microburst_trains

January 4, 2022

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sampex_microburst_indices.load.catalog import Catalog
from sampex_microburst_indices.analysis.dial import Dial
```

1 Load the Microburst Catalog

Load the CSV file, parse the time stamps, remove rows with unfilled attitude data, and removes spin times.

[ ]:
```
cat = Catalog(0, parse_dates=True).load()
```

[ ]:
```
print(cat.columns, '\n', cat.shape)
```

```
Index(['burst_param', 'date', 'GEO_Long', 'GEO_Lat', 'Altitude', 'L_Shell',
       'MLT', 'Att_Flag', 'Pitch', 'AE', 'AL', 'AU', 'SYM/D', 'SYM/H', 'ASY/D',
       'ASY/H'],
      dtype='object')
 (244020, 16)
```

[ ]:
```
cat.head()
```

[ ]:
```
                         burst_param        date  GEO_Long  GEO_Lat  Altitude  \
dateTime
1997-11-09 19:56:40.720         35.3  1997-11-09   108.686  69.4370   681.303
1997-11-09 19:56:46.920         16.5  1997-11-09   109.084  69.1038   681.373
1997-11-09 19:57:02.440         10.4  1997-11-09   110.202  68.0984   681.558
1997-11-09 19:57:02.760         10.2  1997-11-09   110.202  68.0984   681.558
1997-11-09 19:57:02.980         10.4  1997-11-09   110.202  68.0984   681.558

                         L_Shell      MLT  Att_Flag    Pitch     AE     AL  \
dateTime
1997-11-09 19:56:40.720  5.47977  3.30990       0.0  32.9061  660.0 -518.0
1997-11-09 19:56:46.920  5.34191  3.33664       0.0  32.8835  660.0 -518.0
1997-11-09 19:57:02.440  4.99033  3.41316       0.0  32.8191  660.0 -518.0
```

```
1997-11-09 19:57:02.760  4.99033  3.41316       0.0  32.8191  660.0 -518.0
1997-11-09 19:57:02.980  4.99033  3.41316       0.0  32.8191  660.0 -518.0

                            AU  SYM/D  SYM/H  ASY/D  ASY/H
dateTime
1997-11-09 19:56:40.720  142.0   -5.0  -16.0   23.0   42.0
1997-11-09 19:56:46.920  142.0   -5.0  -16.0   23.0   42.0
1997-11-09 19:57:02.440  142.0   -5.0  -16.0   23.0   42.0
1997-11-09 19:57:02.760  142.0   -5.0  -16.0   23.0   42.0
1997-11-09 19:57:02.980  142.0   -5.0  -16.0   23.0   42.0
```

Calculate the time differences between each microburst. The first element is a NaN.

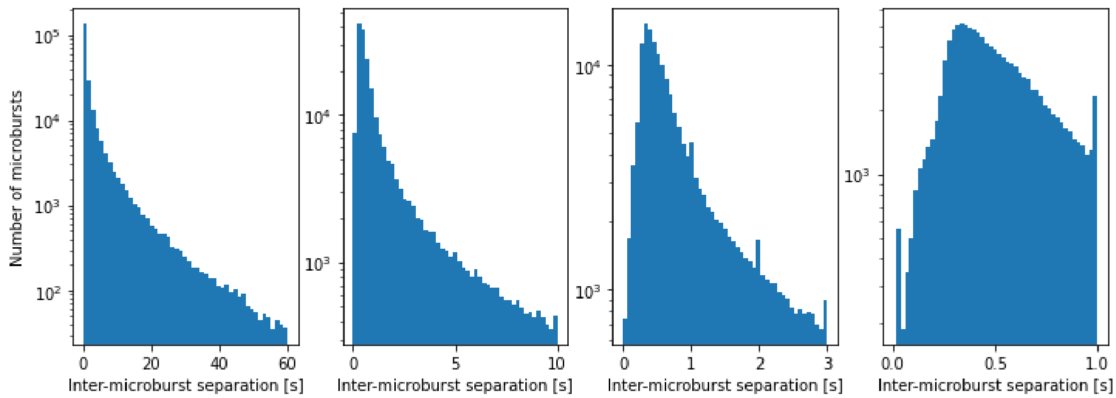
[ ]:
```python
cat['dt'] = cat.index.to_series().diff().dt.total_seconds()
# cat = cat.dropna()
cat['dt']
```

[ ]:
```
dateTime
1997-11-09 19:56:40.720          NaN
1997-11-09 19:56:46.920         6.20
1997-11-09 19:57:02.440        15.52
1997-11-09 19:57:02.760         0.32
1997-11-09 19:57:02.980         0.22
                                 …
2007-08-11 19:25:55.640         6.76
2007-08-11 19:25:55.900         0.26
2007-08-11 19:25:55.940         0.04
2007-08-11 19:25:56.240         0.30
2007-08-22 14:22:12.900    932176.66
Name: dt, Length: 244020, dtype: float64
```

Visualize the distribution of *dt* with varying scales.

[ ]:
```python
fig, ax = plt.subplots(1, 4, figsize=(12, 4))
ax[0].hist(cat['dt'], bins=np.linspace(0, 60))
ax[1].hist(cat['dt'], bins=np.linspace(0, 10))
ax[2].hist(cat['dt'], bins=np.linspace(0, 3))
ax[3].hist(cat['dt'], bins=np.linspace(0, 1))

for ax_i in ax:
    ax_i.set_yscale('log');
    ax_i.set_xlabel('Inter-microburst separation [s]')
ax[0].set_ylabel('Number of microbursts')

H, bins = np.histogram(cat['dt'], bins=np.linspace(0, 2))
print(f'The peak of the dt distribution is at {round(bins[np.argmax(H)], 3)} s.
 ↪')
# ax[3].axvline(bins[np.argmax(H)], color='k')
```

```
The peak of the dt distribution is at 0.327 s.
```



As we zoomed in, we found that the dt distribution is peaked at ~0.3 seconds.

[ ]:
```
np.nanquantile(cat['dt'], q=(0.25, .50, 0.75))
```

[ ]:
```
array([0.46, 0.96, 4.2 ])
```

[ ]:
```
np.nanmean(cat['dt'])
```

[ ]:
```
1265.0126923723153
```

… and it is heavily skewed by the outliars.

2 Calculate the microburst trains

- First, define a maximum time difference between sequential microbursts to qualify as a microburst train.
- Second, identify the start/end indices of `cat` for each microburst train. We use the `intervals()` function to calculate them.
- Third, create a microburst `trains` dataset where each row corresponds to a microburst train with `n_microbursts` the number of microbursts that were within `threshold_s` seconds.

[ ]:
```
threshold_s = 1
```

2.1 Step 2

[ ]:
```
def progressbar(i, n):
    """ A simple progressbar that I implemented in aurora-asi-lib. """
    progress_percent = round(100 * i / n)
    progress_str = "#" * (progress_percent // 5)
    print(f'Calculating intervals: |{progress_str:<20}| {progress_percent}%',␣
 ↪end='\r')
```

```
    return

def intervals(dt, threshold_s=1):
    """
    Given a list of time differences, dt, calculate the start and end indices␣
 ↪for
    intervals in which all dt < threshold_s (units of seconds).
    """
    i = 0
    indices = np.zeros((0, 2), dtype=int)

    while i < len(dt):

        progressbar(i, len(dt))

        if (dt[i] > threshold_s) or np.isnan(dt[i]):
            i += 1
            continue
        else:
            j = i
            if j == len(dt):
                break
            while dt[j] <= threshold_s:
                j += 1
            # This assumes pandas's inclusive index slicing.
            # Change j-1 to j for numpy indexing.
            indices = np.vstack((indices, [i-1, j-1]))
            i = j
    # Add a newline to avoid merging the progress bar text and the next print␣
 ↪line.
    print()
    return indices

indices = intervals(cat['dt'], threshold_s=threshold_s)
```

```
Calculating intervals: |####################| 100%
```

2.2 Validate microburst trains

The indices array correspond to the `cat` indices (pandas indexing) that are within `threshold_s`. Lets visually check that the first few indices match to the appropriate `cat` rows (hence we use `reset_index()` to add a numerical index for reference).

[ ]:
```
indices
```

[ ]:
```
array([[     2,      4],
       [    12,     13],
```

```
       [    14,     17],
       …,
       [244003, 244004],
       [244010, 244013],
       [244015, 244018]])
```

[ ]:
```
cat['dt'].reset_index().iloc[:20]
```

[ ]:
```
                  dateTime        dt
0  1997-11-09 19:56:40.720       NaN
1  1997-11-09 19:56:46.920      6.20
2  1997-11-09 19:57:02.440     15.52
3  1997-11-09 19:57:02.760      0.32
4  1997-11-09 19:57:02.980      0.22
5  1997-11-09 19:57:09.720      6.74
6  1997-11-09 20:42:16.280   2706.56
7  1997-11-10 00:31:21.700  13745.42
8  1997-11-10 00:47:48.780    987.08
9  1997-11-10 01:20:27.020   1958.24
10 1997-11-10 02:58:24.840   5877.82
11 1997-11-10 03:14:31.560    966.72
12 1997-11-10 04:52:21.960   5870.40
13 1997-11-10 04:52:22.540      0.58
14 1997-11-10 04:52:24.560      2.02
15 1997-11-10 04:52:25.180      0.62
16 1997-11-10 04:52:25.580      0.40
17 1997-11-10 04:52:26.500      0.92
18 1997-11-10 05:39:04.000   2797.50
19 1997-11-10 07:16:46.440   5862.44
```

2.3 Step 3: Create the trains dataset

[ ]:
```
trains = pd.DataFrame(
    data=np.nan*np.zeros((indices.shape[0], len(cat.columns)+1)),
    index=cat.index[indices[:,0]],
    columns=np.concatenate((cat.columns.to_numpy(), ['n_microbursts']))
)
```

[ ]:
```
for i, (start, end) in enumerate(indices):
    progressbar(i, indices.shape[0])
    trains.loc[cat.index[start], cat.columns] = cat.iloc[start:end, :].mean()
    # The +1 is to avoid averaging the dt from the previous gap and skewing the␣
 ↪true mean time
    # separation between the microbursts in that microburst train
    trains.loc[cat.index[start], 'dt'] = cat.loc[cat.index[start+1]:cat.
 ↪index[end], 'dt'].mean()
```

```
    trains.loc[cat.index[start], 'n_microbursts'] = end-start+1
trains.drop(columns=['burst_param', 'date'], inplace=True)
```

```
Calculating intervals: |                    | 0%

/tmp/ipykernel_133751/698226168.py:3: FutureWarning: Dropping of nuisance
columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a
future version this will raise TypeError.  Select only valid columns before
calling the reduction.
  trains.loc[cat.index[start], cat.columns] = cat.iloc[start:end, :].mean()

Calculating intervals: |####################| 100%
```

[ ]:
```
trains.shape
```

[ ]:
```
(45648, 16)
```

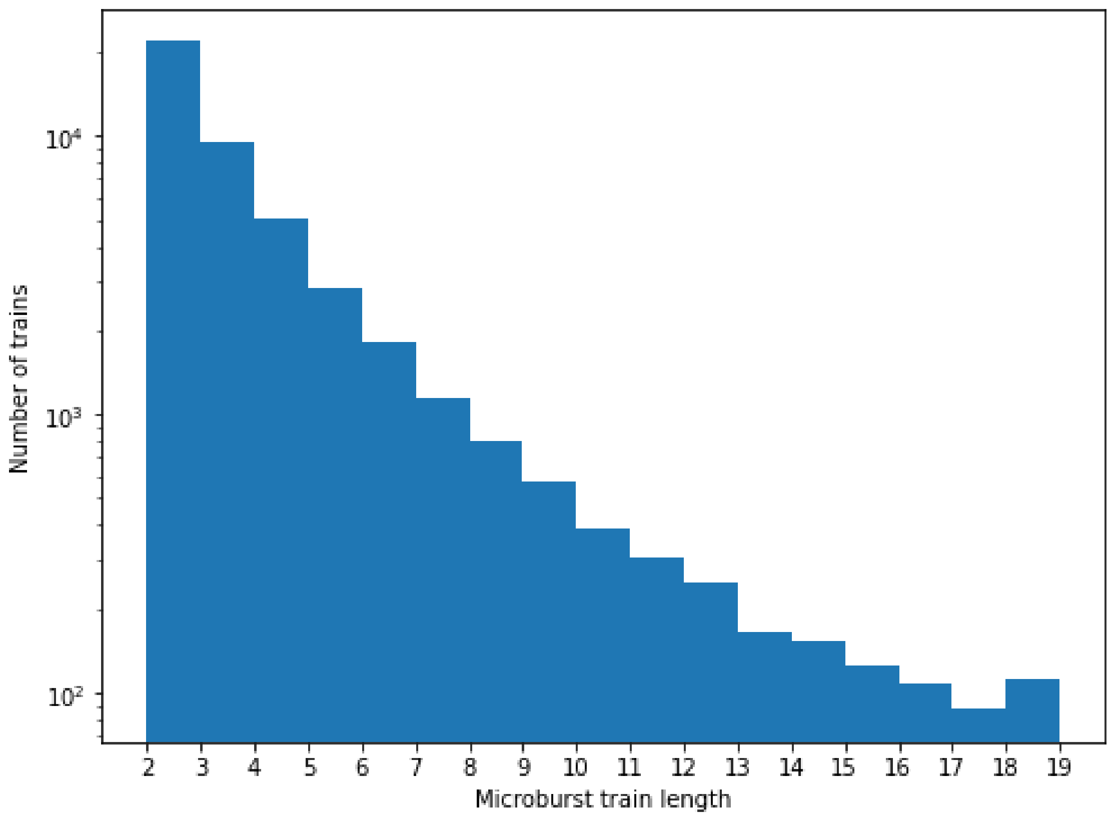
[ ]:
```
trains.head()
```

[ ]:
```
                         GEO_Long  GEO_Lat  Altitude  L_Shell       MLT  \
dateTime
1997-11-09 19:57:02.440   110.202  68.0984   681.558  4.99033   3.41316
1997-11-10 04:52:21.960   165.599 -53.4134   531.769  4.82424  17.46640
1997-11-10 04:52:24.560   165.728 -53.0439   531.669  4.69422  17.45850
1997-11-10 07:17:11.700   309.107  53.6210   680.153  4.55372   5.23839
1997-11-14 12:38:19.000   215.561  62.1377   681.167  5.36051   2.23292

                         Att_Flag     Pitch     AE     AL     AU  SYM/D  \
dateTime
1997-11-09 19:57:02.440       0.0   32.8191  660.0 -518.0  142.0   -5.0
1997-11-10 04:52:21.960       0.0  155.8127  257.0  -58.0  199.0    4.0
1997-11-10 04:52:24.560       0.0  155.9210  257.0  -58.0  199.0    4.0
1997-11-10 07:17:11.700       0.0   25.8763  399.0 -298.0  101.0    5.0
1997-11-14 12:38:19.000       0.0   45.8750  405.0 -340.0   65.0    4.0

                         SYM/H  ASY/D  ASY/H        dt  n_microbursts
dateTime
1997-11-09 19:57:02.440  -16.0   23.0   42.0  0.270000            3.0
1997-11-10 04:52:21.960  -43.0   32.0   33.0  0.580000            2.0
1997-11-10 04:52:24.560  -43.0   32.0   33.0  0.646667            4.0
1997-11-10 07:17:11.700  -34.0   29.0   51.0  0.980000            2.0
1997-11-14 12:38:19.000  -22.0   16.0   45.0  0.710000            3.0
```

3 Visualize microburst train statistics

3.1 First: what is the distribution of the number of microbursts in each microburst train?

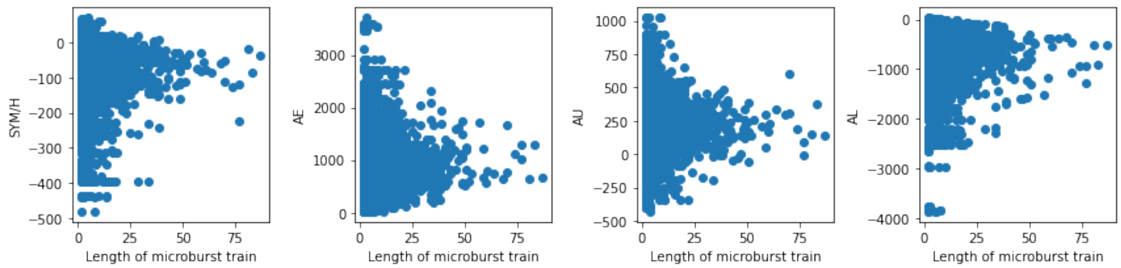
[ ]:
```
train_bins = np.arange(2, 20)
fig, cx = plt.subplots(figsize=(8,6))
cx.hist(trains['n_microbursts'], bins=train_bins)
cx.set_xticks(train_bins)
cx.set(xlabel='Microburst train length', ylabel='Number of trains');
cx.set_yscale('log')
```



A train of two microbursts seems to be the most common by a factor of two. **Is this good enough criteria for a microburst train?** Two microbursts within `threshold_s` could be due to chance.

[ ]:
```
scatter_index_names = ['SYM/H', 'AE', 'AU', 'AL']

fig, dx = plt.subplots(1, len(scatter_index_names), figsize=(12, 3))

for dx_i, scatter_index_name in zip(dx, scatter_index_names):
    dx_i.scatter(trains['n_microbursts'], trains[scatter_index_name])
    dx_i.set_ylabel(scatter_index_name)
```

```
    dx_i.set_xlabel('Length of microburst train')

plt.tight_layout()
```



Hmm. I don't see any strong patterns as a function of geomagnetic indices. **Maybe I should make a microburst catalog using the running average OMNI values?** (that will be easy to implement.)

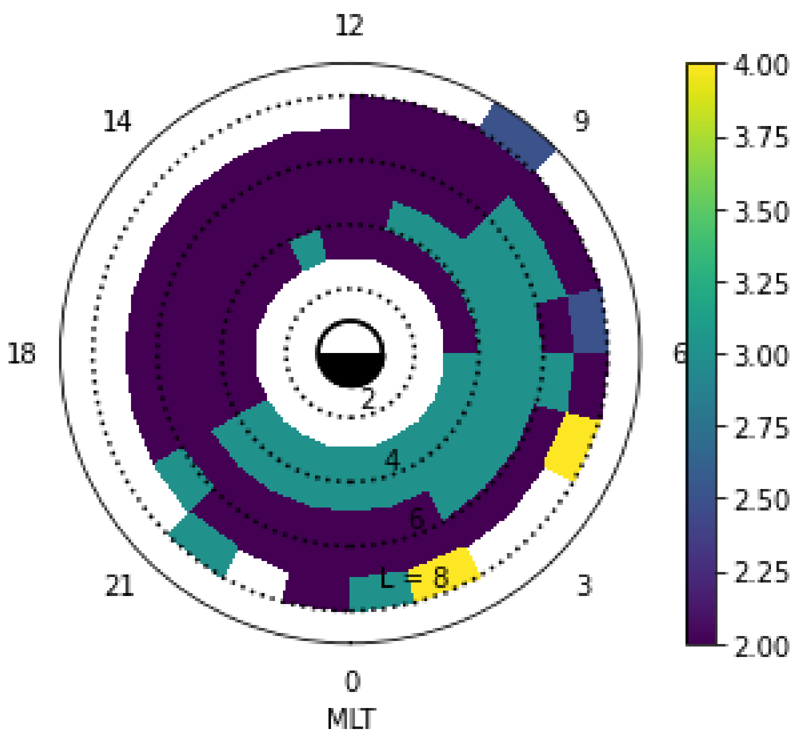
3.2 Second: The L-MLT distribution

Lets plot the median length of a microburst train as a function of L and MLT.

[ ]:
```
L_bins = np.arange(3, 10)
MLT_bins = np.arange(0, 24.1)
percentiles = np.array([50])

H = np.nan*np.zeros(
    (len(MLT_bins), len(L_bins), len(percentiles))
                    )

for i, (start_MLT, end_MLT) in enumerate(zip(MLT_bins[:-1], MLT_bins[1:])):
    for j, (start_L, end_L) in enumerate(zip(L_bins[:-1], L_bins[1:])):
        df_flt = trains.loc[(
            (trains['MLT'] > start_MLT) &  (trains['MLT'] < end_MLT) &
            (trains['L_Shell'] > start_L) &  (trains['L_Shell'] < end_L)
            ), 'n_microbursts']
        if df_flt.shape[0] >= 10:
            H[i, j, :] = df_flt.quantile(percentiles/100)

fig, ex = plt.subplots(subplot_kw={'projection': 'polar'})
d = Dial(ex, MLT_bins, L_bins, H[:,:,0])
d.draw_earth()
d.draw_dial()
```

```
/home/mike/research/sampex_microburst_indices/sampex_microburst_indices/analysis
/dial.py:38: MatplotlibDeprecationWarning: shading='flat' when X and Y have the
same dimensions as C is deprecated since 3.3.  Either specify the corners of the
```

```
quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or
set rcParams['pcolor.shading'].  This will become an error two minor releases
later.
  p = self.ax.pcolormesh(angular_grid*np.pi/12, radial_grid, self.H.T,
**mesh_kwargs)
/home/mike/research/sampex_microburst_indices/sampex_microburst_indices/analysis
/dial.py:73: UserWarning: FixedFormatter should only be used together with
FixedLocator
  self.ax.set_xticklabels(mlt_labels) # Transform back from 0->2pi to 0->24.
```

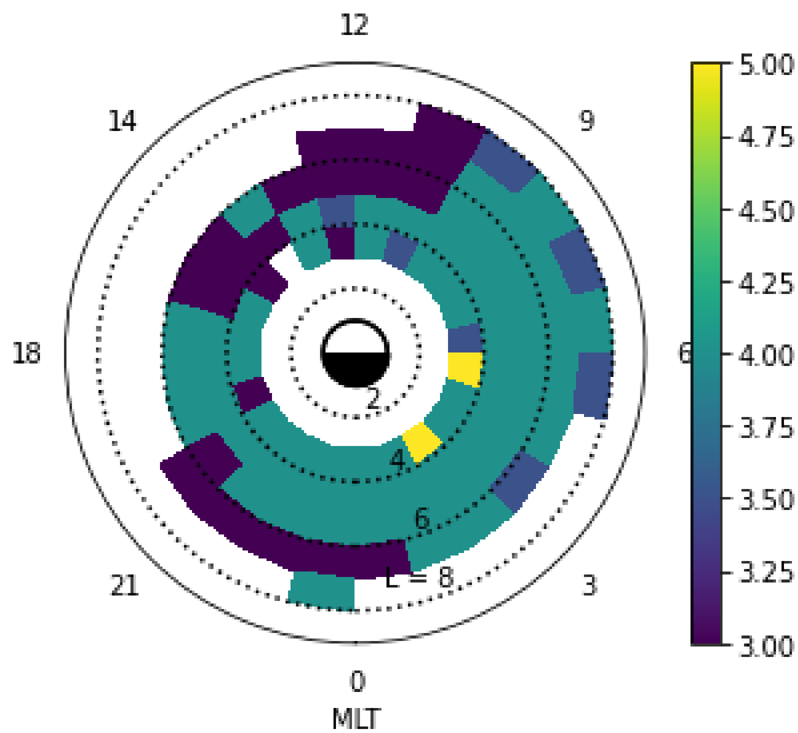


Hmm. A slight trend, but nothing convincing. Let's try again after filtering out trains with less than `n` microbursts

[ ]:
```python
n = 2
long_trains = trains[trains['n_microbursts'] > n]

L_bins = np.arange(3, 10)
MLT_bins = np.arange(0, 24.1)
percentiles = np.array([50])

H = np.nan*np.zeros(
    (len(MLT_bins), len(L_bins), len(percentiles))
                    )
```

```
for i, (start_MLT, end_MLT) in enumerate(zip(MLT_bins[:-1], MLT_bins[1:])):
    for j, (start_L, end_L) in enumerate(zip(L_bins[:-1], L_bins[1:])):
        df_flt = long_trains.loc[(
            (long_trains['MLT'] > start_MLT) &  (long_trains['MLT'] < end_MLT) &
            (long_trains['L_Shell'] > start_L) &  (long_trains['L_Shell'] <␣
↪end_L)
            ), 'n_microbursts']
        if df_flt.shape[0] >= 10:
            H[i, j, :] = df_flt.quantile(percentiles/100)

fig, ex = plt.subplots(subplot_kw={'projection': 'polar'})
d = Dial(ex, MLT_bins, L_bins, H[:,:,0])
d.draw_earth()
d.draw_dial()
```

```
/home/mike/research/sampex_microburst_indices/sampex_microburst_indices/analysis
/dial.py:38: MatplotlibDeprecationWarning: shading='flat' when X and Y have the
same dimensions as C is deprecated since 3.3.  Either specify the corners of the
quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or
set rcParams['pcolor.shading'].  This will become an error two minor releases
later.
  p = self.ax.pcolormesh(angular_grid*np.pi/12, radial_grid, self.H.T,
**mesh_kwargs)
/home/mike/research/sampex_microburst_indices/sampex_microburst_indices/analysis
/dial.py:73: UserWarning: FixedFormatter should only be used together with
FixedLocator
  self.ax.set_xticklabels(mlt_labels) # Transform back from 0->2pi to 0->24.
```



Hmm. A qualitatively similar distribution. I have a hunch that there is a better way to tease out a trend but I can't think of anything now. Until next time!