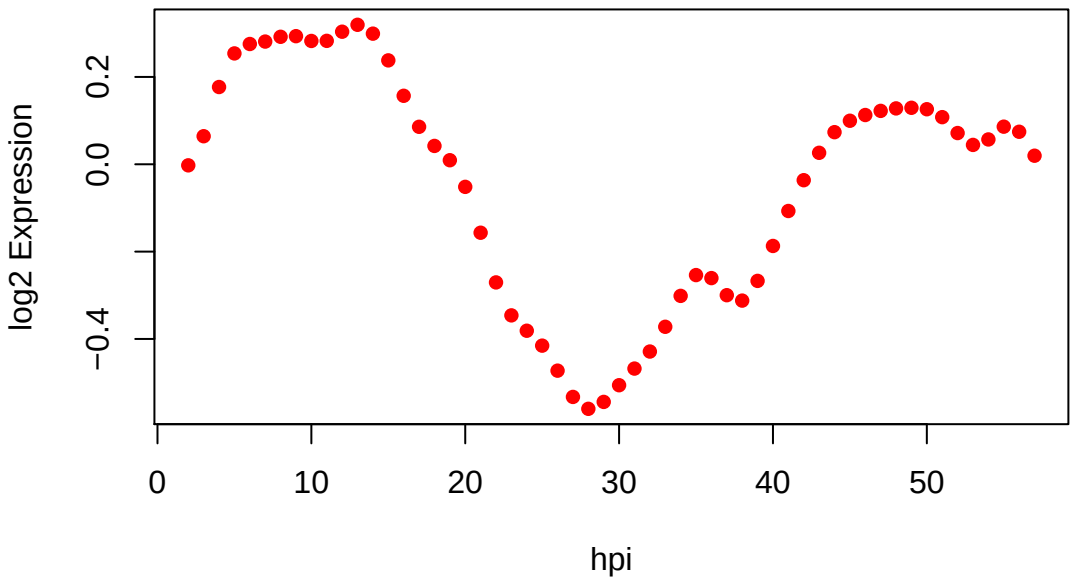
```
#We need a Time matrix for the function; pull from NF54log2
NF54Time <- data.matrix(NF54noNAlog2["Time",rownames.force = T])

#Smooth data, with default
NF54smooth <- lSmooth(expdata = NF54noNAlog2, time = NF54Time)

plot(NF54smooth["Time",], NF54smooth["PF3D7_1343700",],
     xlab = "hpi", ylab = "log2 Expression", pch = 16, col = "red")
```



The loess smoothing here is the same method used by the original *P. falciparum* transcriptional time course paper (https://doi.org/10.1371/journal.pbio.0000005). Unfortunately, it does do some expression magnitude compression, and that isn't great.

At this point, I imputed points to generate an hourly time course using `fftimpute`. This function is incredibly specific to my data, unfortunately. The function uses zero padding to generate an hourly time course.

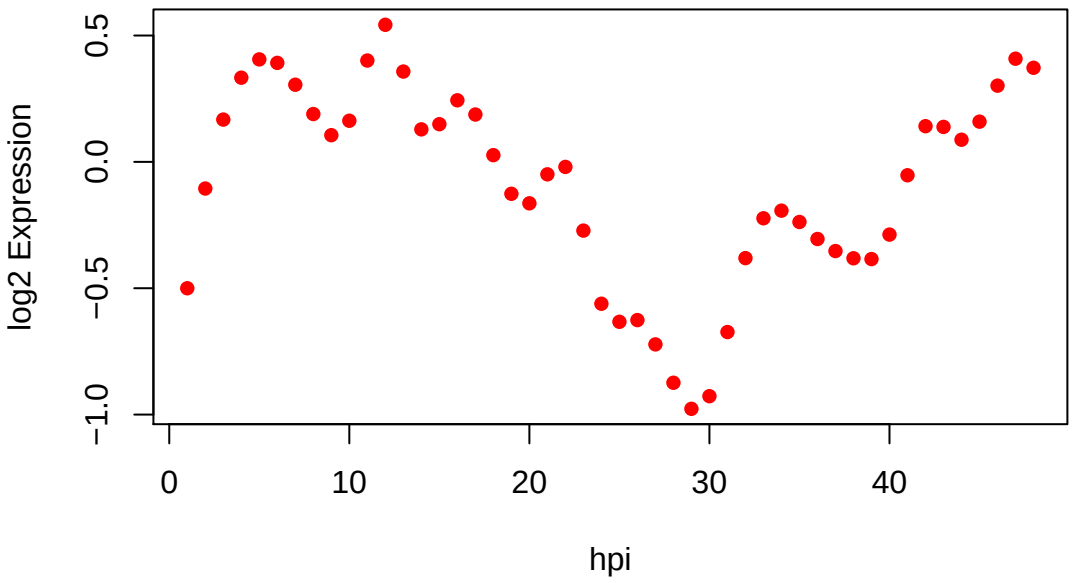
```
#impute time course
NF54imputed <- fftimpute(NF54smooth, NF54Time)

#Again, we operated the Time row. Let's replace it.
NF54imputed["Time",] <- 2:57

plot(NF54imputed["Time",], NF54imputed["PF3D7_1343700",],
     xlab = "hpi", ylab = "log2 Expression", pch = 16, col = "red")
```



0.2.3 New Methodology

Honestly that was all far more elaborate than necessary. We can get very similar results using spline fit time courses, and the spline fit methodology allows for imputing sparser and irregularly sampled time courses. This is probably what I should have done in the first place. Using the original log2 transformed NF54 data, I can simply run the function `splineCourse` to do the work for me.

```
NF54spline <- splineCourse(NF54log2, NF54Time, fulltimes = 1:48)

plot(NF54spline["Time",], NF54spline["PF3D7_1343700",],
     xlab = "hpi", ylab = "log2 Expression", pch = 16, col = "red")
```



This method doesn't give us quite the smooth, beautiful imputation as the original method; however, it handles missing data and sparse time courses far, far more elegantly than my original method, and does not see the loess compression.

There are reasons to smooth the time course, especially the reference- the staging by transcription methodology described below works far better on smoothed time course data.

0.3 Time Course Analysis Methods

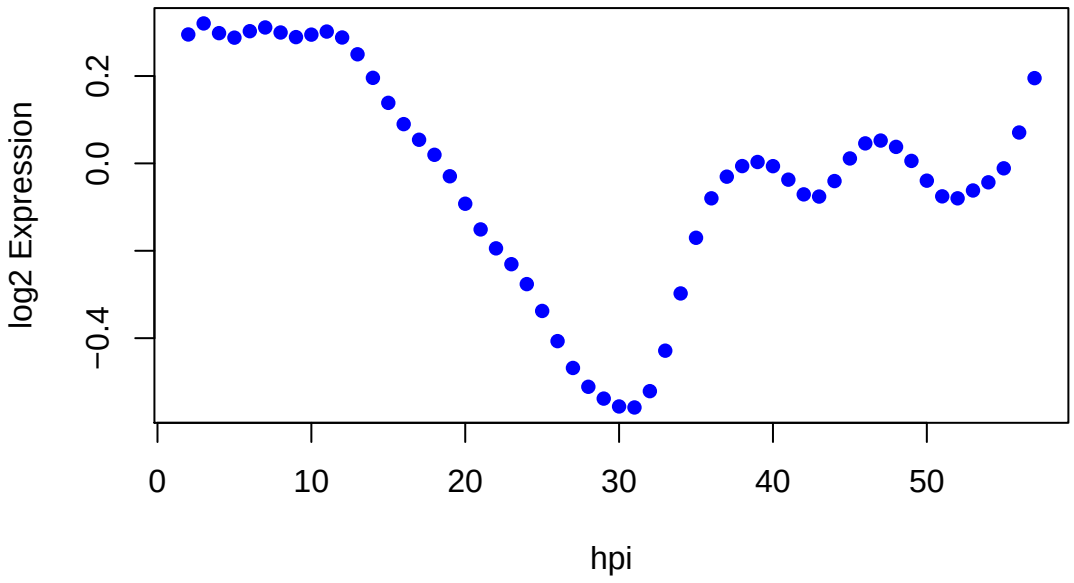
Comparing time courses is a complex chore. I have a few functions here to make that more palatable.

0.3.1 Staging by Transcription

The cyclical nature of expression in apicomplexan parasites provides an opportunity- we can use this to identify where in the cycle a parasite sample is by correlating the sample to each time point of a transcriptional time course, and identifying the reference time point at which the sample maximally correlates. Here, I use the previously imputed NF54spline time course as a reference, and introduce a new time course: NHP4026. This parasite time course was generated in an identical method to the NF54 reference time course.

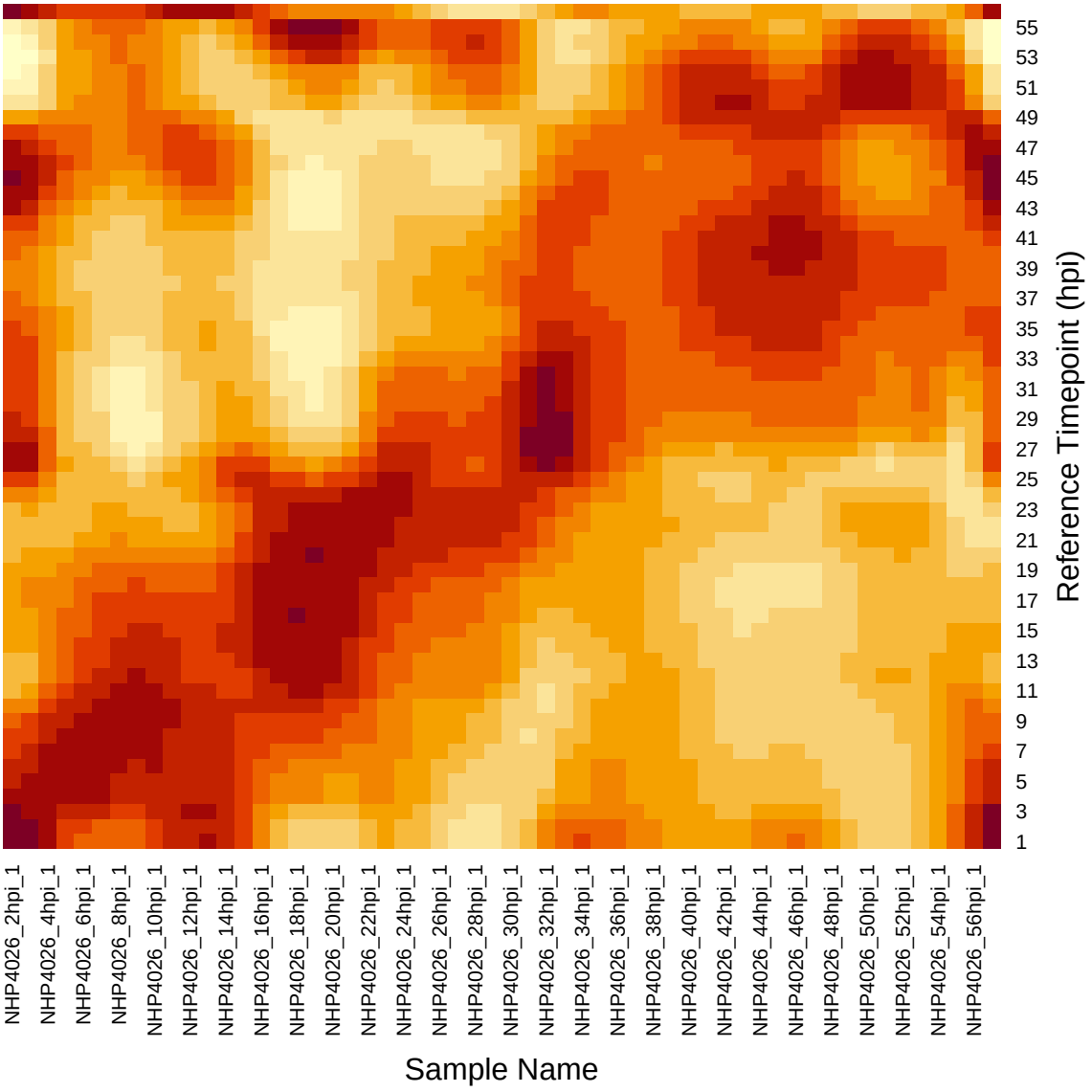
```
#First, let us use the methods demonstrated previously to curate the data

#Get log2 transformed data, using the NF54Avg data allowing for comparisons
NHP4026log2 <- LogTransform(newdata = NHP4026, refavg = NF54Avg)

#We had Time as a row; it was log transformed. Let's fix that.
NHP4026log2["Time",] <- NHP4026["Time",]

#kNN imputation of missing values
NHP4026noNAlog2 <- tcImpute(NHP4026log2)

#We need a Time matrix for the function; pull from NF54log2
```

```
NHP4026Time <- data.matrix(NHP4026noNAlog2["Time",rownames.force = T])

#Smooth data, with default
NHP4026smooth <- lSmooth(expdata = NHP4026noNAlog2, time = NHP4026Time)

#impute to hourly time course
NHP4026imputed <- fftimpute(NHP4026smooth, NHP4026Time)

#Again, we operated the Time row. Let's replace it.
NHP4026imputed["Time",] <- 2:57

plot(NHP4026imputed["Time",], NHP4026imputed["PF3D7_1343700",],
     xlab = "hpi", ylab = "log2 Expression", pch = 16, col = "blue")
```

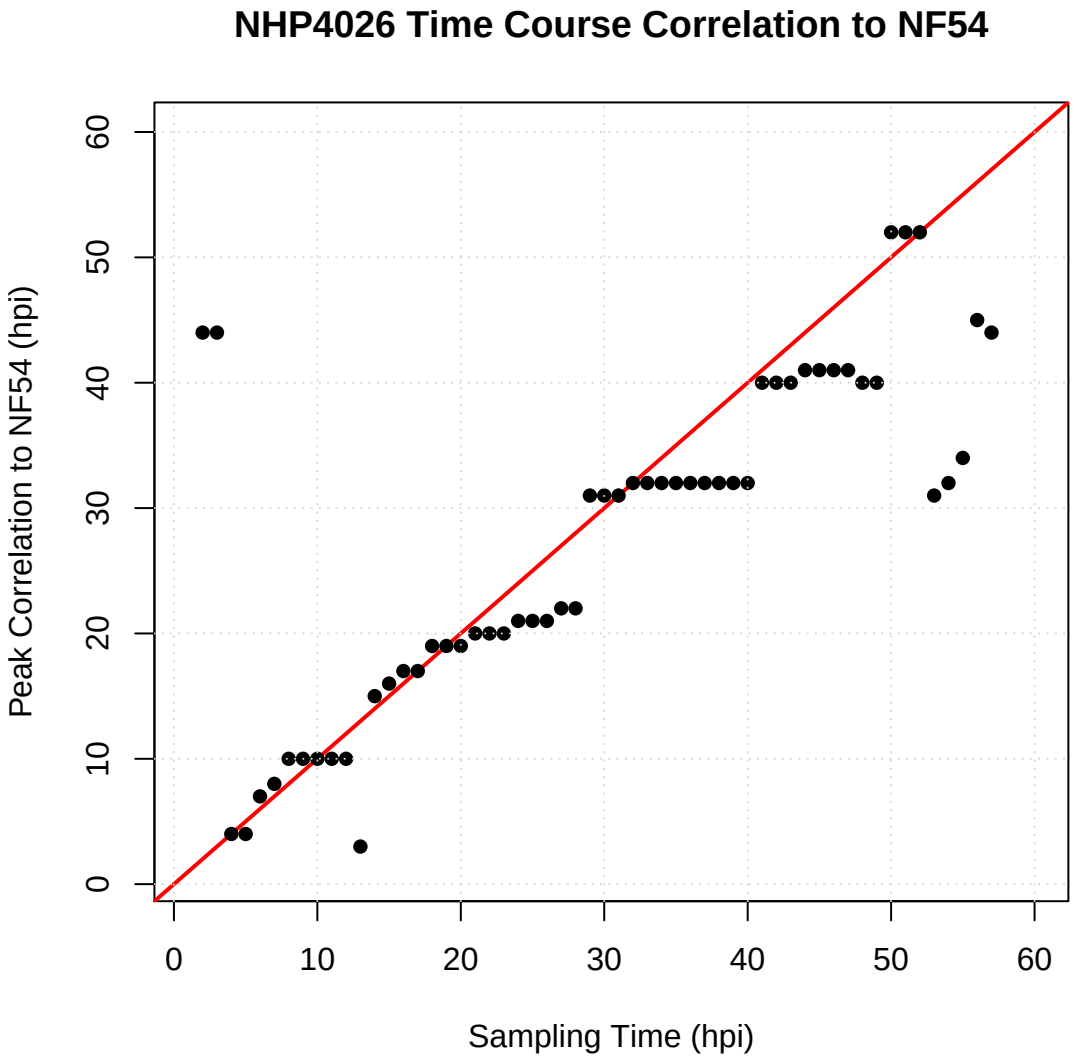


So, a short stretch of code gets us to an hourly time course over a single intraerythrocytic development cycle. Now we can stage every time point in the NHP4026 time course to compare progression of the two parasites, using the `stagingByTranscription` function.

```
#Stage all NHP4026 time points; generates a heat map by default
NHP4026staging <- stagingByTranscription(NHP4026imputed[!rownames(NHP4026imputed) %in% "Time",],
                                         NF54imputed[!rownames(NF54imputed) %in% "Time",])
```




```
#plot NHP4026 stages
plot(seq(1,60), seq(1,60), type = "n", main = "NHP4026 Time Course Correlation to NF54",
     xlab = "Sampling Time (hpi)", ylab = "Peak Correlation to NF54 (hpi)")
abline(a = 0, b = 1, col = "red", lwd = 2)
points(2:57, NHP4026staging[,1], pch = 16, cex = 1)
grid()
```



If the progression rates were identical, we would see all points on the diagonal; the deviations are extremely interesting, but we need to do further analysis to get to potential root causes.

0.3.2 Peak Staging

The apixcomplexan gene expression canon is a "just-in-time" model; briefly, each gene is not only cyclically expressed itself, but each gene's phase is in a static order between parasites, and that phase progression is steady throughout the intraerythrocytic development cycle. Observing peak expression time is a valuable tool to see if that is actually the case in our time courses.

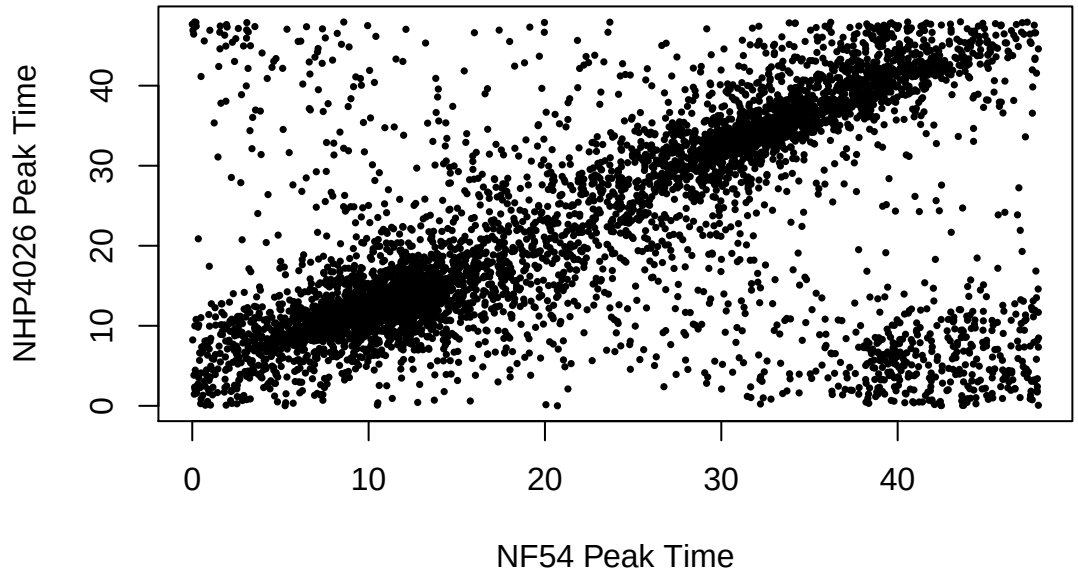
I have included several methods for this, with differing strengths and weaknesses. All of these methods are called from the `peakPhase` function. All of these functions fit a function to the expression profile of each gene and determine the peak expression time. The functions include:

- `sin`
  - This method fits a sin curve with a period of 48h. This is the broadest and simplest method.
- `fft`
  - This method pulls the pi/2 frequency from a Fourier Transformation of the data. It requires an

evenly sampled time course (easy when you impute)

- `spline`
  - This method fits a spline curve. Does not assume cyclicality.
- `poly`
  - This method fits a 6th order polynomial to the data. Does not assume cyclicality. As data is not normalized to have a constant slope across the IDC, often gives results at the outer bounds. Not terribly valuable for individual genes, but identifies very interesting trends.

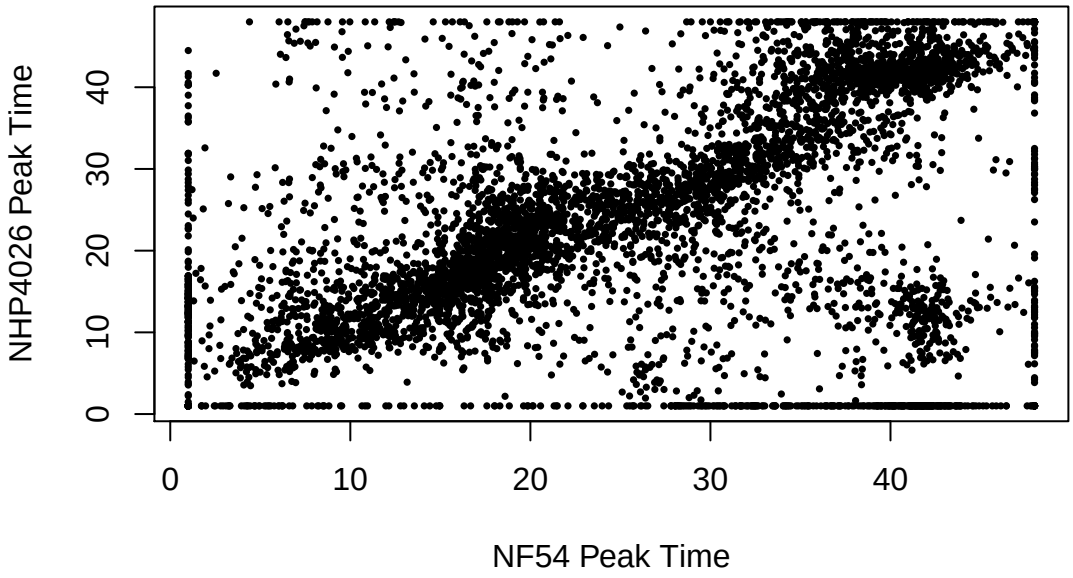
```
NF54alltimes <- data.matrix(NF54imputed["Time",])
NF54peak <- peakPhase(NF54imputed[!rownames(NF54imputed) %in% "Time",],
                          NF54alltimes, method = 'FFT')

NHP4026alltimes <- data.matrix(NHP4026imputed["Time",])
NHP4026peak <- peakPhase(NHP4026imputed[!rownames(NHP4026imputed) %in% "Time",],
                          NHP4026alltimes, method = 'FFT')

plot(NF54peak[,1], NHP4026peak[,1],
     xlab = "NF54 Peak Time", ylab = "NHP4026 Peak Time", pch = 16, cex = 0.5)
```



This method shows most genes peaking along the diagonal, implying similar expression timing. We can use other methods to identify interesting variation that may be obscured by the cyclical assumptions:

```
NF54peak <- peakPhase(NF54imputed[!rownames(NF54imputed) %in% "Time",],
                          NF54alltimes, method = 'poly')


NHP4026peak <- peakPhase(NHP4026imputed[!rownames(NHP4026imputed) %in% "Time",],
                          NHP4026alltimes, method = 'poly')

plot(NF54peak[,1], NHP4026peak[,1],
```

```
xlab = "NF54 Peak Time", ylab = "NHP4026 Peak Time", pch = 16, cex = 0.5)
```



Now we have biological differences we can chase!

0.4 Curation of Expression Datasets

GEO has a wealth of *P. falciparum* datasets- however, the changes in curation over the years often leave us with platforms using outdated names and alignments. The PFExpTools package allows for curation of GEO datasets in a few lines of code, a vast improvement over a grab bag of scripts as I've been using in the past. It has a variety of options- there are many different renaming strategies (blast, renaming from aliases, straight swap renaming) and the package handles all of them.

0.4.1 The easiest example

I'll repeat the Mok 2015 data curation here first. If the stars align and everything is perfect, you can run the entire thing with a single function call. All you need is a GEO accession number. Let's save some time, the accession number for the ex-vivo Mok 2015 expression data is GSE59097.

```
library(PFExpTools)

Mokdata <- fullCurate(GSE = "GSE59097")
Mokexp <- Mokdata[[1]]
Mokmeta <- Mokdata[[2]]
```

That's it- you now have a matrix of expression data, and you have a data frame of the metadata for the experiment (including parasite clearance half-life). This does take a bit of time to run- it's not terribly efficient, and I'll discuss why in a bit (and why I have no real desire to fix it).

Now, I am making a TON of assumptions here when I start this process. Let's have another look, showing the default calls.

```
Mokdata <- fullCurate(GSE = "GSE59097",
```

```
                        method = "blast",
                        transcripts = "current",
                        platcols = c(1,2),
                        aliases = NA,
                        match = 130,
                        secmatch = 60,
                        pullmeta = T,
                        pct = 0.8)
```

This call provides the same result as the first line earlier. The GSE parameter is obvious- it's the accession number. The `platcols` field here represents the columns in the platform data frame that represent the probe ID and the sequence, respectively. The default method is 'blast'- the function will actually pull the platform ID from the expression data set, download the platform, and pull the probe IDs and sequences. When `transcripts = "current"`, the function downloads the current PlasmoDB transcriptome file. Otherwise, this must be a filename for a transcriptome fasta file. The blast function will then build a blast database (provided you have blast installed and in your PATH), and blast all probes against the database. The `match` and `secmatch` variables represent perfect bitscore matches and an acceptable cutoff for secondary matches, respectively. The function will then keep only probes with a single perfect hit (`match`) and no second hits over the cutoff (`secmatch`). The `pullmeta` parameter allows you to pull the metadata or not. The `pct` parameter will remove any genes that do not have expression values in at least that percentage of samples.

So, if we go to the Mok 2015 data at the accession number above, we can navigate to the platform (GPL18893). We can look at the platform table- columns 1 and 2 contain the probe ID and the sequence, respectively, so we have what we need for the single line call.

0.4.2 What happens when the data doesn't cooperate?

Not all the data submitted to GEO is curated very well- sadly some of our own historical data is not laid out well enough to run without changing a few things. Let's use the Gonzales eQTL data as an example. The accession number is GSE12515. Going to the page, the platform is GPL7189. The platform table doesn't contain sequences, and the expression data has numeric IDs for probes, so `blast` won't work here. The platform also doesn't have series 3 gene names, so `swap` won't work either, leaving us with `alias`. For the `alias` method, the `platcols` parameter must be ProbeID and ORF. So, to curate this, we run the function as follows:

```
Gonzalesdata <- fullCurate(GSE = "GSE12515",
                           platcols = c(1,2),
                           method = "alias")
```

There are a few platforms that actually have series 3 gene names (what we consider "new" gene names in *P. falciparum*). If you're really lazy, you can just pull those and swap old ORF to new ORF from the platform file. The Mok 2015 data has that information- you can make that call as follows:

```
Mokdata <- fullCurate(GSE = "GSE59097",
                      method = "swap",
                      platcols = c(1,6))
```

There are two sub functions here that do all the heavy lifting, `nameChange` and `curateExpData`. The `nameChange` function takes the method and builds an appropriate oldname/newname file for reference. The `curateExpData` function takes the expression data and the name list and a few other variables and does the name swapping and gene consolidation. I won't go in to detail here on how that all works, the pdf reference is sufficient there.

Now, when you perform the entire curation in one shot, the functions end up processing the GSE file twice, once for the name change, once for the expression set curation. It's wildly inefficient that way, but I believe simplicity in function calls for curation and the relative rarity of curating data sets makes this inefficiency bearable.

0.5 Conclusions

The PFExpTools package is a tidy set of functions that encompasses a large part of the code I developed for my PhD thesis. This toolset is largely aimed at people reanalyzing publically available *P. falciparum* datasets and people processing expression time courses, two things I have spent a great deal of time using. I hope these tools prove useful to you, as I believe there is a great deal of excellent science yet to be done in this space!