

Behavpy - Circadian analysis

At the Gilestro lab we work on sleep and often need to do a lot of circadian analysis to compliment our sleep analysis. So we've got a dedicated suite of methods to analyse and plot circadian analysis. For the best run through of this please use the jupyter notebook tutorial that can be found at the end of this whole tutorial.

- [Circadian methods and plots](#)
- [Periodograms](#)

Circadian methods and plots

The below methods and plots should give a good insight into your specimens circadian rhythm. If you think another method should be added please don't hesitate to contact us and we'll see what we can do.

Head to the circadian notebook for an interactive run through of everything below.

Actograms

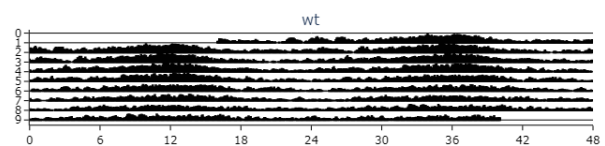
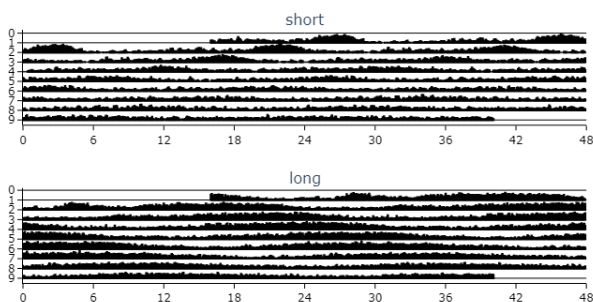
Actograms are one of the best initial methods to quickly visualise changes in the rhythm of a specimen over several days. An actogram double plots each days variable (usually movement) so you can compare each day with its previous and following day.

```
# .plot_actogram() can be used to plot individual specimens actograms or plot the average per group
# below we'll demonstrate a grouped example
```

```
fig = df.plot_actogram(
    mov_variable = 'moving',
    bin_window = 5, # the default is 5, but feel free to change it to smooth out the plot or vice versa
    facet_col = 'period_group',
    facet_arg = ['short', 'wt', 'long'],
    title = 'Grouped Actogram')

fig.show()
```

Grouped Actogram



```
# plot_actogram_tile will plot every specimen in your behavpy dataframe
# Be careful if your dataframe has lots of flies as you'll get a very crowded plot!

fig = df.plot_actogram_tile()
```

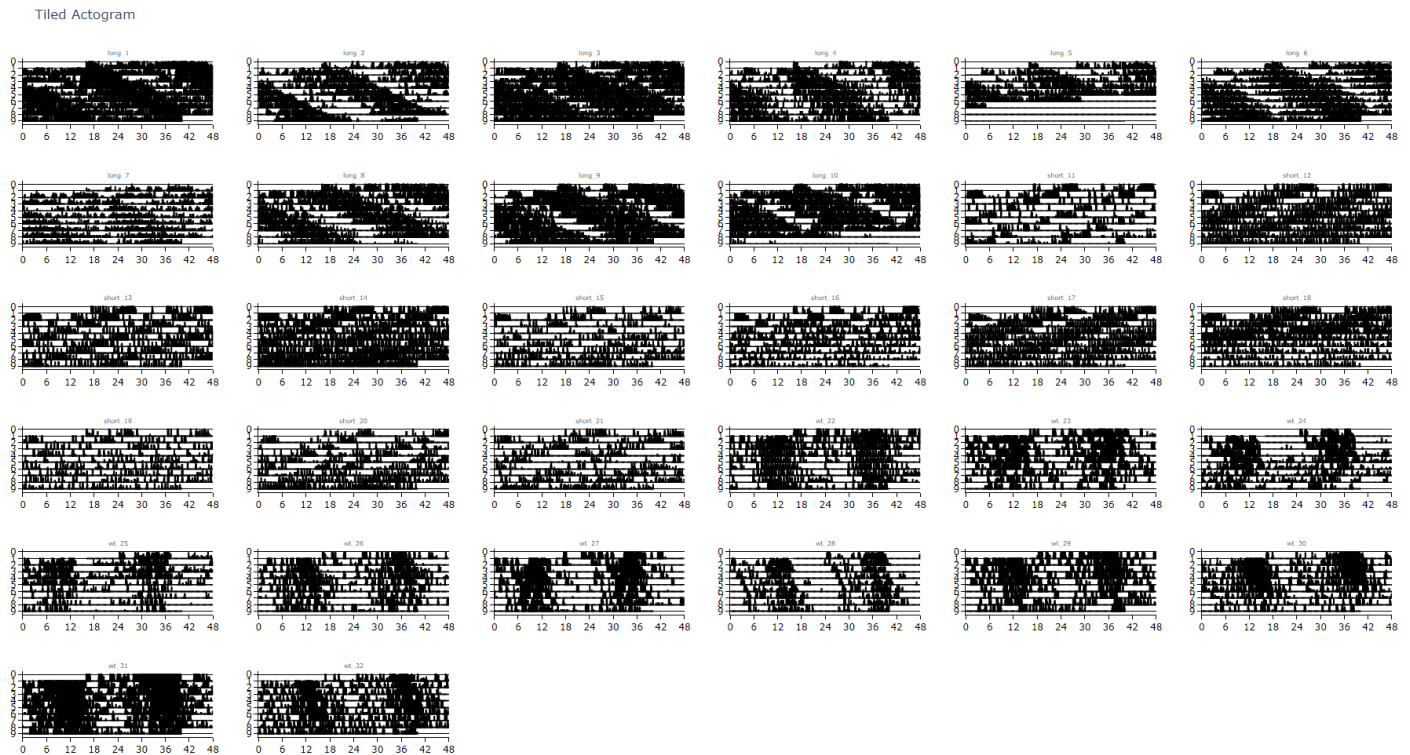
```
mov_variable = 'moving',
```

```
labels = None, # By default labels is None and will use the ID column for the labels.
```

```
## However if individual labels in the metadata add that column here. See the tutorial for an example
```

```
title = 'Tiled Actogram')
```

```
fig.show()
```



Anticipation Score

Many animals including *Drosophila* have peaks of activity in the morning and evening as lights turn on and off respectively. Given this daily activity the activity score looks to quantify when the specimens internal clock anticipates these moments. The score is calculated as the ratio of the final 3 hours of activity prior to lights on and off compared to the whole 6 hours prior.

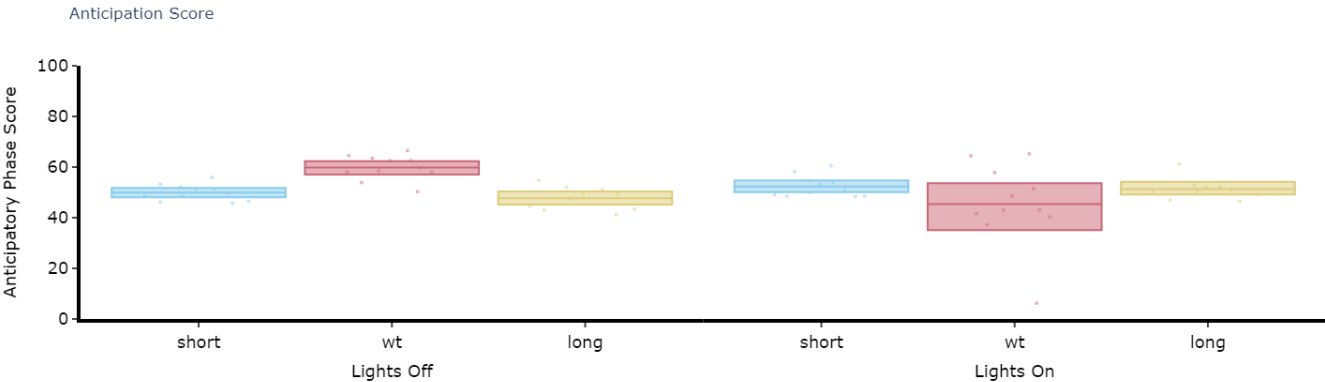
```
# Simply call the plot_anticipation_score() method to get a box plot of your results
```

```
# the day length and lights on/off time is 24 and 12 respectively, but these can be changed if you're augmenting the environment
```

```
fig, stats = df.plot_anticipation_score(  
    mov_variable = 'moving',  
    facet_col = 'period_group',  
    facet_arg = ['short', 'wt', 'long'],  
    day_length = 24,  
    lights_off = 12,
```

```
title = 'Anticipation Score')
```

```
fig.show()
```



Periodograms

Periodograms are essential for definitely showing periodicity in a quantifiable way. Periodograms often make use of algorithms created for spectral analysis, to decompose a signal into its component waves of varying frequencies. This has been adopted to behavioural data, in that it can find a base rhythm over several days from what is usually unclean data.

Ethoscopy has 5 types of periodograms built into its `behavpy_periodogram` class, which are 'chi squared' (the most commonly used), 'lomb scargle', 'fourier', and 'welch' (all based on the Fast Fourier Transformation (FFT) algorithm) and 'wavelet' (using FFT but maintaining the time dimension).

Try them all out on your data and see which works best for you.

Initialising the Periodogram class

To access the methods that calculate and plot periodograms you need to initialise your data frame not as the base `behavpy` but as the periodogram `behavpy` class: **`etho.behavpy_periodogram()`**

```
per_df = etho.behavpy_periodogram(data, metadata, check = True)

# If you've already initialised it as a behavpy object
per_df = etho.behavpy_periodogram(df, df.meta, check = True)
```

Calculating Periodograms

```
# The below method calculates the output from a periodogram function, returning a new dataframe with
information about each period in your given range
# and the power for each period
# the periodogram can only compute 4 types of periodograms currently, choose from this list ['chi_squared',
'lomb_scargle', 'fourier', 'welch']

per_chi = per_df.periodogram(
    mov_variable = 'moving',
    periodogram = 'chi_squared', # change the argument here to the other periodogram types
    period_range = [10,32], # the range in time (hours) we think the circadian frequency lies, [10,32] is the
default
    sampling_rate = 15, # the time in minutes you wish to smooth the data to. This method will interpolate
missing results
    alpha = 0.01 # the cutoff value for significance, i.e. 1% confidence
```

```
)
```

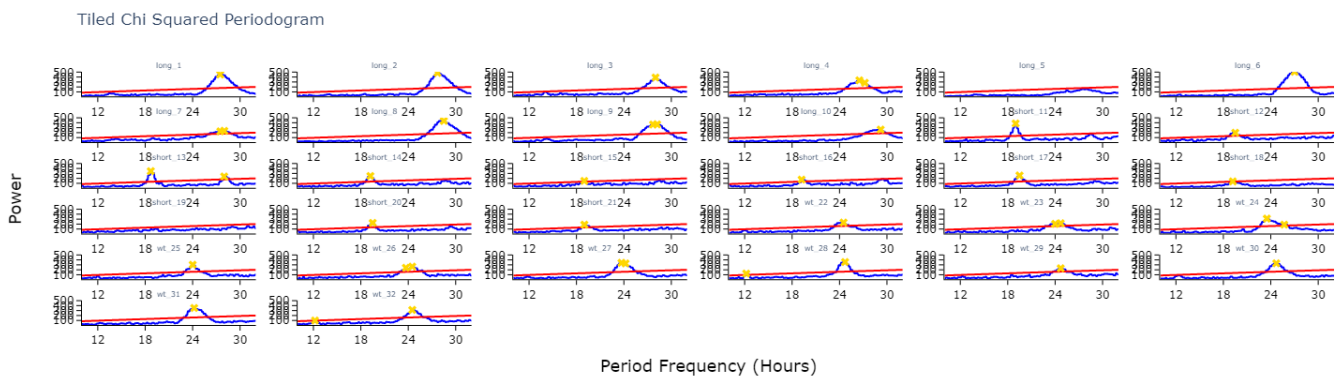
Hint! Sometimes increasing the sampling_rate can improve you're periodograms. On average we found 15 to work best with our data, but it can be different for everyone

Plotting Periodograms

Tile Plots

To get a good understanding of each individual specimen you can use the tiled periodogram plot

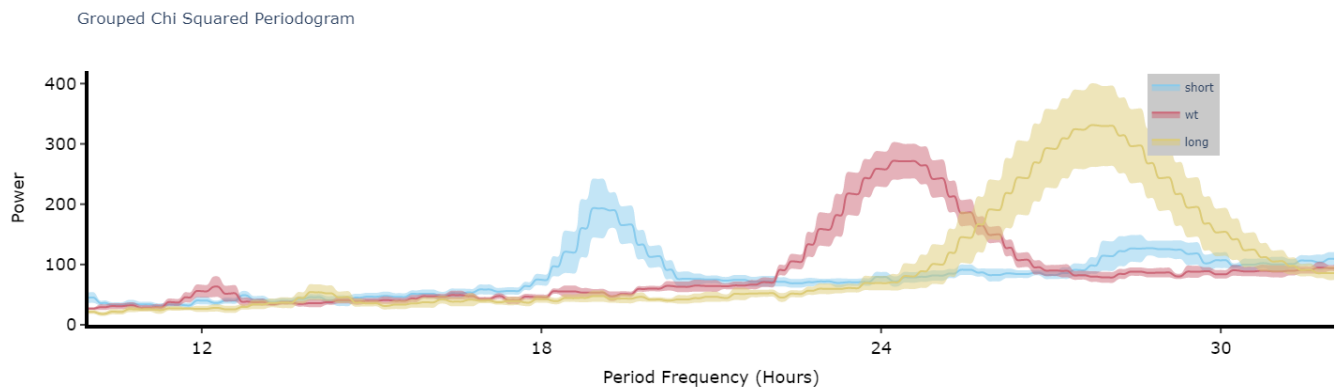
```
# Like the actogram_tile method we can rely on the ids of each specimen or we can use labels we created
fig = per_chi.plot_periodogram_tile(
    labels = 'tile_labels',
    find_peaks = True, # make find_peaks True to add a marker over significant peaks
    title = 'Tiled Chi Squared Periodogram'
)
fig.show()
```



Grouped plots

Plot the average of each specimens periodograms to get a better view of the difference between experimental groups.

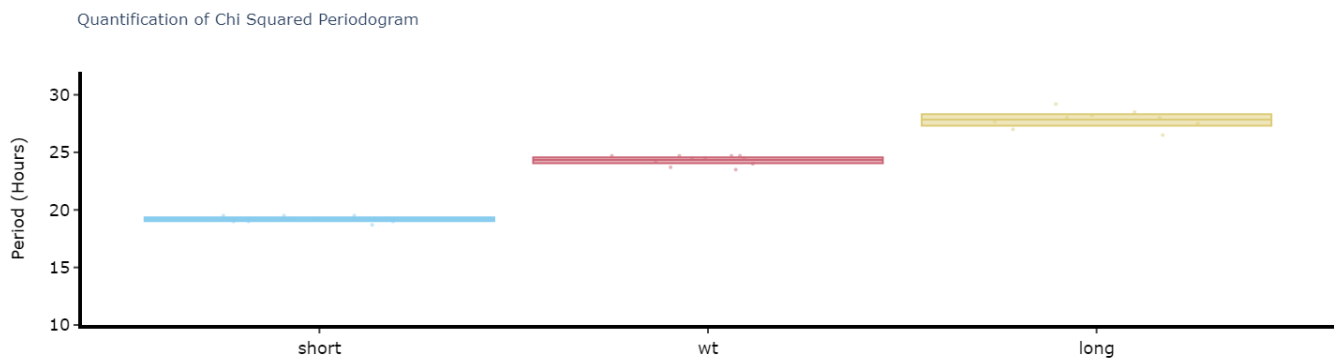
```
fig = per_chi.plot_periodogram(
    facet_col = 'period_group',
    facet_arg = ['short', 'wt', 'long'],
    title = 'Grouped Chi Squared Periodogram'
)
fig.show()
```



Quantifying plots

Like with the ethograms we can plot the above as quantification plot to see if what we see by eye is true statistically.

```
# Quantify the periodograms by finding the highest power peak and comparing it across specimens in a group
# call .quantify_periodogram() to get the results
fig, stats = per_chi.quantify_periodogram(
    facet_col = 'period_group',
    facet_arg = ['short', 'wt', 'long'],
    title = 'Quantification of Chi Squared Periodogram'
)
fig.show()
```



Wavelets

Wavelets are a special branch of periodograms as they are calculated in 3 dimensions rather than 2. Finding the period and power not for the whole given time, but rather broken down from the start to end. This then gives you an understanding of how rhythmicity changes over the course of your experiment, such as when you shift from LD to DD.

Wavelet plots are quite visually intensive, therefore the wavelet method will only compute and plot the average of all the data present in the data frame. Therefore, you will need to do some filtering beforehand to look at individual groups or an individual specimen.

```
# This method is powered by the python package pywt, Head to https://pywavelets.readthedocs.io/en/latest/ for information about the package and the other wavelet types
```

```
# filter your df prior to calling the plot
```

```
wt_df = per_df.xmv('period_group', 'wt')
```

```
fig = wt_df.wavelet(
```

```
    mov_variable = 'moving',
```

```
    sampling_rate = 15, # increasing the sampling rate will increase the resolution of lower frequencies (16-30 hours), but lose resolution of higher frequencies (2-8 hours)
```

```
    scale = 156, # the default is 156, increasing this number will increase the resolution of lower frequencies (and lower the high frequencies), but it will take longer to compute
```

```
    wavelet_type = 'morl', # the default wavelet type and the most commonly used in circadian analysis. Head to the website above for the other wavelet types or call the method .wavelet_types() to see their names
```

```
    title = 'Wavelet: WT'
```

```
)
```

```
fig.show()
```

