Per1 Gene Expression and Circadian Rhythm in the Brain

Program Imports

Import cell. This is run to import libraries used for various functions throughout the code.

In [239]:
```
import numpy as np
import pandas as pd
import scipy.signal
from scipy.signal import find_peaks

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

import bebi103

import plotly.graph_objects as go
from statsmodels.distributions.empirical_distribution import ECDF

import bokeh
import bokeh.plotting as bk
from bokeh.models.annotations import Title
from bokeh.io import show, output_file, export_png
from bokeh.plotting import figure
from bokeh.layouts import row
bokeh.io.output_notebook()

notebook_url = 'localhost:8888'

%load_ext watermark
%watermark -v -p numpy,scipy,pandas,bokeh,bebi103,sklearn,jupyterlab,plotly
```

BokehJS 1.2.0 successfully loaded.

```
The watermark extension is already loaded. To reload it, use:
  %reload_ext watermark
CPython 3.7.3
IPython 7.6.1

numpy 1.16.4
scipy 1.2.1
pandas 0.24.2
bokeh 1.2.0
bebi103 0.0.43
sklearn 0.21.2
jupyterlab 1.1.1
plotly 4.2.1
```

Formatting Data

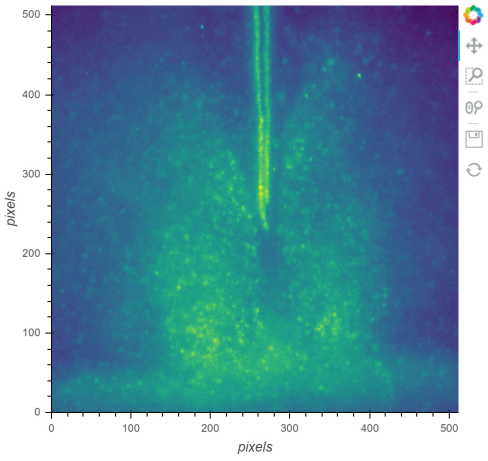
The dataset itself is loaded and formatted in this block. The data from the CSV is formatted into an array showing the image number, frame number, and time. The images are loaded by numpy. After this block of code, we can now move on to actually analyzing the data.

In [240]:
```
#Loads in data + formats for data processing
data = np.loadtxt('datapts.csv', delimiter=',', skiprows=1)
image_no=data[:,0].reshape(-1,1)
frame_no=data[:,1].reshape(-1,1)
time_hrs=data[:,2].reshape(-1,1)
nb_images = data.shape[0]
flo_image_1 = np.load('flo_image_1.npz')
flo_image_2 = np.load('flo_image_2.npz')
image_ids = np.concatenate([flo_image_1['image_ids'], flo_image_2['image_ids']])
images = np.concatenate([flo_image_1['image_stack'], flo_image_2['image_stack']])
```

First Image

This block of code is simply meant to show the image before any graphs are shown. It provides the reader a visualization of the images being analyzed and where the bright regions of fluorescence are occurring.

In [242]:
```
# show primary image
image = bebi103.image.imshow(images[0])
bokeh.io.show(image)
```

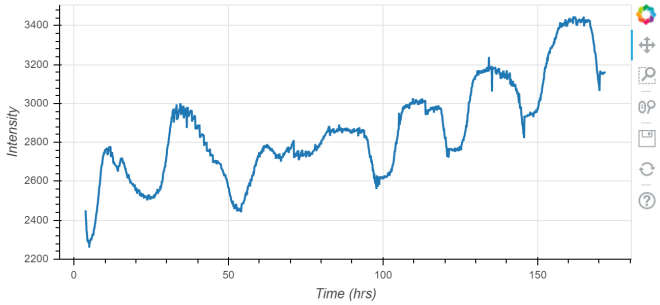


Quantifying Image Intensities Over Time

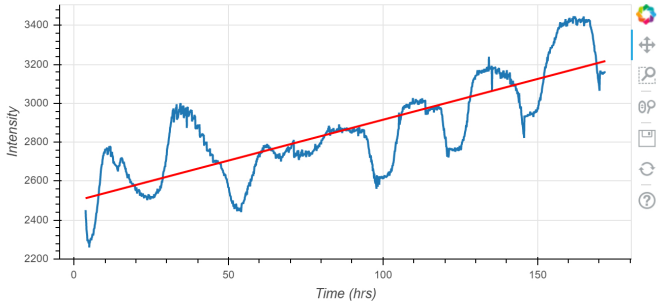
We then found the intensities of the images over time and graphed them using Bokeh. The graph shows the intensity of the images over time and these intensities rise over the course of the two weeks for unknown reasons. Because of these rising intensities it is challenging to identify the peaks along this continuous function. To account for this issue, we created a linear regression of all of the points and then subtracted the value of that line from each point, creating a graph where that line of best fit becomes 0 on the y-axis. Using this normalized graph, we were able to find the peaks along each period.

With these peaks identified, we plotted the difference in peak separation over time. These points were fit with a linear model. Taking the average of this line, we found the average difference in time between each peak to be 25.05 hours. This time is close to 24 hours, but this is the least rigorous test we have decided to use.

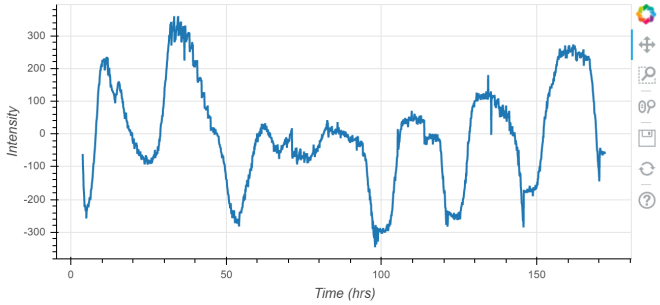
In [243]:
```
#intensity values of the image per time

mean_values = np.mean(images, axis=(1,2))
time_hrs1 = time_hrs.flatten()

p = bokeh.plotting.figure(plot_width=650,
                          plot_height=300,
                          x_axis_label='Time (hrs)',
                          y_axis_label='Intensity')

p.line(time_hrs1, mean_values, line_width=2, line_join='bevel')

bokeh.io.show(p)
```



In [244]:
```
#add a linear regression

model = LinearRegression()
model.fit(time_hrs, mean_values)

p.line(time_hrs1, model.predict(time_hrs),color='red',line_width=2)

bokeh.io.show(p)
```

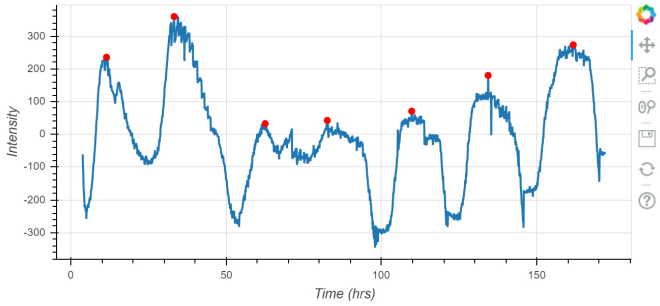


In [245]:
```
#subtract the linear regression to normalize the peaks. able to find peaks now

mean_linear = model.predict(time_hrs)
norm_values = mean_values - mean_linear

q = bokeh.plotting.figure(plot_width=650,
                          plot_height=300,
                          x_axis_label='Time (hrs)',
                          y_axis_label='Intensity')

q.line(time_hrs1, norm_values, line_width=2, line_join='bevel')

bokeh.io.show(q)
```

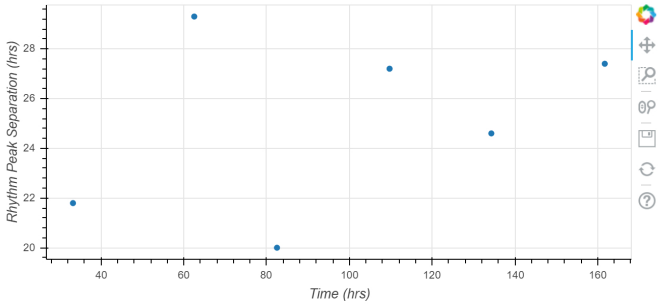


In [246]:
```
#giving a distance of 15 hrs between peaks. there are 10 frames per hour
#now that it is flat, able to find peaks with the find_peaks function

peak_ind = find_peaks(norm_values, height=0, distance=150)
p_index = peak_ind[0]
dfp = pd.DataFrame({'Time':time_hrs1[p_index],'Intensity':norm_values[p_index]})

q.circle(dfp['Time'],dfp['Intensity'],size=6,color='red')
```

```
bokeh.io.show(q)
```

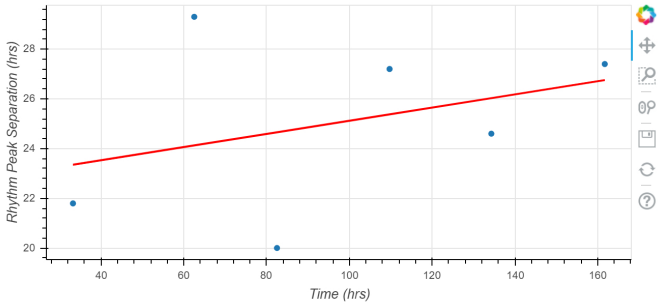


In [247]:
```
#plotting all of the peak differences as dots over time

dfp["circpeaks"] = dfp["Time"].diff()

s = bokeh.plotting.figure(plot_width=650,
                          plot_height=300,
                          x_axis_label='Time (hrs)',
                          y_axis_label='Rhythm Peak Separation (hrs)')

s.circle(dfp['Time'], dfp["circpeaks"], size=5)

bokeh.io.show(s)
```



In [248]:
```
#finding the peak average and making a linear regression through the points

print("The average difference in peak intensity is", dfp["circpeaks"].mean(), "hrs")

noNANt = dfp["Time"].drop([0])
noNANc = dfp["circpeaks"].dropna()

df2dta = np.reshape(noNANt.to_numpy(), (-1,1))
df2dca = np.reshape(noNANc.to_numpy(), (-1,1))

df2dtf = df2dta.flatten()
df2dcf = df2dca.flatten()

model.fit(df2dta, df2dca)

s.line(df2dtf, model.predict(df2dta).flatten(),color='red',line_width=2)

bokeh.io.show(s)
```

```
The average difference in peak intensity is 25.048946303333334 hrs
```

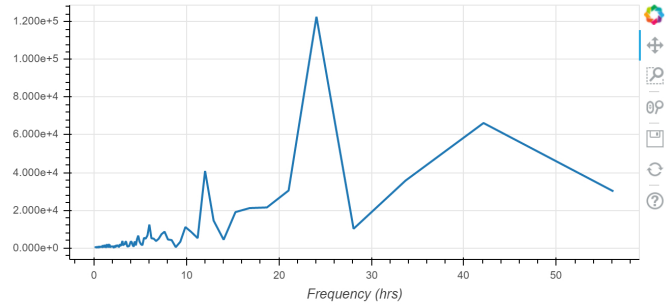


The Fast Fourier Transform

To more rigorously find the difference between peaks, we decided to apply the Fourier transform to find the underlying frequencies of the waveform given by the first intensity versus time graph. Taking the Fourier transform was simple to code because within Numpy there is a operation called the Fast Fourier Transform (FFT). This is a version of the Fourier transform that reduces the computational complexity of the transform by taking advantage of the symmetries that exist within the integral. Because the Fourier transform is already a complex calculation for very basic waveforms when a random waveform like the one generated by the intensity is input, it takes a lot of computational power, so the FFT allows the computer to run the code much faster. By changing to frequency on the x-axis, we can see that the most common frequency is around the 24-hour mark as expected.

In [263]:
```
#doing a fourier transform statistical analysis to show that the dominant value is closer to 24.5 hrs according to the data

spectrum = np.fft.rfft(norm_values)
freq = np.fft.rfftfreq(len(norm_values), 1/10)  #because there are ten image frames per hour

t = bokeh.plotting.figure(plot_width=650,
                          plot_height=300,
                          x_axis_label='Frequency (hrs)', )

t.line(1.0 / freq[3:], abs(spectrum[3:len(freq)]), line_width=2, line_join='bevel')
```

```
bokeh.io.show(t)
```



Specific ROIs

To further this project, we decided to study the differences in circadian oscillation of different parts of the image. To do this, we used Bebi103 to draw ROIs in the right SCN, left SCN, and the third ventricle. For ease of grading, we converted the ROIs to CSV format to submit along with the pdf. The ROIs were then labeled into ROIs 1, 2, and 3.

In [250]:
```
#now, to study the certain areas of the scan, draw the 3 ROIS with bebi103

roisdraw = bebi103.image.draw_rois(images[0], flip=False)
```

In [252]:
```
#convert ROIS to csv for submission

df_rois = bebi103.image.roicds_to_df(roisdraw)
df_rois.to_csv('ROISFinalnGrid.csv', index=False)
```

In [253]:
```
#read the file + make the roi csv ready to apply to an image

roisdf = pd.read_csv('ROISFinalnGrid.csv', header=[0])
rois = [bebi103.image.verts_to_roi(g[['x', 'y']].values, *images[0].shape) for _, g in roisdf.groupby('roi')]
```

Creating the graphing functions

Now, we begin to format the data, setting bounds to ensure all points are included within each ROI and able to be parsed by the code. We will then create functions to make the analytical graphs.

In [254]:
```
#set the bounds for image analysis for loop

global bounds0,bounds1,bounds2

rois0 = roisdf[roisdf["roi"]==0]
rois1 = roisdf[roisdf["roi"]==1]
rois2 = roisdf[roisdf["roi"]==2]

xboundmin0 = int(rois0['x'].min())
xboundmax0 = int(rois0['x'].max())
yboundmin0 = int(rois0['y'].min())
yboundmax0 = int(rois0['y'].max())

bounds0 = [xboundmin0,xboundmax0,yboundmin0,yboundmax0]

xboundmin1 = int(rois1['x'].min())
xboundmax1 = int(rois1['x'].max())
yboundmin1 = int(rois1['y'].min())
yboundmax1 = int(rois1['y'].max())

bounds1 = [xboundmin1,xboundmax1,yboundmin1,yboundmax1]

xboundmin2 = int(rois2['x'].min())
xboundmax2 = int(rois2['x'].max())
yboundmin2 = int(rois2['y'].min())
yboundmax2 = int(rois2['y'].max())

bounds2 = [xboundmin2,xboundmax2,yboundmin2,yboundmax2]
```

Showing the Image

Before creating the graphs analyzing the images, we decided to make plots of the images to be shown alongside its data. That way, it will be easy for the reader to associate each image and ROI with the data.

In [255]:
```
#creates a graph of the specific ROI from the drawings

def showimage(roinum):
    global i0,i1,i2

    roi, roi_bbox, roi_box = rois[roinum]

    im = images[0][roi_bbox]
    q = bebi103.image.imshow(im)

    t = Title()
    t.text = 'Image of ROI{}'.format(roinum+1)
    q.title = t

    if (roinum == 0):
        i0 = q
    if (roinum == 1):
        i1 = q
    if (roinum == 2):
        i2 = q

    bokeh.io.show(q)
```

Creating the Intensity Graph

This cell of code is by far the most intensive. This function features a triple-nested for loop made to parse through the 3D array of images which stand for the

following: image number, rows of pixels, and columns of pixels. This parses through all the ROIs for the intensity values, means them out and plots them on a normalized intensity graph. The mean and median rhythms are then also added below these graphs for convenience and extra numerical data.

In [256]:
```
#create the intensity over time chart. this is done by using the bounds of the ROI and parsing through the images intensity data
with a triple nested for loop. (image number,pixel rows,pixel columns)

def showintchart(roinum):
    global s0,s1,s2
    global d0,d1,d2
    global bounds0,bounds1,bounds2
    global cvmean,cvmedian

    roi, roi_bbox, roi_box = rois[roinum]

    im = images[0][roi_bbox]

    total_int = np.zeros(int(images.size/images[0][0].size/images[0][0].size)) #two images[0][0] because image is a square

    if (roinum == 0):
        bounds = bounds0
    if (roinum == 1):
        bounds = bounds1
    if (roinum == 2):
        bounds = bounds2

    intvalues = np.zeros((bounds[3]-bounds[2])*(bounds[1]-bounds[0])+1)

    #parsing through the intensity values for every image in a nested for loop
    for i in range(int(images.size/images[0][0].size/images[0][0].size)): #two images[0][0] because image is a square
        counter = 0
        for r in range(bounds[2],bounds[3]):
            for c in range(bounds[0],bounds[1]):
                counter+=1
                intvalues[counter] = images[i][r][c]
        total_int[i] = intvalues.mean()

    time_hrs1 = time_hrs.flatten()

    #normalizing values like before
    model.fit(time_hrs, total_int)
    mean_linear = model.predict(time_hrs)
    norm_values = total_int - mean_linear
    peak_ind = find_peaks(total_int,distance=100,width=10)

    v = bokeh.plotting.figure(plot_width=650,plot_height=300,x_axis_label='Time (hrs)', y_axis_label='Intensity (Normalized)',ti
tle='Normalized Intensity per Hour for ROI{}'.format(roinum+1))

    v.line(time_hrs1, norm_values, line_width=2, line_join='bevel')

    p_index = peak_ind[0]

    #creating peak difference data and peak data for input into image
    pts = pd.DataFrame({'Time':time_hrs1[p_index],'Intensity':norm_values[p_index]})
    pts["circpeaks"] = pts["Time"].diff()
    cvmean[roinum] = pts["circpeaks"].mean()
    cvmedian[roinum] = pts["circpeaks"].median()

    v.circle(pts['Time'],pts['Intensity'],size=3,color='black',alpha=.7)

    bokeh.io.show(v)

    #storing data for later
    if (roinum == 0):
        s0 = v
        d0 = pts["circpeaks"]
        d0 = d0.to_numpy()[~np.isnan(d0)]
    if (roinum == 1):
        s1 = v
        d1 = pts["circpeaks"]
        d1 = d1.to_numpy()[~np.isnan(d1)]
    if (roinum == 2):
        s2 = v
        d2 = pts["circpeaks"]
        d2 = d2.to_numpy()[~np.isnan(d2)]

    print("The mean rhythm for ROI #{} is {}".format(roinum+1,cvmean[roinum]))
    print("The median rhythm for ROI #{} is {}".format(roinum+1,cvmedian[roinum]))

    fouriertransforms(roinum, norm_values)
```

ROI Fourier Transforms

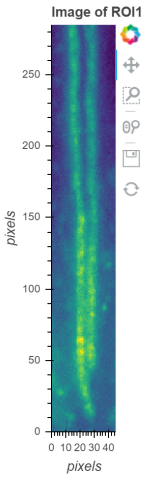
This portion goes through the same coding for the FFT from before, applying it to the three ROIs and saving all of this data to be called upon in the next step.

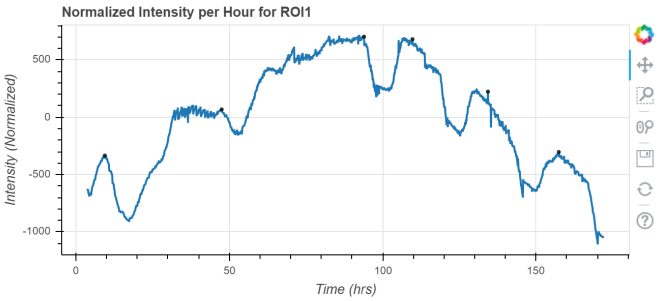
In [264]:
```
#creation of fourier transforms. need it called from the showintchart function because we need the norm_values to create the gra
ph

def fouriertransforms(roinum, norm_values):
    global f0,f1,f2

    spectrum = np.fft.rfft(norm_values)
    freq = np.fft.rfftfreq(len(norm_values), 1/10)  #because there are ten image frames per hour

    w = bokeh.plotting.figure(plot_width=650,plot_height=300,x_axis_label='Frequency (hrs)',title='Fourier Transform for ROI{}'.
format(roinum+1))

    w.line(1.0 / freq[3:], abs(spectrum[3:len(freq)]), line_width=2, line_join='bevel')

    if (roinum == 0):
        f0 = w
    if (roinum == 1):
        f1 = w
    if (roinum == 2):
        f2 = w
```

Showing the Results of the ROIs

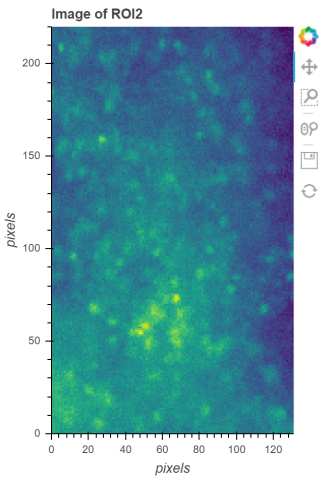
This block of code begins the calling of the functions created previously. It is parsing through every ROI using a simple for loop and calls all the functions which show the data using the ROI number as the parameter. To help with easy dissemination of this data, the next cell of code puts all of these graphs together in a smaller, more manageable form.

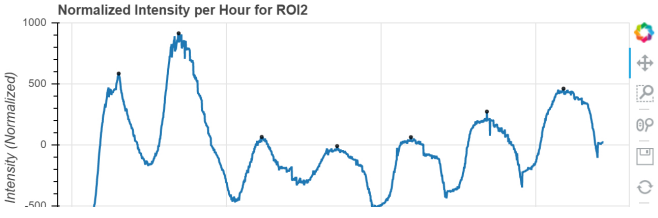
In [265]:
```
#where the for loop and parsing begins for the ROI segment of the project

global cvmean,cvmedian
global s0,s1,s2
global d0,d1,d2
global f0,f1,f2
global i0,i1,i2

cvmean = np.zeros(3)
cvmedian = np.zeros(3)
plots = np.empty(3)

for roinum in range(3):
    showimage(roinum)
    showintchart(roinum) #Fourier Transforms called at the end of this function

#this data is created in the for loop, but used later in the following ECDF plot. it is put in this cell so we don't have to cal
l the global d variables again
peakdiffs = np.concatenate([d0,d1,d2])
roisnums =  np.concatenate([np.zeros_like(d0),np.ones_like(d1),np.full_like(d2,2)])

datapeaks = pd.DataFrame({'ROI': roisnums,'Peak Difference': peakdiffs})
```

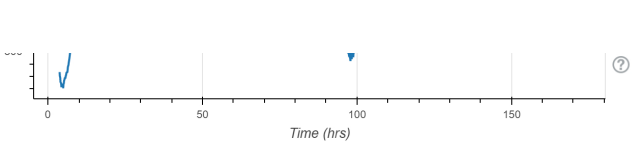




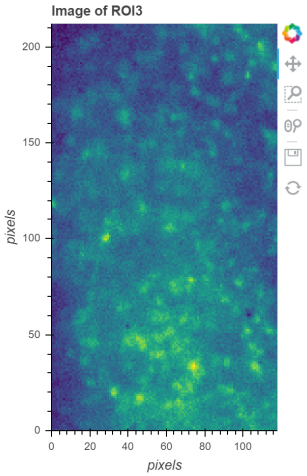

```
The mean rhythm for ROI #1 is 29.5987355644
The median rhythm for ROI #1 is 24.599999999999994
```

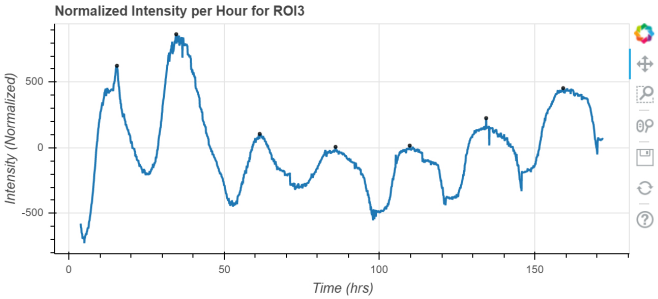






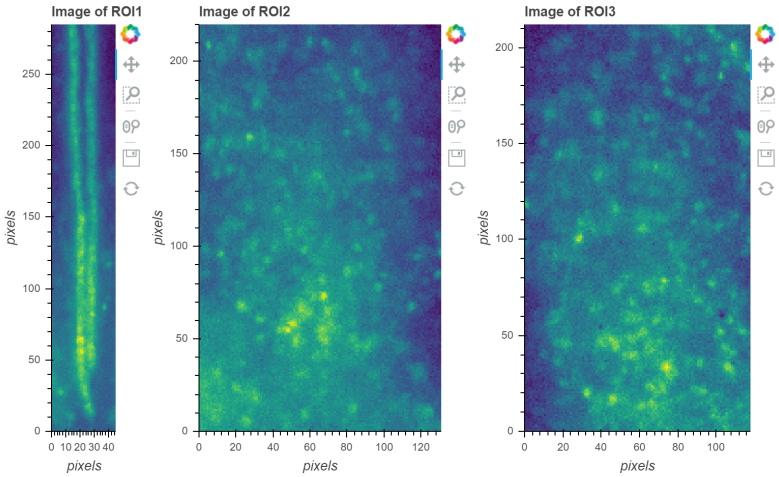

```
The mean rhythm for ROI #2 is 23.998946303333337
The median rhythm for ROI #2 is 24.5
```

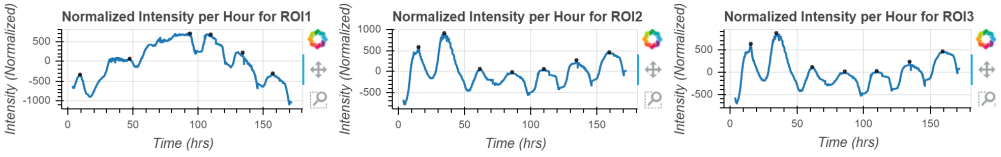


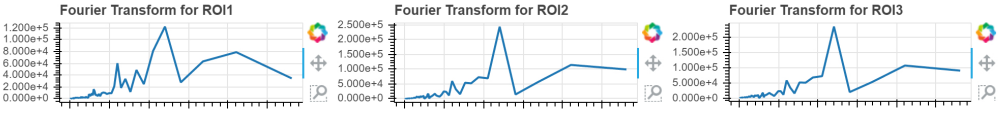


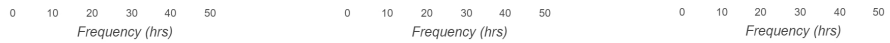

```
The mean rhythm for ROI #3 is 23.948946303333333
The median rhythm for ROI #3 is 24.5
```

In [266]:
```
#shows the previous data in a compact and easy form

x = row([i0, i1, i2])
bokeh.io.show(x)
y = row([s0, s1, s2], sizing_mode='scale_width')
bokeh.io.show(y)
z = row([f0, f1, f2], sizing_mode='scale_width')
bokeh.io.show(z)
```









ECDF Plot

To create the ECDF Catplot we first created the function that sorts and manages the data. We then created the figure, assigning titles to the graph and each of the axes. Then, for each ROI we assigned a specific color, name and data to show up on the plot. We then coded the plot to show the amount of the different images that had a certain amount of time between the peaks. This graph showed that the two sides of the SCN had almost the exact same proportional distribution, having the same number of hours between peaks, centered around 24 hours in a close cluster. It also showed that the third ventricle (red) was very different from the SCN, and it lacked the same centering around the 24-hour mark that the two sides of the SCN showed. This graph is probably the best depiction of the differences between the SCN and third ventricle's oscillations as although they had similar means, the SCN data is much stronger than the third ventricle's.

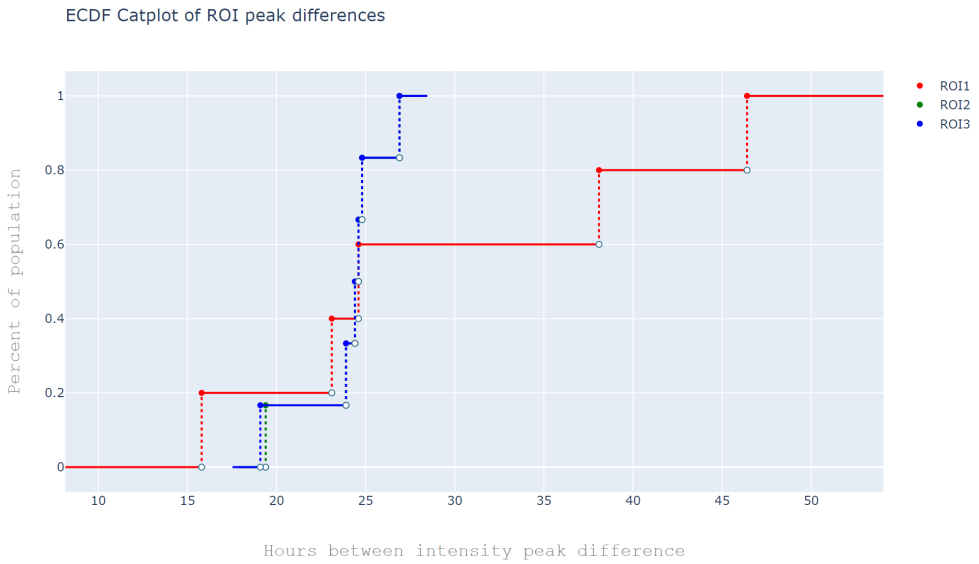
In [260]:
```
#ECDF graph creation

def ecdf(x):
    x = np.sort(x)
    def result(v):
        return np.searchsorted(x, v, side='right') / x.size
    return result
```

In [261]:
```
#creation of the figure for lines to be drawn with title and axes

layout = go.Layout(
    title=go.layout.Title(
        text='ECDF Catplot of ROI peak differences',
        xref='paper',
        x=0
    ),
    xaxis=go.layout.XAxis(
        title=go.layout.xaxis.Title(
```

```
            text='Hours between intensity peak difference',
            font=dict(
                family='Courier New, monospace',
                size=18,
                color='#7f7f7f'
            )
        )
    ),
    yaxis=go.layout.YAxis(
        title=go.layout.yaxis.Title(
            text='Percent of population',
            font=dict(
                family='Courier New, monospace',
                size=18,
                color='#7f7f7f'
            )
        )
    )
)

fig = go.Figure(layout=layout)
```

In [ ]:
```
#values of the peak differences in all the rois, now separated by ROI

roi0 = datapeaks[datapeaks['ROI'] == 0]['Peak Difference']
roi1 = datapeaks[datapeaks['ROI'] == 1]['Peak Difference']
roi2 = datapeaks[datapeaks['ROI'] == 2]['Peak Difference']
```

In [262]:
```
#adding all the lines and creating the ECDF graph

color = ''
data = roi0

#parsing through each ROI and setting values specific to each one
for roinum in range(3):
    if (roinum == 0):
        data = roi0.to_numpy().flatten()
        color = 'red'
        name = 'ROI1'
    if (roinum == 1):
        data = roi1.to_numpy().flatten()
        color = 'green'
        name = 'ROI2'
    if (roinum == 2):
        data = roi2.to_numpy().flatten()
        color = 'blue'
        name = 'ROI3'

    #data mamipulation so it will fit the ECDF formatting
    xs = np.unique(data)
    steps=ecdf(data)(np.unique(data))

    xs1 = np.column_stack([np.insert(xs,0,xs[0]-np.diff(xs).mean()),
                           np.append(xs, xs[-1]+np.diff(xs).mean())])
    steps1 = np.concatenate([[[0,0]], np.column_stack([steps,steps])])

    xs2 = np.column_stack([xs,xs])
    steps2 = np.column_stack([np.insert(steps, 0, 0)[:-1], steps])



    #parsing through data and adding the lines, creating the graph in segments
    for i in range(len(steps)+1):
        fig.add_scatter(x=xs1[i],y=steps1[i], mode='lines', line_color=color, showlegend=False)
    for i in range(len(steps)):
        fig.add_scatter(x=xs2[i],y=steps2[i], mode='lines', line_color=color, line_dash='dot', showlegend=False)
    fig.add_scatter(x=xs, y=steps, mode='markers', marker_color=color, name=name)
    fig.add_scatter(x=xs, y=np.pad(steps,1,mode='constant')[:-2], mode='markers', marker_color='white', marker = dict(line_color
='#367588', line_width=1), showlegend=False)

fig.show(renderer='svg', height=600, width=1000)
```




```
tornado.application - ERROR - Uncaught exception GET /autoload.js?bokeh-autoload-element=375286&bokeh-absolute-url=http://local
host:52660&resources=none (::1)
HTTPServerRequest(protocol='http', host='localhost:52660', method='GET', uri='/autoload.js?bokeh-autoload-element=375286&bokeh-
absolute-url=http://localhost:52660&resources=none', version='HTTP/1.1', remote_ip='::1')
Traceback (most recent call last):
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\tornado\web.py", line 1699, in _execute
    result = await result
```

```
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\tornado\gen.py", line 742, in run
    yielded = self.gen.throw(*exc_info)  # type: ignore
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\server\views\autoload_js_handler.py", line 60, in get
    session = yield self.get_session()
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\tornado\gen.py", line 735, in run
    value = future.result()
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\tornado\gen.py", line 742, in run
    yielded = self.gen.throw(*exc_info)  # type: ignore
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\server\views\session_handler.py", line 77, in get_session
    session = yield self.application_context.create_session_if_needed(session_id, self.request)
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\tornado\gen.py", line 735, in run
    value = future.result()
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\tornado\gen.py", line 748, in run
    yielded = self.gen.send(value)
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\server\contexts.py", line 215, in create_session_if_needed
    self._application.initialize_document(doc)
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\application\application.py", line 178, in initialize_document
    h.modify_document(doc)
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\application\handlers\function.py", line 133, in modify_document
    self._func(doc)
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bebi103\image.py", line 580, in modify_doc
    doc.add_root(bokeh.layouts.column(p, table))
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\document\document.py", line 304, in add_root
    self._pop_all_models_freeze()
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\document\document.py", line 1019, in _pop_all_models_freeze
    self._recompute_all_models()
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\document\document.py", line 1042, in _recompute_all_models
    a._attach_document(self)
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\model.py", line 714, in _attach_document
    raise RuntimeError("Models must be owned by only a single document, %r is already in a doc" % (self))
RuntimeError: Models must be owned by only a single document, ColumnDataSource(id='375285', ...) is already in a doc
tornado.application - ERROR - Uncaught exception GET /autoload.js?bokeh-autoload-element=375286&bokeh-absolute-url=http://local
host:52660&resources=none (::1)
HTTPServerRequest(protocol='http', host='localhost:52660', method='GET', uri='/autoload.js?bokeh-autoload-element=375286&bokeh-
absolute-url=http://localhost:52660&resources=none', version='HTTP/1.1', remote_ip='::1')
Traceback (most recent call last):
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\tornado\web.py", line 1699, in _execute
    result = await result
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\tornado\gen.py", line 742, in run
    yielded = self.gen.throw(*exc_info)  # type: ignore
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\server\views\autoload_js_handler.py", line 60, in get
    session = yield self.get_session()
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\tornado\gen.py", line 735, in run
    value = future.result()
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\tornado\gen.py", line 742, in run
    yielded = self.gen.throw(*exc_info)  # type: ignore
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\server\views\session_handler.py", line 77, in get_session
    session = yield self.application_context.create_session_if_needed(session_id, self.request)
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\tornado\gen.py", line 735, in run
    value = future.result()
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\tornado\gen.py", line 748, in run
    yielded = self.gen.send(value)
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\server\contexts.py", line 215, in create_session_if_needed
    self._application.initialize_document(doc)
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\application\application.py", line 178, in initialize_document
    h.modify_document(doc)
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\application\handlers\function.py", line 133, in modify_document
    self._func(doc)
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bebi103\image.py", line 580, in modify_doc
    doc.add_root(bokeh.layouts.column(p, table))
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\document\document.py", line 304, in add_root
    self._pop_all_models_freeze()
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\document\document.py", line 1019, in _pop_all_models_freeze
    self._recompute_all_models()
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\document\document.py", line 1042, in _recompute_all_models
    a._attach_document(self)
  File "C:\Users\Max Ginsberg\Anaconda3\lib\site-packages\bokeh\model.py", line 714, in _attach_document
    raise RuntimeError("Models must be owned by only a single document, %r is already in a doc" % (self))
RuntimeError: Models must be owned by only a single document, ColumnDataSource(id='375285', ...) is already in a doc
```

In [ ]: