Optimal GHA Bin Sizes

February 2, 2021

David Lewis, Steven Murray
EDGES ASU Memo Series
#188

1 Motivation

As we use EDGES to look for evidence of the 21cm redshifted EoR signal, we want to determine the best way to filter through the data for each night of observation. RFI is present at many frequencies at different times in EDGES observations. This RFI is flagged in time and frequency. Later, in the EDGES data reduction pipeline, we wish to bin data in time and frequency to reduce the total data volume during analysis. However, bins that include flagged data can have biased estimates of the true bin center value. Here, our goal is to determine the largest bin size in both time (GHA) and frequency so that our signal is not biased from the true average more than a certain amount. The maximum bias we will tolerate is about 10 mK. Given sky temperatures of order 2000-10,000 K, this is 0.001% to 0.0001% of the original signal.

2 Method

We create a model of the sky based in both frequency and time. We also create an error model for both of these based on flagging fraction and bin size. The error for all different permutations is calculated and plotted. Finally, based on these models, a theoretical minimum bin size is calculated for the time-dependent model.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors
from matplotlib.ticker import LogFormatter
from matplotlib.ticker import FormatStrFormatter, ScalarFormatter
import matplotlib.ticker as ticker
import matplotlib
%matplotlib inline
```

3 Error Modeling

To get a more granular view of possible bias, we consider the error caused by both frequency and time bin size at arbitrary flag fractions in some basic models, as defined by the difference between the normalized integrals of a full spectrum and a spectrum with a variably sized flag fraction.

3.0.1 For the Frequency Part

For the frequency part of our error modeling, the model is

$$T(\nu) = T_0(\nu/75)^{-\alpha} \tag{1}$$

The equation for our error model is:

$$\epsilon = \frac{1}{d\nu}\int_{\nu_0}^{\nu_0+d\nu} T(\nu) - \frac{1}{(1-f)d\nu}\int_{\nu_0}^{\nu_0+(1-f)d\nu} T(\nu) \tag{2}$$

$$\epsilon = \frac{1}{d\nu}\left[\frac{T_0}{1-\alpha}(\nu/75)^{1-\alpha}\right]_{\nu_0}^{\nu_0+d\nu} - \frac{1}{(1-f)d\nu}\left[\frac{T_0}{1-\alpha}(\nu/75)^{1-\alpha}\right]_{\nu_0}^{\nu_0+(1-f)d\nu} \tag{3}$$

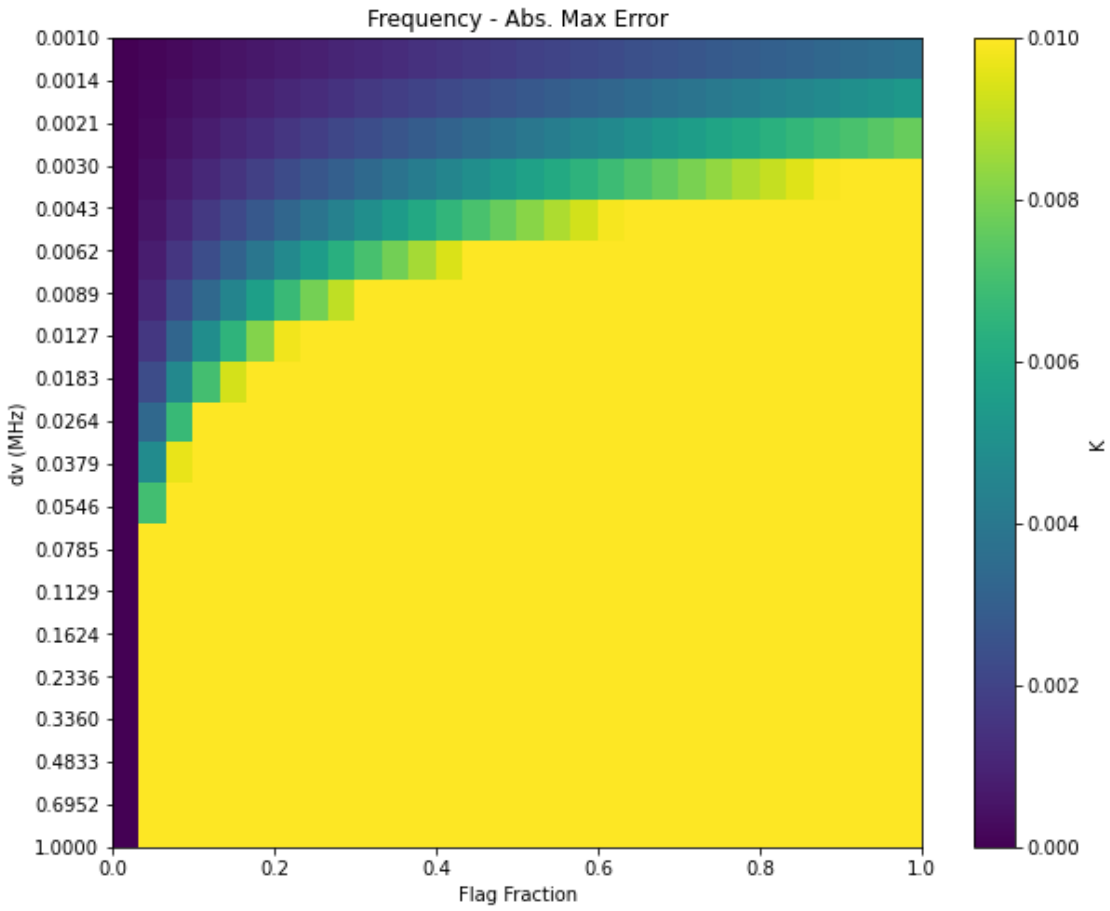
$$\epsilon = \frac{1}{d\nu}\frac{T_0}{1-\alpha}\left(\left[((\nu_0+d\nu)/75)^{1-\alpha} - (\nu_0/75)^{1-\alpha}\right] - \frac{1}{(1-f)}\left[((\nu_0+(1-f)d\nu)/75)^{1-\alpha} - (\nu_0/75)^{1-\alpha}\right]\right) \tag{4}$$

$$\epsilon = \frac{1}{d\nu}\frac{T_0}{1-\alpha}\left(\left[((\nu_0+d\nu)/75)^{1-\alpha} + (\nu_0/75)^{1-\alpha}\frac{f}{1-f}\right] - \frac{1}{(1-f)}\left[((\nu_0+(1-f)d\nu)/75)^{1-\alpha}\right]\right) \tag{5}$$

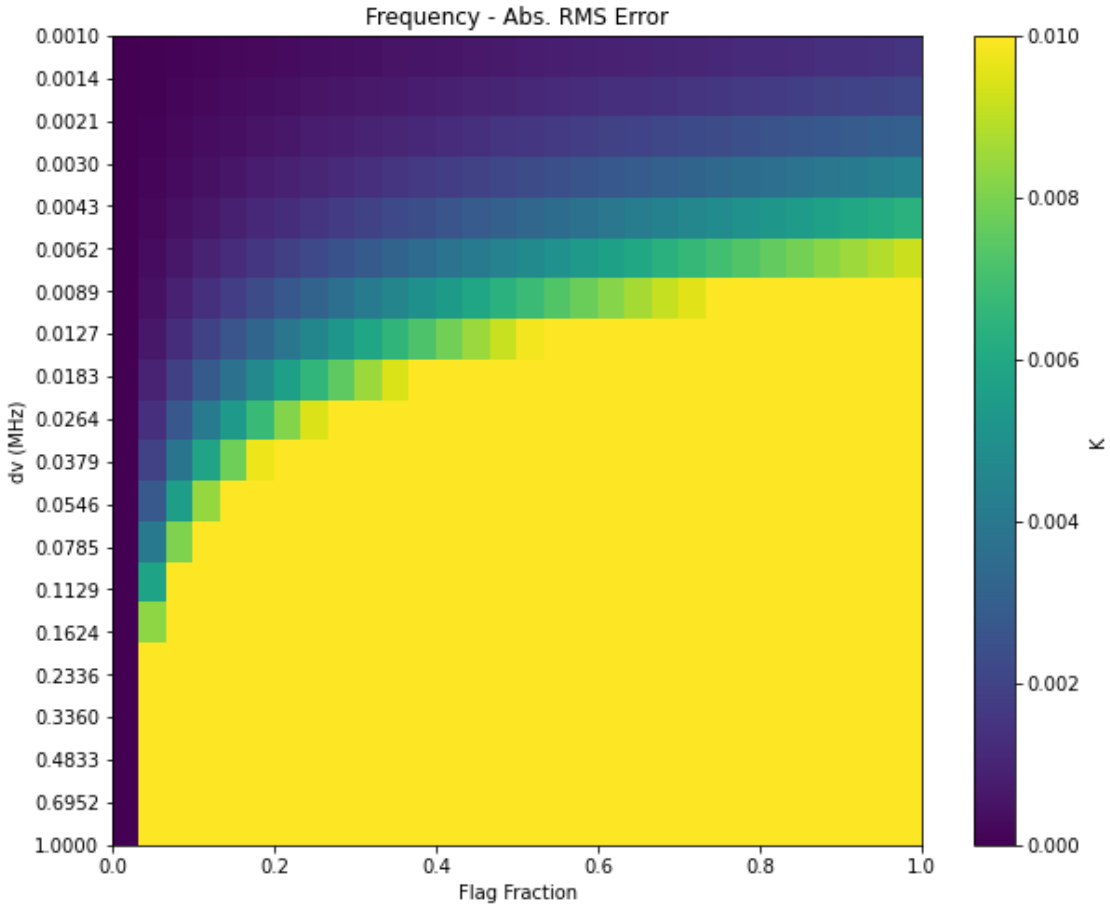
[2]:
```
def error_freq(nu, T0, alpha, dv, ff):
    front = T0 / (dv * (1 - alpha))
    first = ((nu + dv) / 76)**(1 - alpha)
    second = (nu / 76)**(1-alpha) * ff/(1 - ff)
    third = ((nu + (1 - ff)*dv) / 76)**(1 - alpha) / (1 - ff)
    return front * (first + second - third)
```

The resulting error function was then calculated using a range of flag fractions from 0.0-0.95 and frequency bins from 1kHz to 1MHz. We find that the resulting maximum error from these parameters remains under 10mK under all frequency bins below 2 kHz, with higher flag fractions leading to errors above 10mK at larger bin sizes. RMS error remains under this threshold with up to 6 kHz bins, and errors increase once again with higher flag fractions above this frequency bin size. This suggests that we cannot bin larger than our current default bin size of 6.1 kHz unless additional techniques are implemented to mitigate this.

[3]:
```
#nu = np.linspace(50, 100, 51)
dv = np.logspace(-3, 0, 20)
ff = np.linspace(0, 0.95, 30)
T0 = 4000
alpha = 2.55
# Different metrics
max_error = np.zeros((len(dv), len(ff)))
rms_error = np.zeros((len(dv), len(ff)))
#error_at_90 = np.zeros((len(dv), len(ff)))
#error_at_90 = []


for i, dnu in enumerate(dv):
    nu = np.linspace(50, 100, np.int(51/dnu))
    for j, fff in enumerate(ff):
        errors = error_freq(nu, T0, alpha, dnu, fff)
        max_error[i,j] = np.max(abs(errors))
        rms_error[i,j] = np.sqrt(np.mean(np.square(errors)))
```



Above: Absolute Maximum Error for all frequencies using bins from 1khz to 1MHz. For all flag fractions, error remains under our desired levels (~10mK) with bins up to 2KHz. Above this higher flag fractions produce error exceeding 10mK.



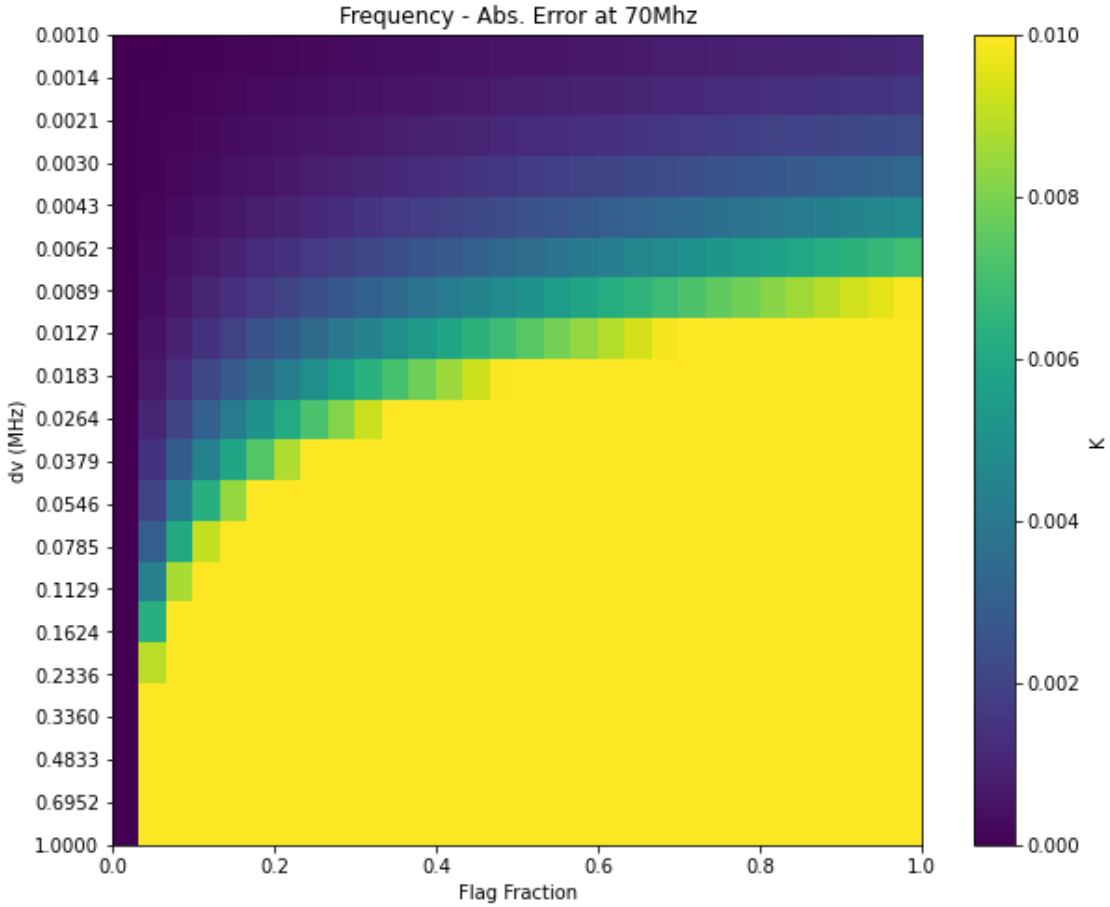
Above: Absolute RMS Error for all frequencies using bins from 1khz to 1MHz. For all flag fractions, error remains under our desired levels (~10mK) only in our smallest bin of 6.1kHz. Above this higher flag fractions produce error exceeding 10mK.

This was replotted with only the 76MHz and 90MHz frequency, using the same dv and ff value ranges. On average, the values of the 76 MHz errors were nearly twice as high than the 90 MHz errors with the same flag fraction and bin size. 76 MHz bins were able to remain under 10mK error until bin sizes exceeded 8.9 kHz, while 90Mhz error remained low until bin size exceeded 18KHz.

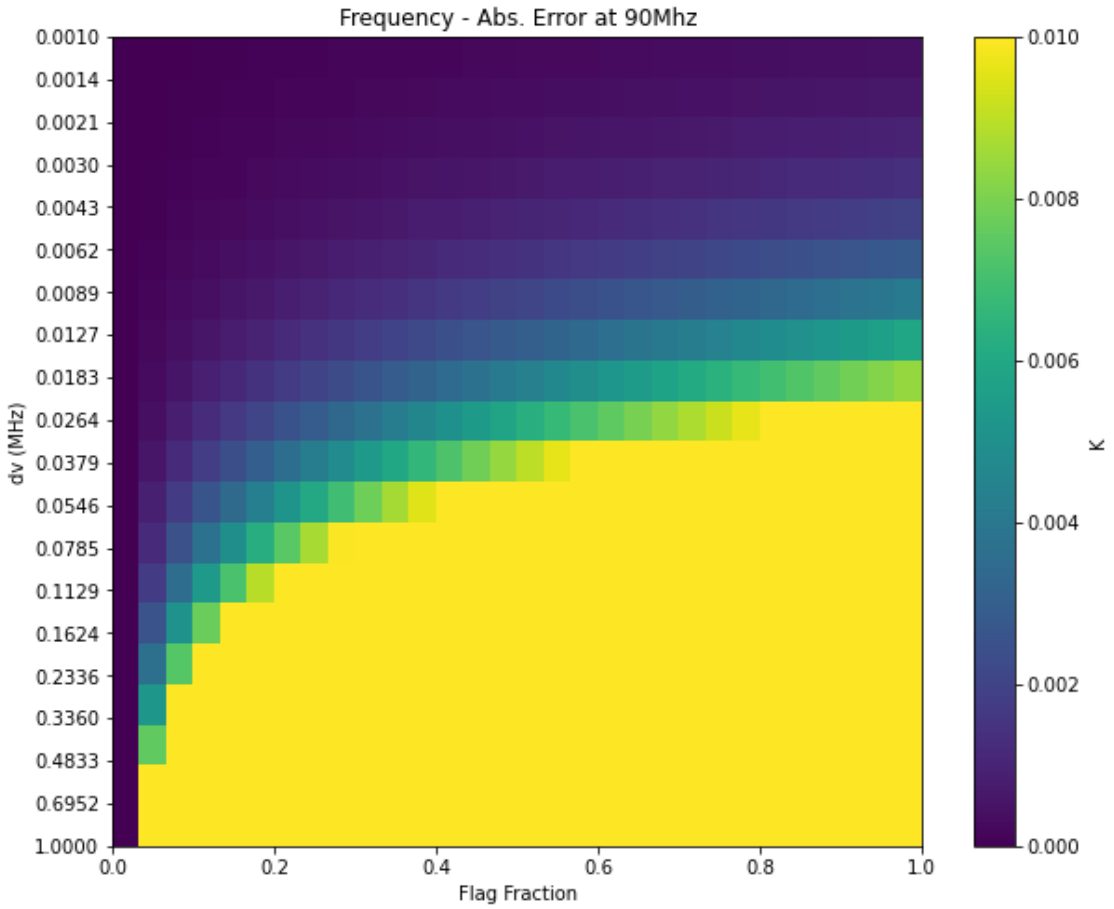
[8]:
```
#90 MHz
dv = np.logspace(-3, 0, 20)
ff = np.linspace(0, 0.95, 30)
T0 = 4000
alpha = 2.55
nu=90
error_at_90 = np.zeros((len(dv), len(ff)))
```

```
for i, dnu in enumerate(dv):
    for j, fff in enumerate(ff):
        errors90 = error_freq(nu, T0, alpha, dnu, fff)
        error_at_90[i,j] = errors90
```

[9]:
```
#70 MHz
dv = np.logspace(-3, 0, 20)
ff = np.linspace(0, 0.95, 30)
T0 = 4000
alpha = 2.55
nu=70
error_at_70 = np.zeros((len(dv), len(ff)))
for i, dnu in enumerate(dv):
    for j, fff in enumerate(ff):
        errors70 = error_freq(nu, T0, alpha, dnu, fff)
        error_at_70[i,j] = errors70
```

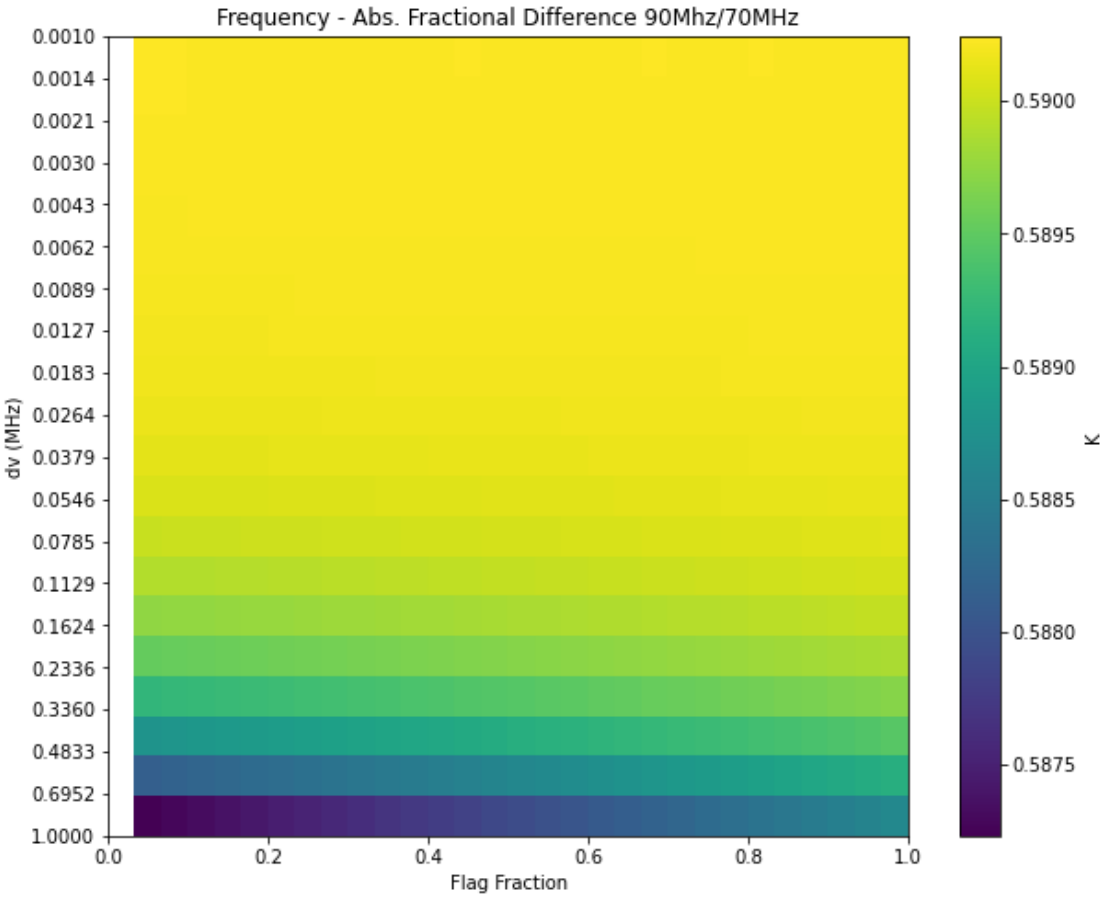


Above: Absolute Maximum Error for 70 MHz using bins from 1khz to 1MHz.



Above: Absolute Maximum Error for 90 MHz using bins from 1khz to 1MHz.

```
<ipython-input-12-21e3ad880e76>:2: RuntimeWarning: invalid value encountered in
true_divide
  plt.imshow(abs(abs(error_at_90) - abs(error_at_70))/abs(error_at_70),
extent=[0,1, dv[-1], dv[0]])
```



Above: Fractional Difference between 76 and 90 MHz error values, as related to 76 MHz. Though there is a difference, this difference is fairly stable across flag fraction and bin size.

3.1 For the Time Part

We repeated this process for the time portion of the model, beginning with a polynomial. Spectra were created using the Haslam 408 MHz All-Sky Map, adjusted down to the EDGES range between 50 and 100 MHz. A single spectrum at 76 MHz across all GHA was fitted with a 9 term polynomial model, and the error model was once again taken by the difference between the binned spectrum and the flag-fractioned spectrum.

Model is polynomial:

$$T(t) = a + bt + ct^2 + dt^3 + \dots \tag{6}$$

Model of Error

$$\epsilon = \frac{1}{dt}\int_{t_0+ndt}^{t_0+(n+1)dt} T(t) - \frac{1}{(1-f)dt}\int_{t_0+ndt}^{t_0+(1-f+n)dt} T(t) \tag{7}$$

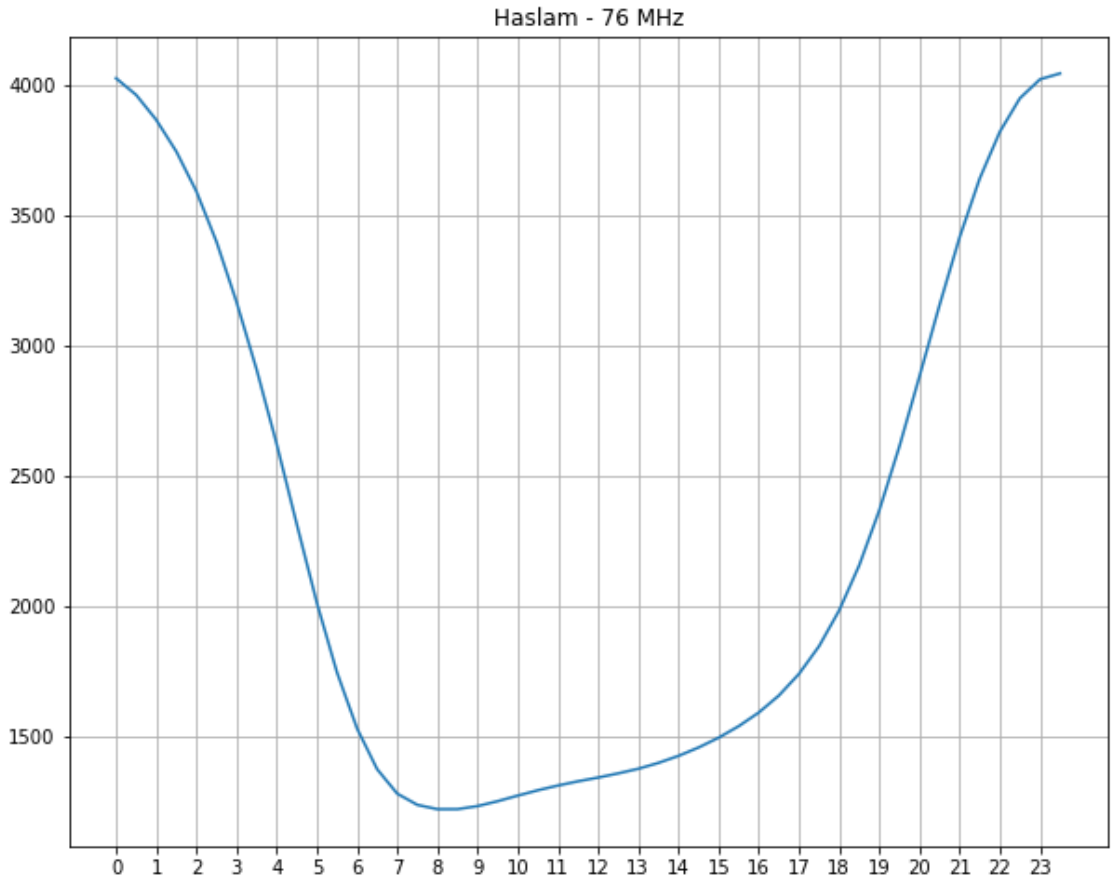
Where n is the bin number, dt is bin size

$$\epsilon = \frac{1}{dt}\left[at + \frac{bt^2}{2} + \frac{ct^3}{3} + \frac{dt^4}{4} + \ldots\right]_{t_0+ndt}^{t_0+(n+1)dt} - \frac{1}{(1-f)dt}\left[at + \frac{bt^2}{2} + \frac{ct^3}{3} + \frac{dt^4}{4} + \ldots\right]_{t_0+ndt}^{t_0+(1-f+n)dt} \tag{8}$$

[13]:
```
def error_time(t,dt,ff):
    t_0=t[0]
    first = np.zeros((len(t)))
    second=np.zeros((len(t)))
    for n, time in enumerate(t):
        first[n] = (1/dt)*(tint(t_0+(n+1)*dt) - tint(t_0+(n*dt)))
        second[n] = (1/((1-ff)*dt))*(tint(t_0+(n+1-ff)*dt) - tint(t_0+(n*dt)))
    return first-second
```

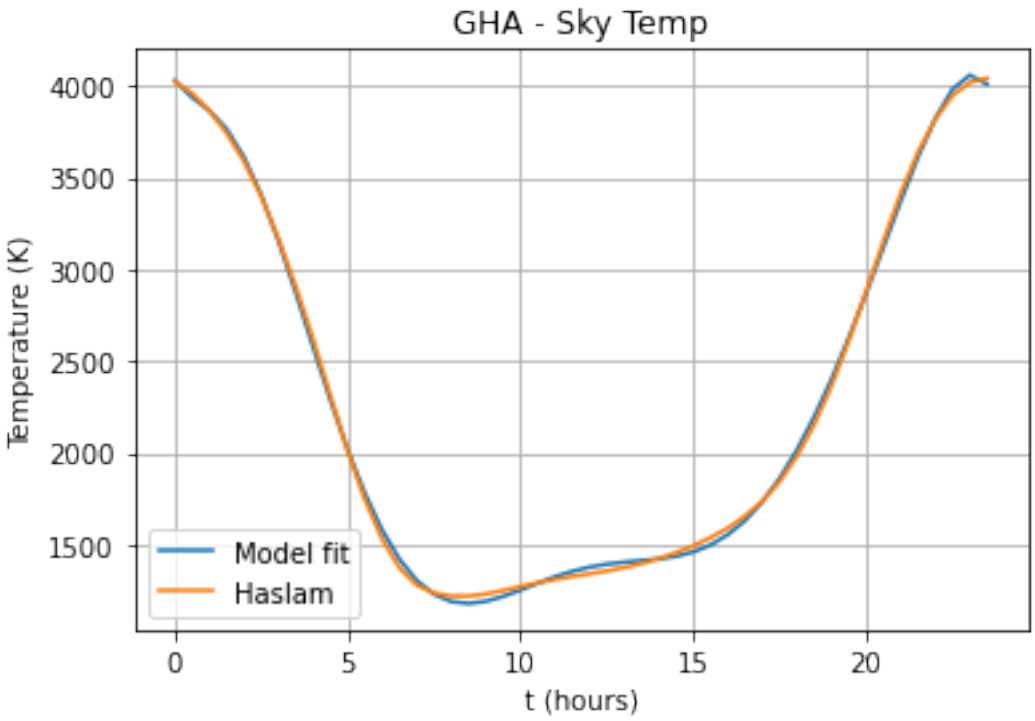
[14]:
```
#get haslam spectrum
haslam = np.load('haslam_simulated_driftscan_25.npz')
haslam_freq = haslam['frequency']
haslam_lst = haslam['lst']
haslam_temp = haslam['antenna_temp']
```

3.1.1 Analysis at 76 MHz

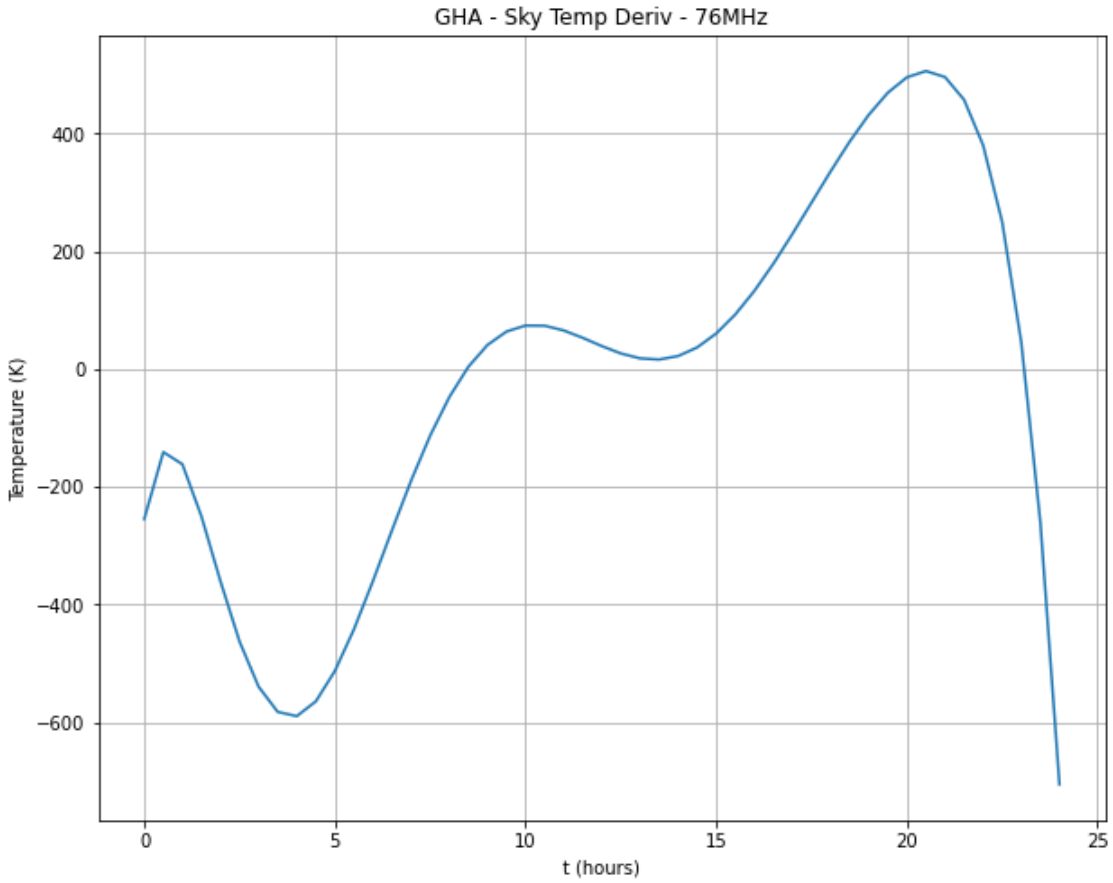


Above: Sky temperature of the Haslam Sky Map in the EDGES Beam at 76MHz.

[18]:
```
t = np.linspace(0, 24, 49)
tfit = np.polyfit(t[:-1], haslam_temp[:,freq_ind], deg=9) #76 MHz
tt = np.poly1d(tfit)
tint=tt.integ()
tder=tt.deriv()
```



Above: Tsky 76MHz temperature and the 9-degree model fitted to it.



Above: TSky temperature derivative at 76 MHz.

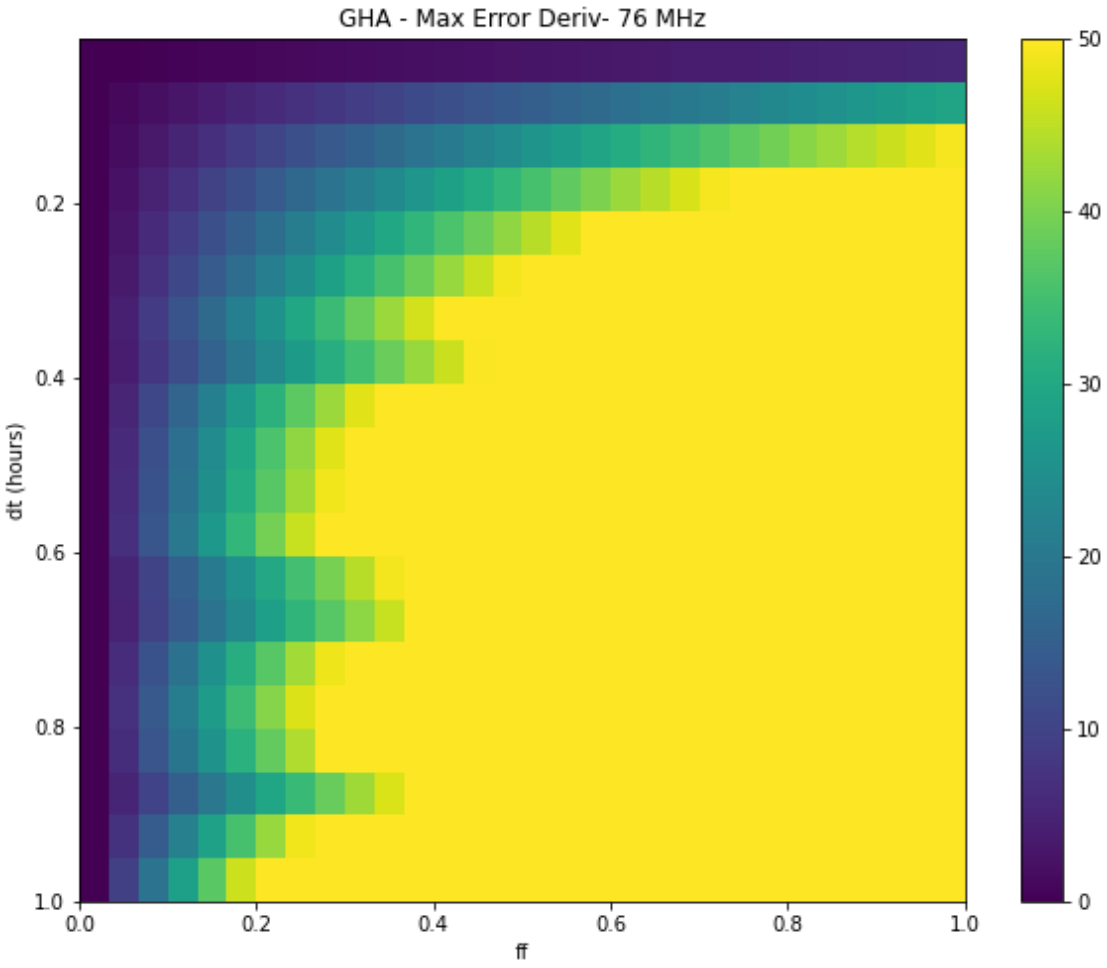
[21]:
```
np.min(tder(t)),np.max(tder(t))
```

[21]:
```
(-705.1985551077863, 505.90513181907215)
```

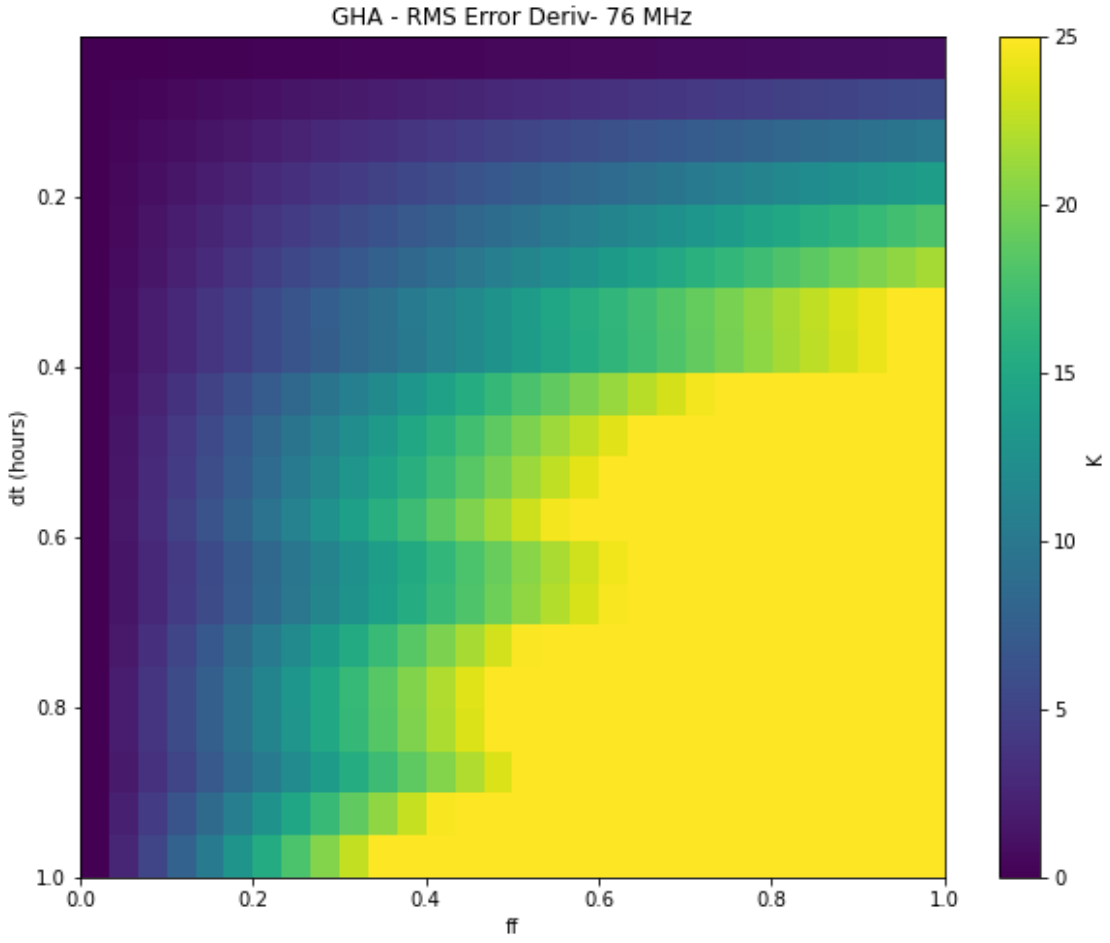
[22]:
```
#use parameters to calculate error function
#t = np.linspace(0, 24, 48) #hours
ff = np.linspace(0, 0.95, 30)
#base_bin = 39 #seconds
dt = np.linspace(0.01083,1,20) #hours (39 sec to 1 hour)


# Different metrics
#full_error = np.zeros((len(dt), len(ff), len(t)))
max_error = np.zeros((len(dt), len(ff)))
rms_error = np.zeros((len(dt), len(ff)))


for i, dtt in enumerate(dt):
```

```
    t = np.linspace(0, 24, np.int(24/dtt))
    for j, fff in enumerate(ff):
        errors = error_time(t,dtt,fff)
        #full_error[i,j] = errors
        max_error[i,j] = np.max(abs(errors))
        rms_error[i,j] = np.sqrt(np.mean(np.square(errors)))
```

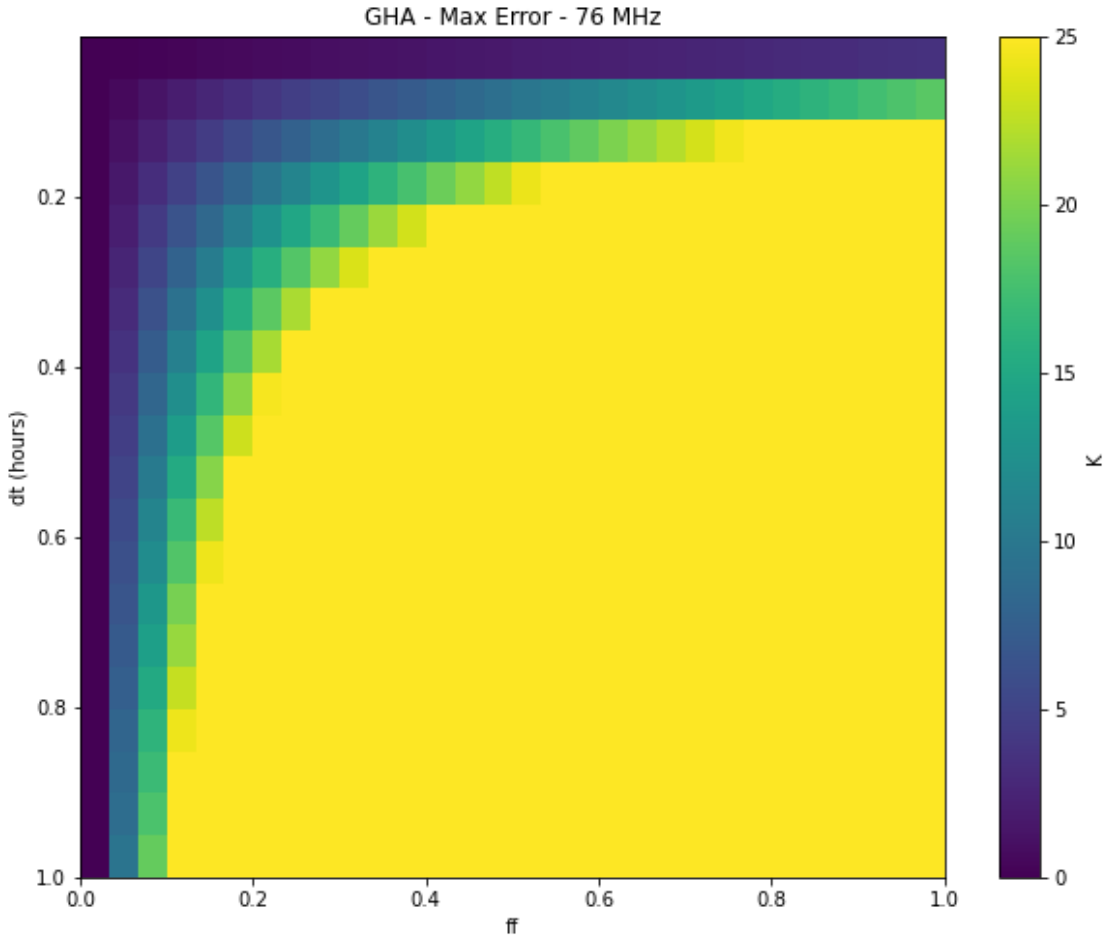
[23]:
```
#use parameters to calculate error function
#t = np.linspace(0, 24, 48) #hours
ff = np.linspace(0, 0.95, 30)
#base_bin = 39 #seconds
dt = np.linspace(0.01083,1,20) #hours (39 sec to 1 hour)


# Different metrics
#full_error = np.zeros((len(dt), len(ff), len(t)))
max_error_d = np.zeros((len(dt), len(ff)))
rms_error_d = np.zeros((len(dt), len(ff)))


for i, dtt in enumerate(dt):
    t = np.linspace(0, 24, np.int(24/dtt))
    errors_d = np.zeros((len(t)-1))
    for j, fff in enumerate(ff):
        errors = error_time(t,dtt,fff)
        for e, err in enumerate(errors[:-1]):
            errors_d[e] = (errors[e+1]-err)/dtt

        #full_error[i,j] = errors
        max_error_d[i,j] = np.max(abs(errors_d))
        rms_error_d[i,j] = np.sqrt(np.mean(np.square(errors_d)))
```



Above: The maximum absolute derivative of the 76 MHz error for each bin size.

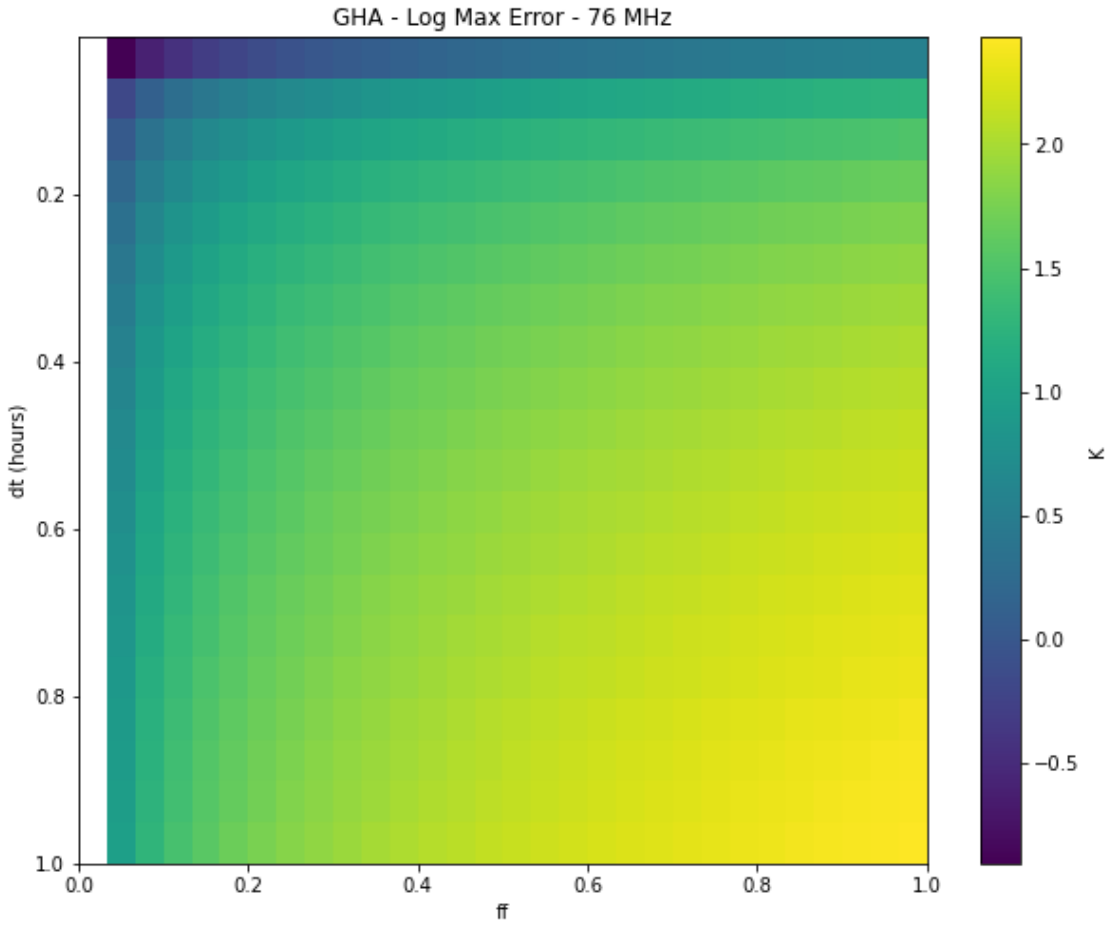


Above: The RMS absolute derivative of the 76 MHz error for each bin size.

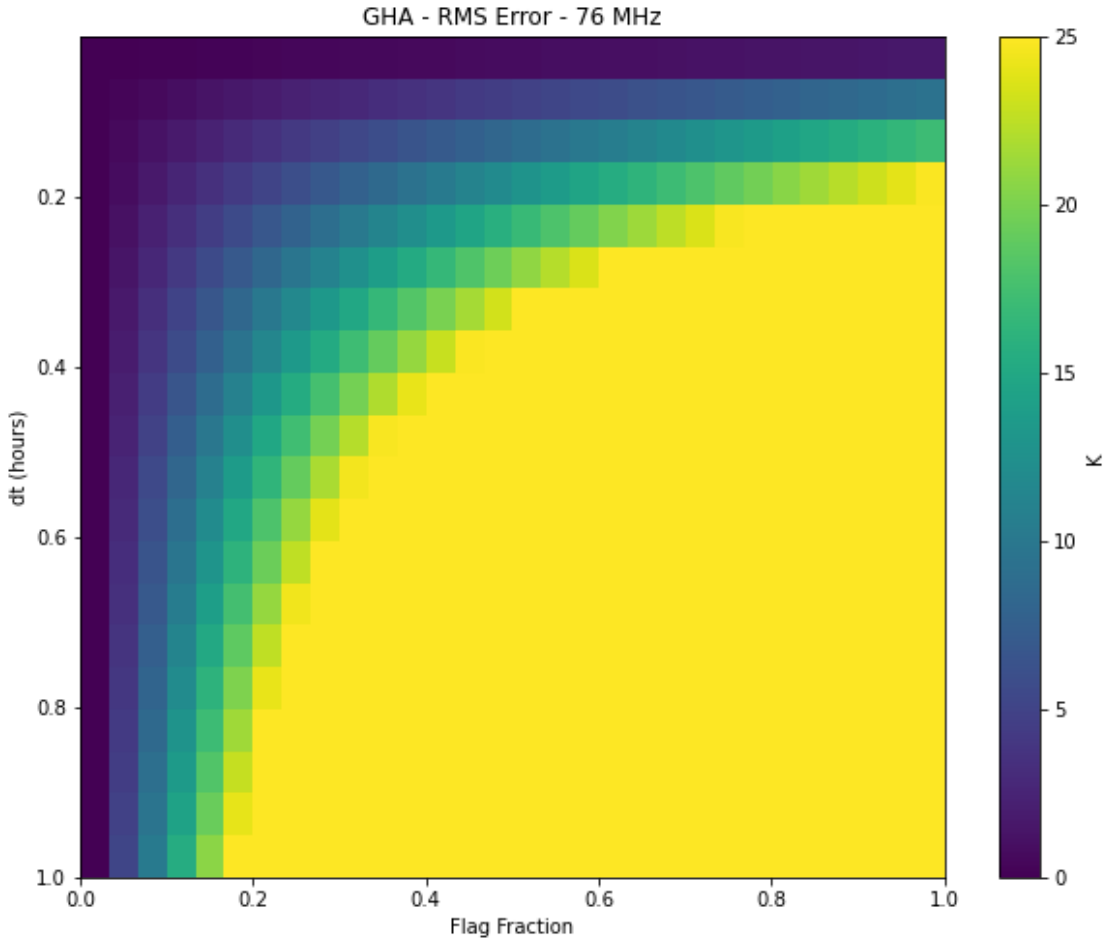


Above: Absolute Maximum Error of the 76 MHz error values.

```
<ipython-input-30-aabc3c151a3d>:2: RuntimeWarning: divide by zero encountered in
log10
  plt.imshow(np.log10(abs(max_error)),extent=[0,1, dt[-1], dt[0]],
aspect='auto')
```

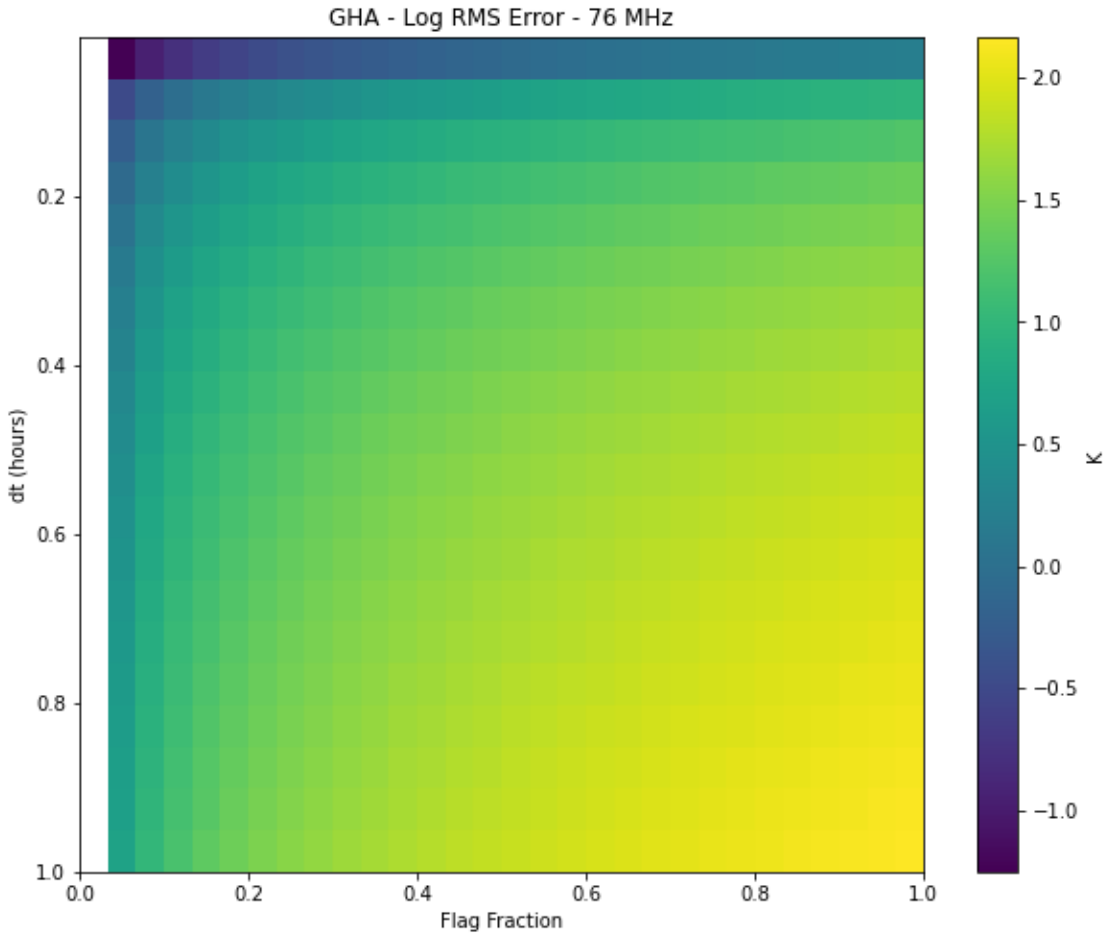


Above: Log of the Maximum error of 76 MHz error values.



Above: RMS Error of 76 MHz error values.

```
<ipython-input-32-29a85e50b8d5>:2: RuntimeWarning: divide by zero encountered in
log10
  plt.imshow(np.log10(abs(rms_error)), extent=[0,1, dt[-1], dt[0]])
```



Above: Log RMS Error of 76 MHz error values.

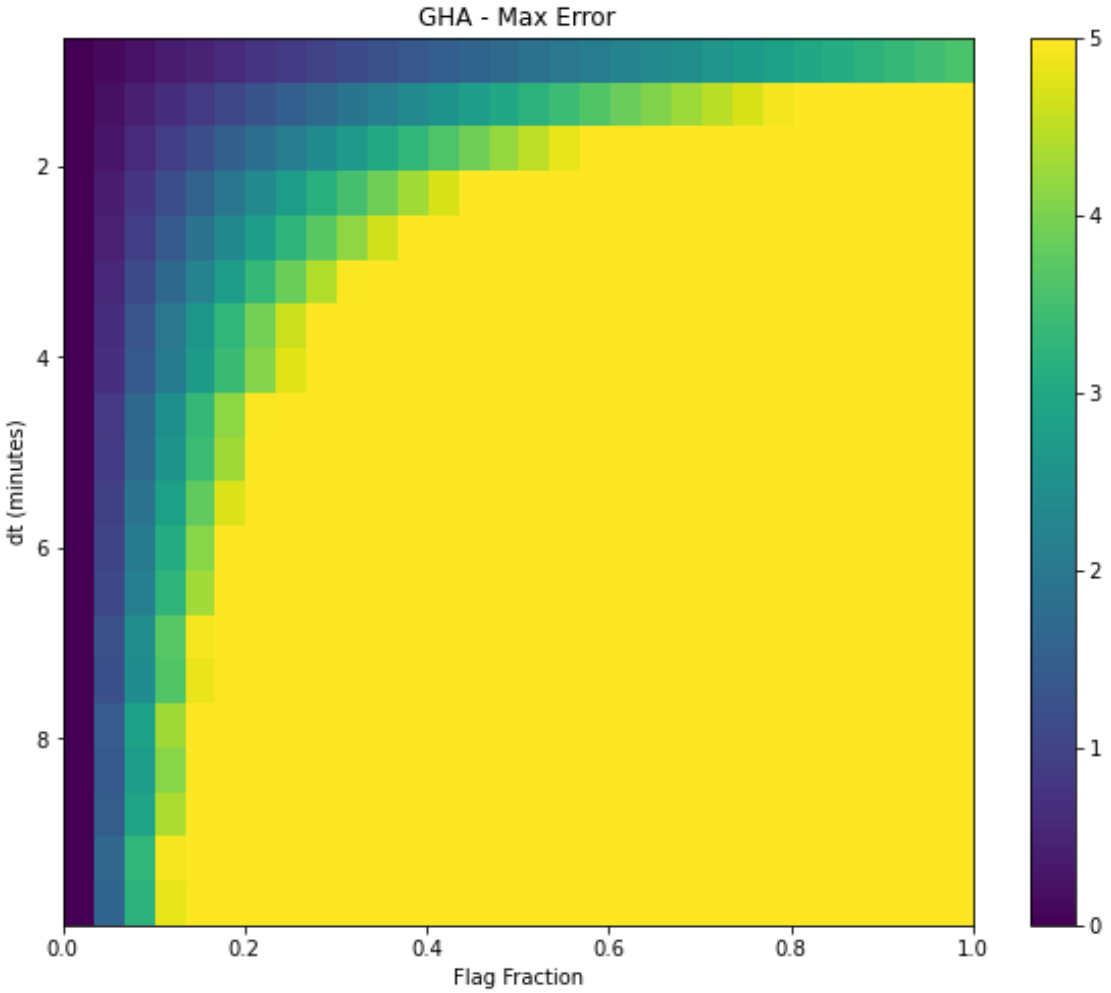
The integrals were calculated analytically. For the time bins, a range from the default edges bin size (39 seconds) to 1 hour was used. This case saw much higher error, with max error unable to get below 500mK and rms error unable to get below 300 mK at anything but the smallest bin size. Even in this case, the 39-second EDGES bin size results in an max error above 100mK for all flag fractions, and an rms error above 100 mK with all but minimal flag fractions. This was repeated with smaller bin sizes to determine whether we are able to reach the desired 10 mK values. In this case, bin sizes from 39 seconds to 10 minutes were used. Similar results were found.

3.1.2 Shorter Time Bins

Lets try this for something like 39 seconds (default minimum) to 10 minutes:

[ ]:
```
#use parameters to calculate error function
ff = np.linspace(0, 0.95, 30)
dt = np.linspace(0.01083,0.166,20) #hours (39 sec to 10 min)
```

```
# Different metrics
max_error = np.zeros((len(dt), len(ff)))
rms_error = np.zeros((len(dt), len(ff)))


for i, dtt in enumerate(dt):
    t = np.linspace(0, 24, np.int(24/dtt))
    for j, fff in enumerate(ff):
        errors = error_time(t,dtt,fff)
        max_error[i,j] = np.max(abs(errors))
        rms_error[i,j] = np.sqrt(np.mean(np.square(errors)))
```

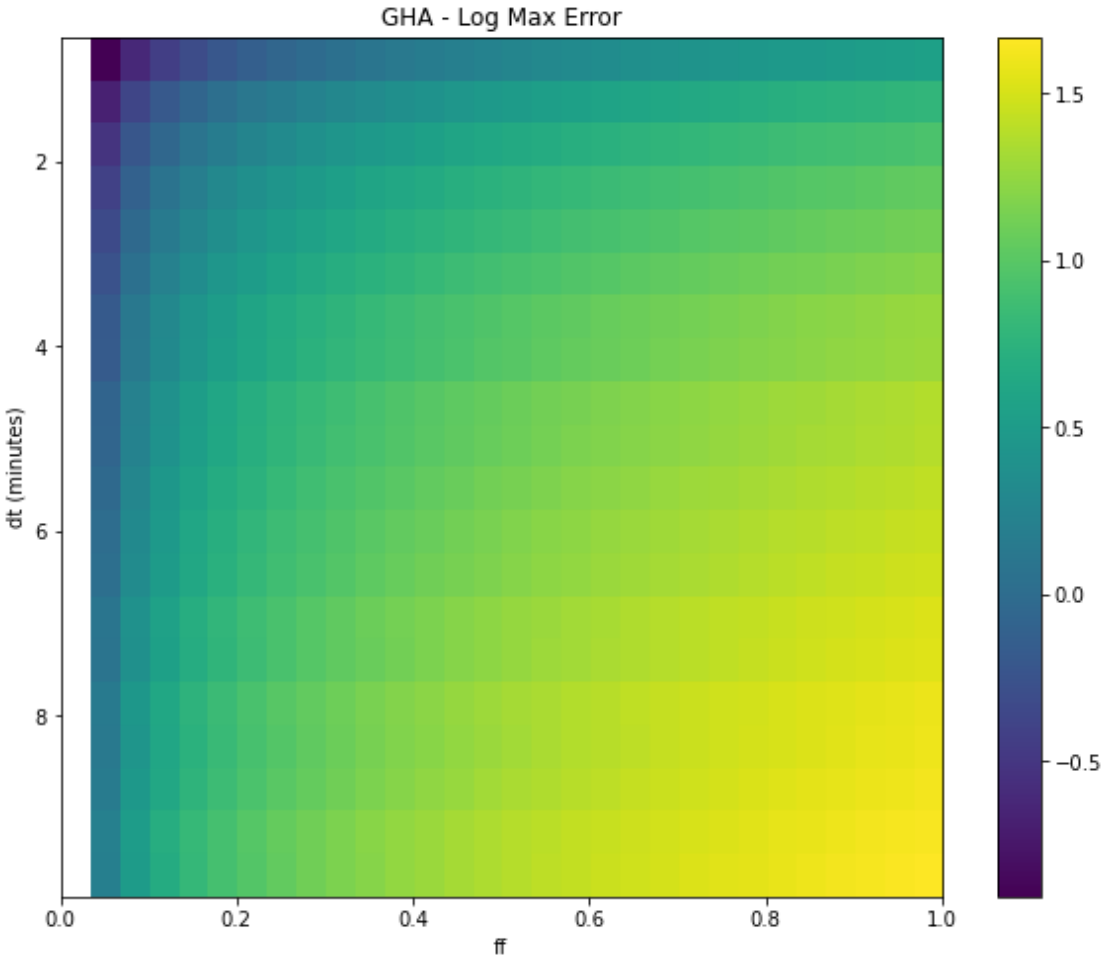


Above: Absolute Maximum Error of the 76 MHz error values, using bin sizes from 39s to 10 minutes.

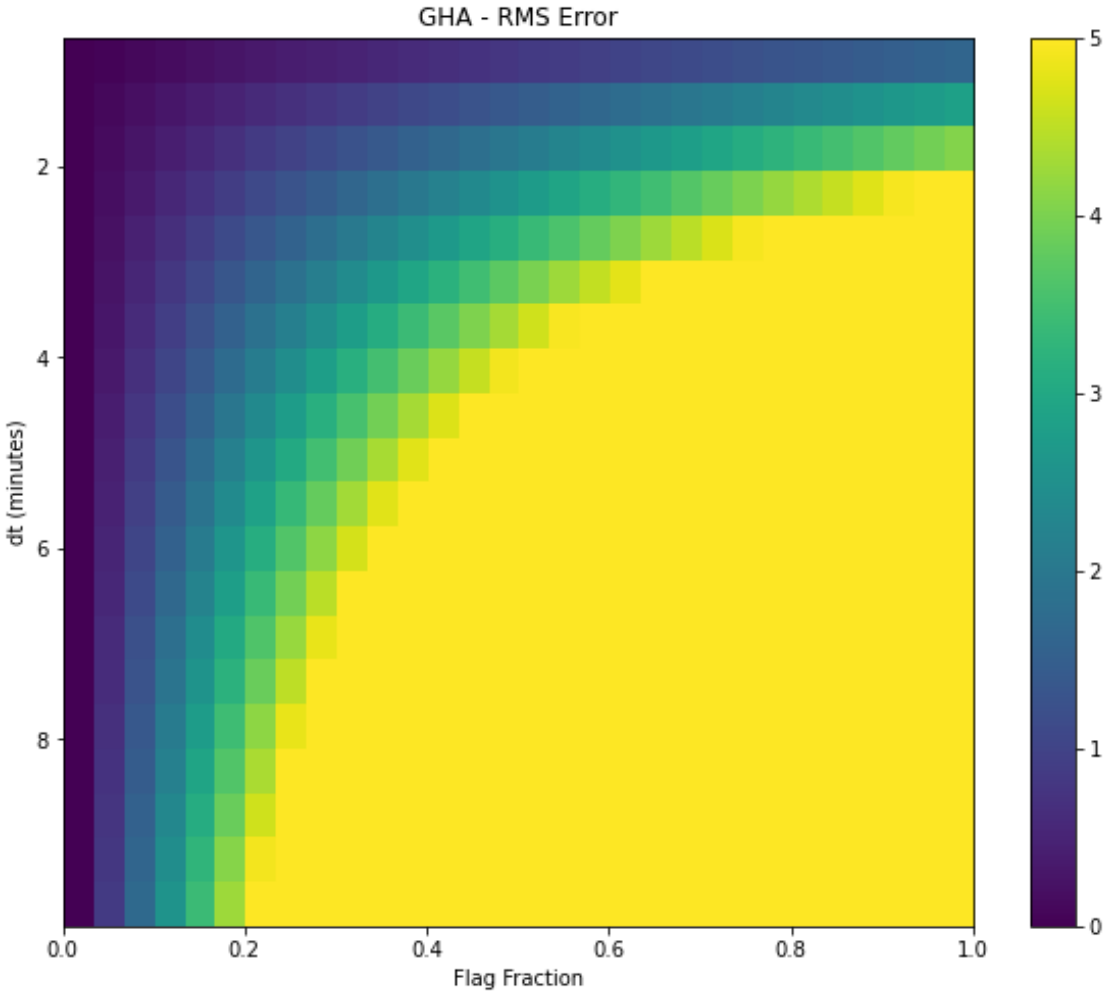
```
<ipython-input-35-e81e31b77446>:2: RuntimeWarning: divide by zero encountered in
```

log10

```
  plt.imshow(np.log10(abs(max_error)),extent=[0,1, dt[-1]*60, dt[0]*60],
aspect='auto')
```

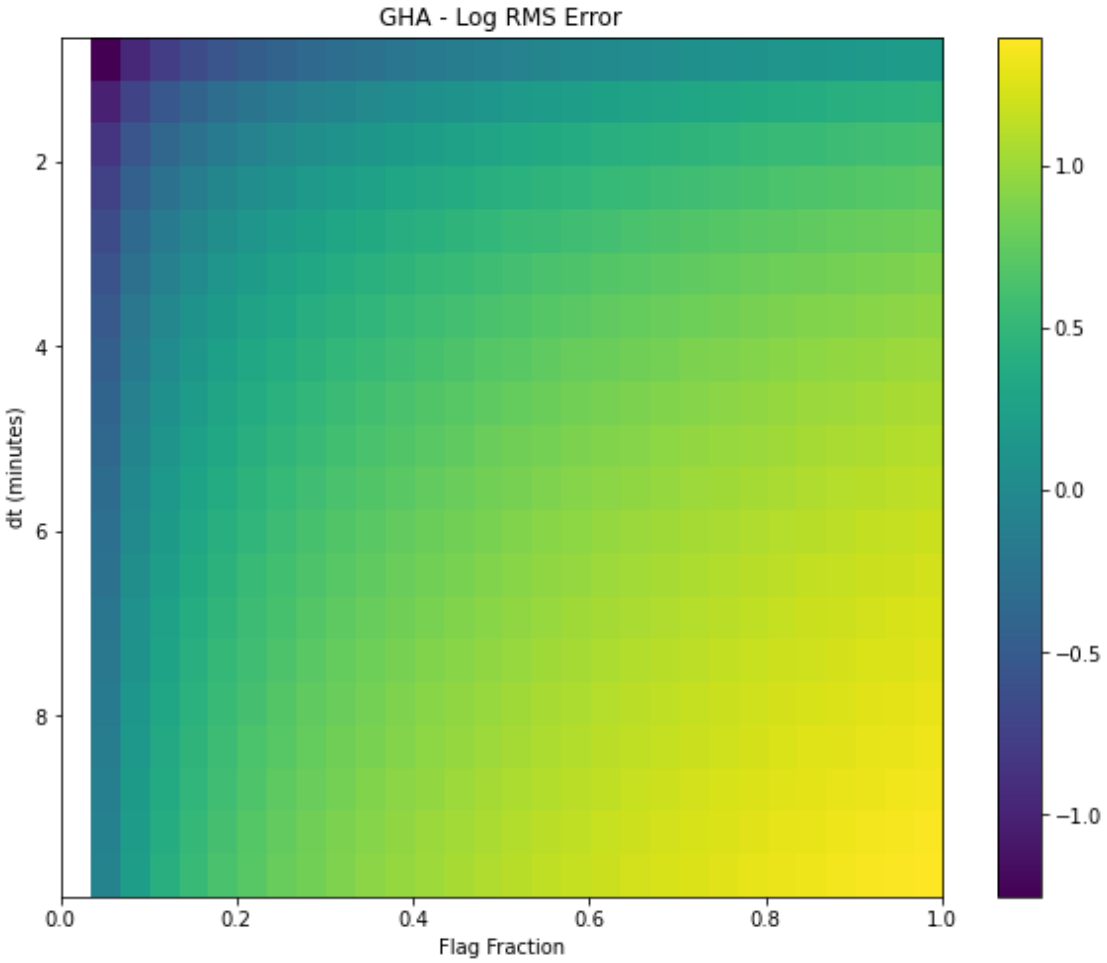


Above: Log Absolute Maximum Error of the 76 MHz error values, using bin sizes from 39s to 10 minutes.



Above: RMS Error of the 76 MHz error values, using bin sizes from 39s to 10 minutes.

```
<ipython-input-39-3e6d07c3b11b>:2: RuntimeWarning: divide by zero encountered in
log10
  plt.imshow(np.log10(abs(rms_error)), extent=[0,1, dt[-1]*60, dt[0]*60],
aspect='auto')
```



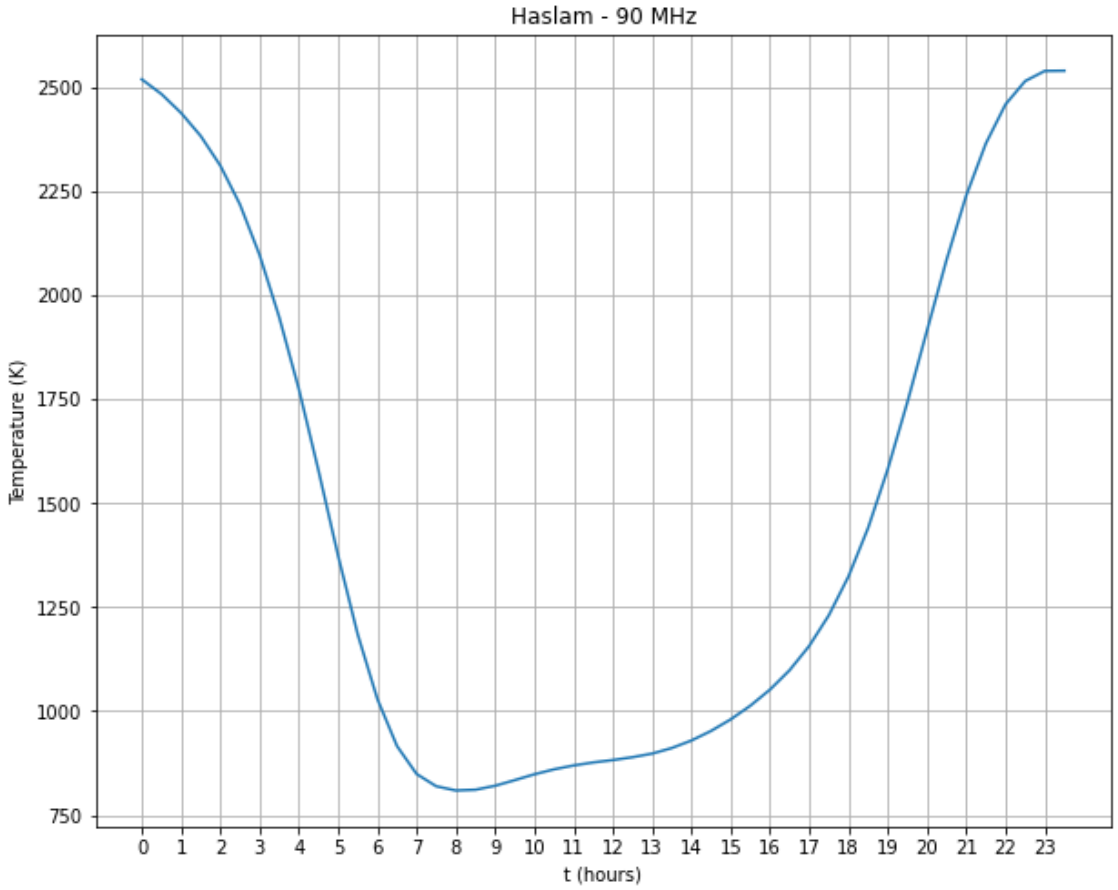
Above: Log RMS Error of the 76 MHz error values, using bin sizes from 39s to 10 minutes.

3.1.3 Analysis at 90MHz

This time binning process was repeated for 90 MHz, well within the FM band, a source of considerable noise and flagging. 90 MHz showed a slight improvement in error, with the smallest bin size and flag fraction (above 0) showing a max error below 80 mK and rms error below 60mK. However, above 3.2% flagging it also moves into errors above 100mK.

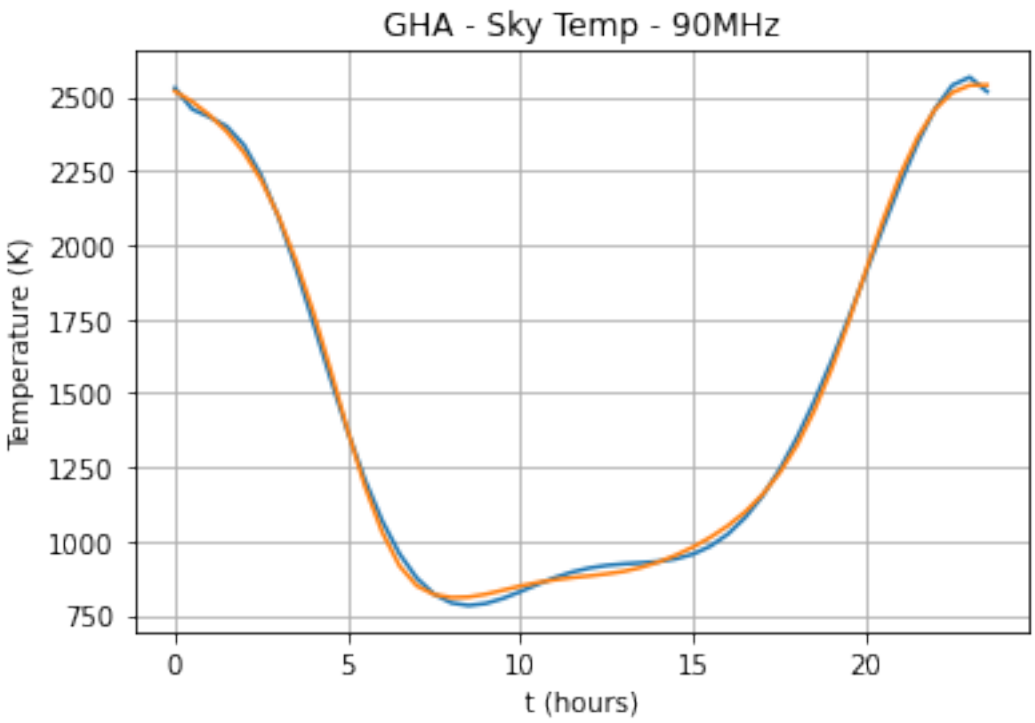
[40]:
```
freq_ind = np.argwhere(haslam_freq==90)[0][0]
```

[41]:
```
t = np.linspace(0, 24, 49)
```

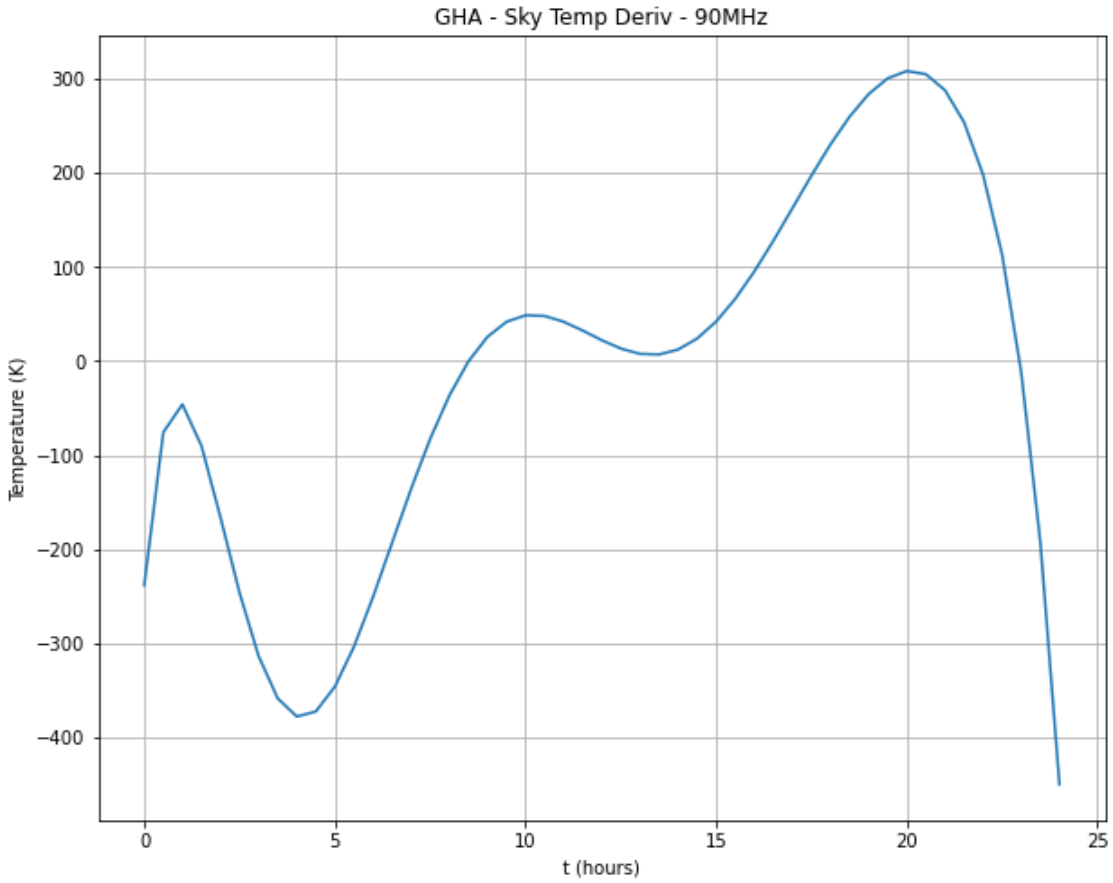


Above: Sky Temperature of the Haslam Sky Map in the EDGES Beam at 90 MHz

[43]:
```
t = np.linspace(0, 24, 49)
tfit = np.polyfit(t[:-1], haslam_temp[:,freq_ind], deg=9) #76 MHz
tt = np.poly1d(tfit)
tint=tt.integ()
tder=tt.deriv()
```



Above: Tsky 90 MHz temperature and the 9-degree model fitted to it.



Above: TSky temperature derivative at 90 MHz.

[46]:
```
tder(t).min(), tder(t).max()
```

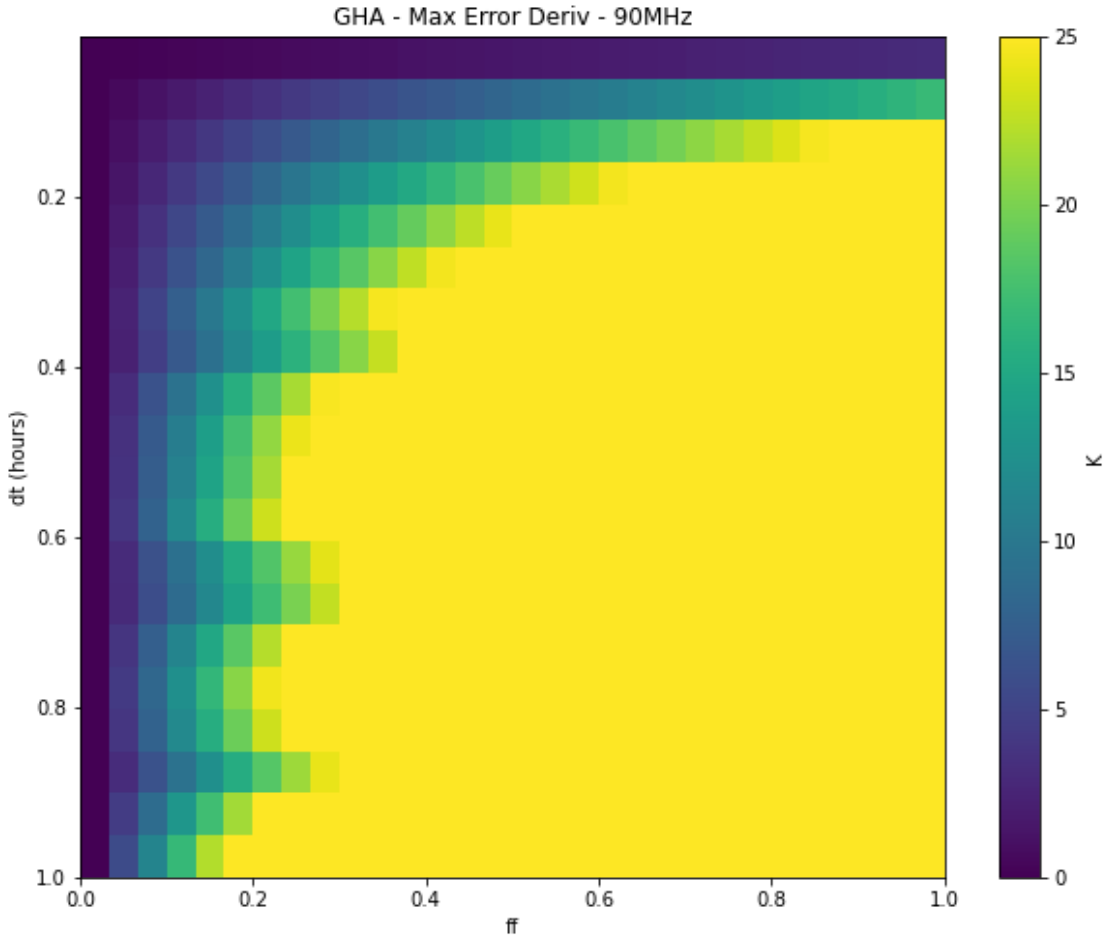
[46]:
```
(-449.94078454755515, 308.4350441691639)
```

[47]:
```
#use parameters to calculate error function
#t = np.linspace(0, 24, 48) #hours
ff = np.linspace(0, 0.95, 30)
#base_bin = 39 #seconds
dt = np.linspace(0.01083,1,20) #hours (39 sec to 1 hour)


# Different metrics
#full_error = np.zeros((len(dt), len(ff), len(t)))
max_error = np.zeros((len(dt), len(ff)))
rms_error = np.zeros((len(dt), len(ff)))


for i, dtt in enumerate(dt):
```

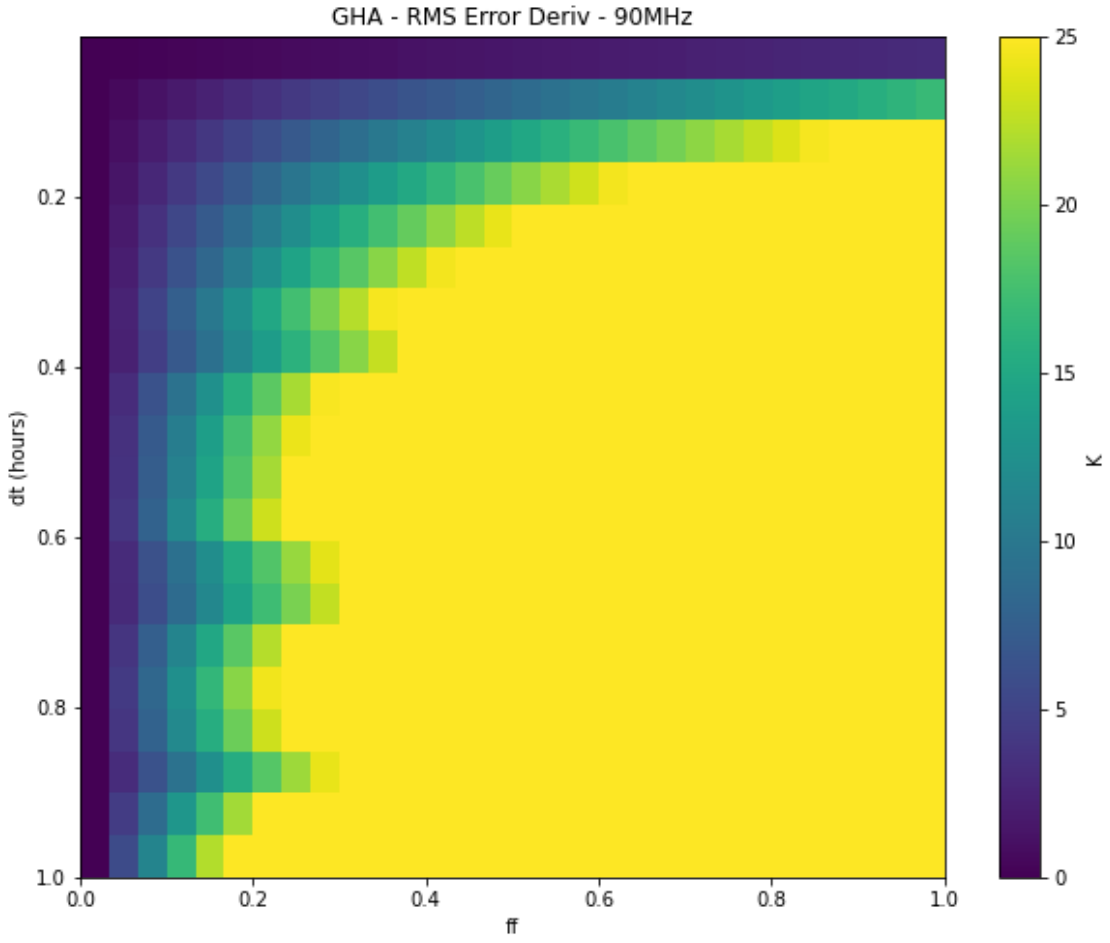
```
    t = np.linspace(0, 24, np.int(24/dtt))
    for j, fff in enumerate(ff):
        errors = error_time(t,dtt,fff)
        #full_error[i,j] = errors
        max_error[i,j] = np.max(abs(errors))
        rms_error[i,j] = np.sqrt(np.mean(np.square(errors)))
```

[48]:
```
#use parameters to calculate error function
#t = np.linspace(0, 24, 48) #hours
ff = np.linspace(0, 0.95, 30)
#base_bin = 39 #seconds
dt = np.linspace(0.01083,1,20) #hours (39 sec to 1 hour)


# Different metrics
#full_error = np.zeros((len(dt), len(ff), len(t)))
max_error_d = np.zeros((len(dt), len(ff)))
rms_error_d = np.zeros((len(dt), len(ff)))


for i, dtt in enumerate(dt):
    t = np.linspace(0, 24, np.int(24/dtt))
    errors_d = np.zeros((len(t-1)))
    for j, fff in enumerate(ff):
        errors = error_time(t,dtt,fff)
        for e, err in enumerate(errors[:-1]):
            errors_d = (errors[e+1]-err)/dtt

        #full_error[i,j] = errors
        max_error_d[i,j] = np.max(abs(errors_d))
        rms_error_d[i,j] = np.sqrt(np.mean(np.square(errors_d)))
```

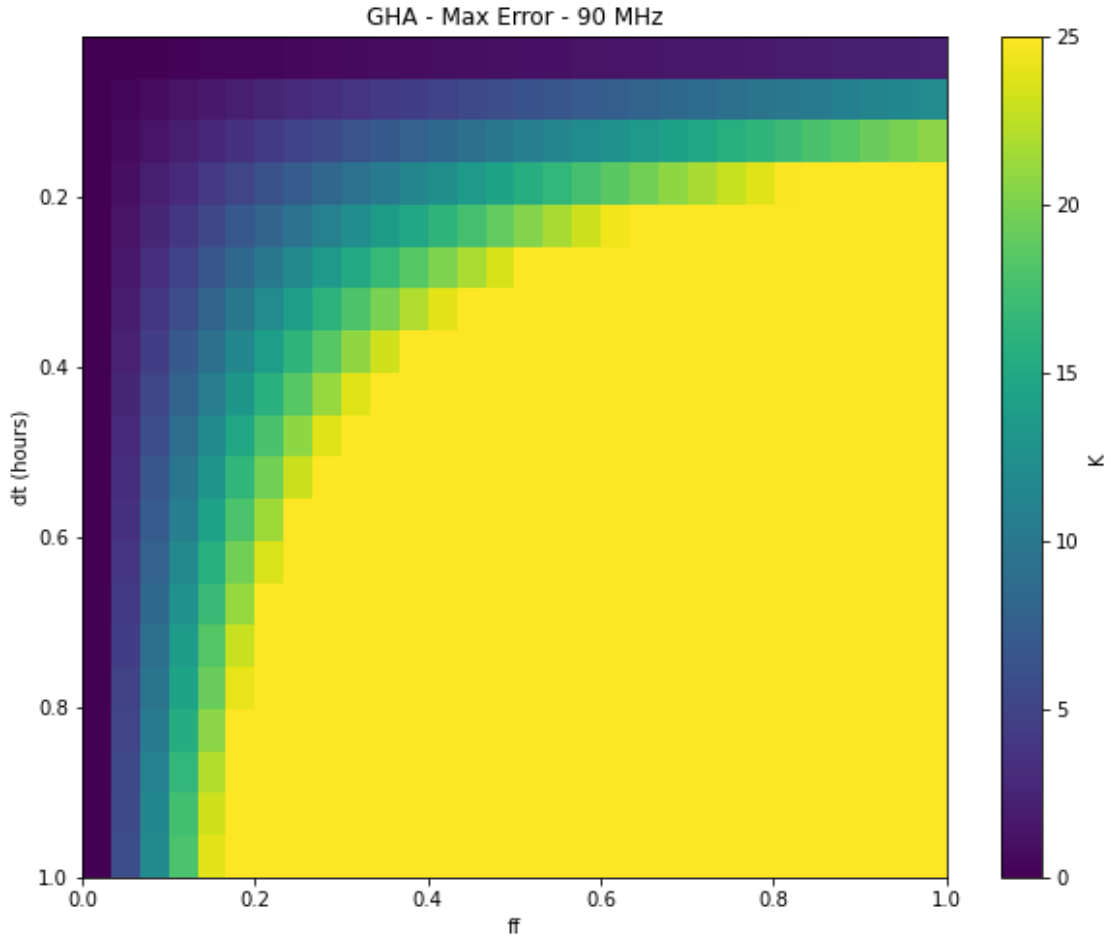
[51]:
```
plt.figure(figsize=(10,8),facecolor='w')
plt.imshow(max_error_d,extent=[0,1, dt[-1], dt[0]],vmax=25, aspect='auto')
plt.title('GHA - Max Error Deriv - 90MHz')
plt.xlabel('ff')
plt.ylabel('dt (hours)')
plt.colorbar(label="K")
plt.show()
```



Above: The maximum absolute derivative of the 90 MHz error for each bin size.

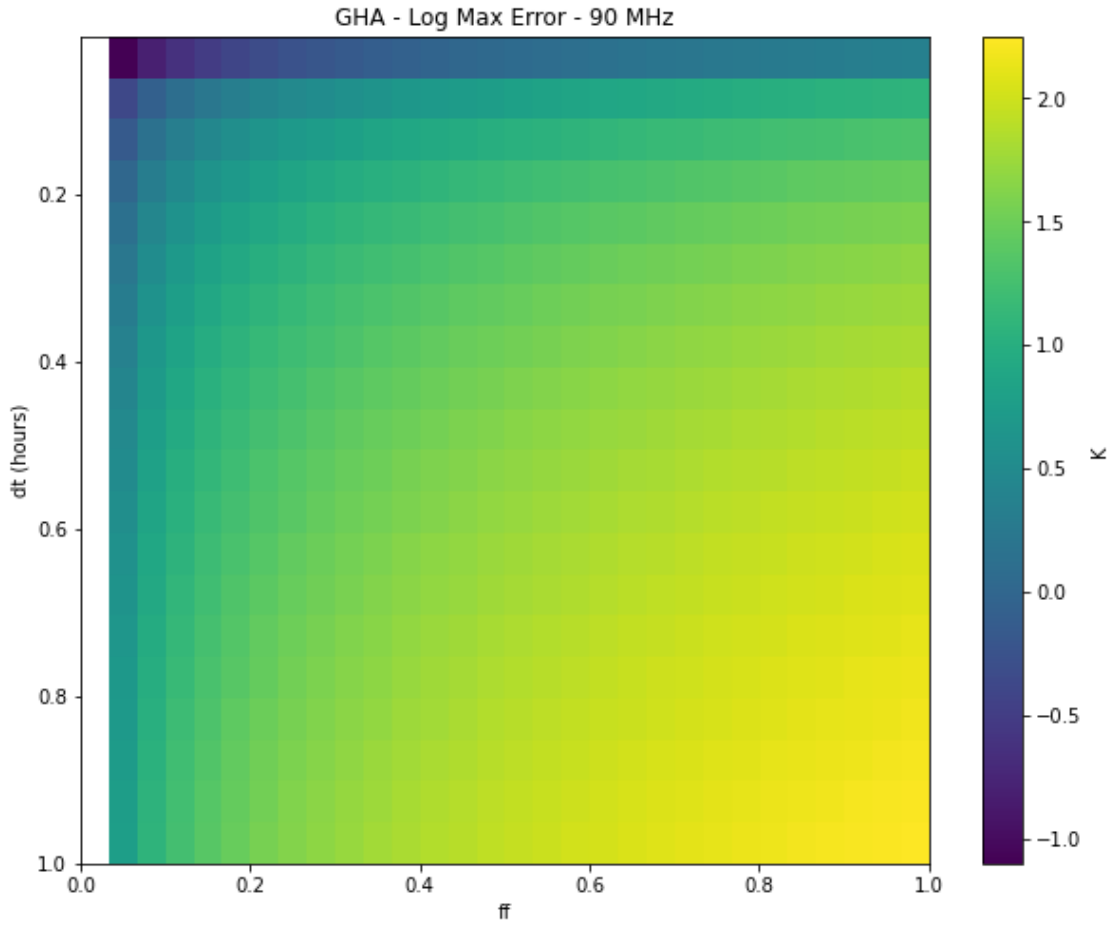


Above: The RMS absolute derivative of the 90 MHz error for each bin size.

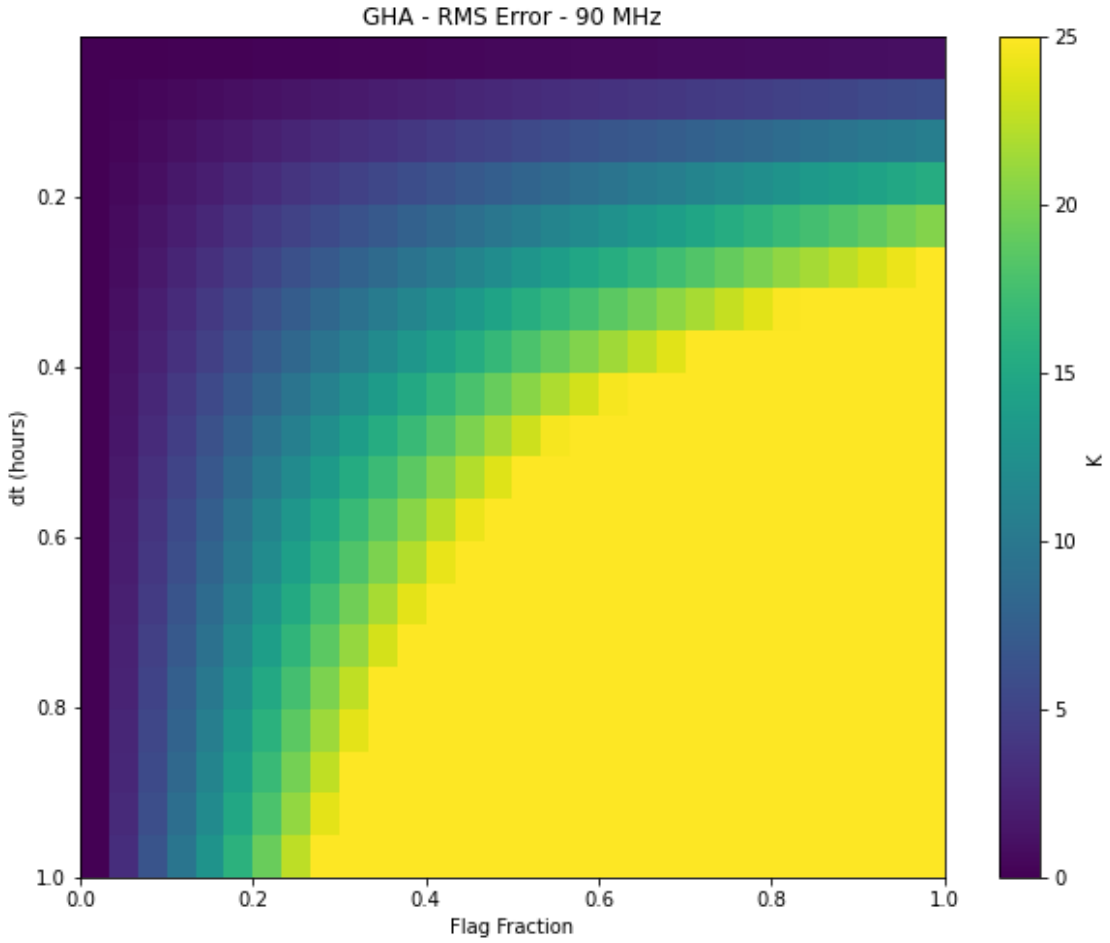


Above: Absolute Maximum Error of the 90 MHz error values.

```
<ipython-input-58-8d0f9845fee9>:2: RuntimeWarning: divide by zero encountered in
log10
  plt.imshow(np.log10(abs(max_error)),extent=[0,1, dt[-1], dt[0]],
aspect='auto')
```

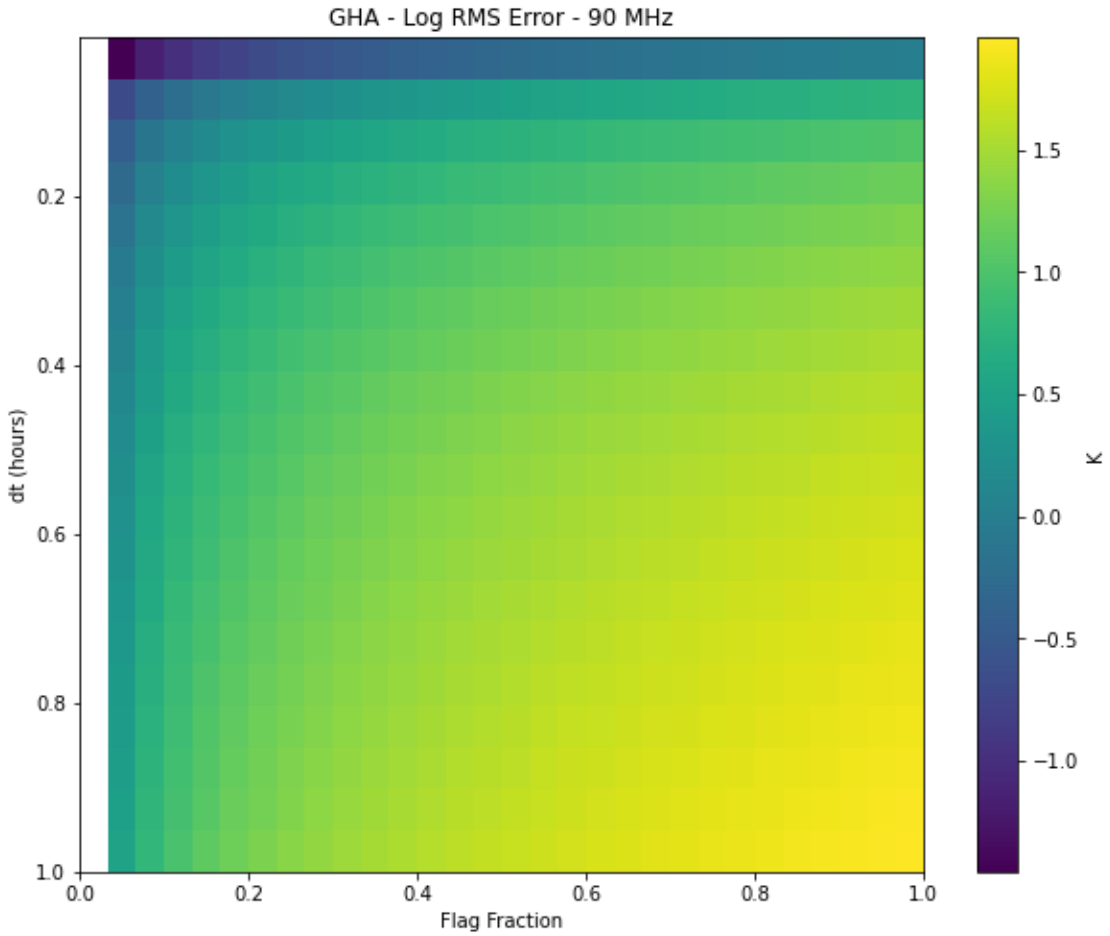


Above: Log Absolute Maximum Error of the 90 MHz error values.



Above: RMS Error of the 90 MHz error values.

```
<ipython-input-60-527ca5cfbaac>:2: RuntimeWarning: divide by zero encountered in
log10
  plt.imshow(np.log10(abs(rms_error)), extent=[0,1, dt[-1], dt[0]])
```



Above: Log RMS Error of the 90 MHz error values.

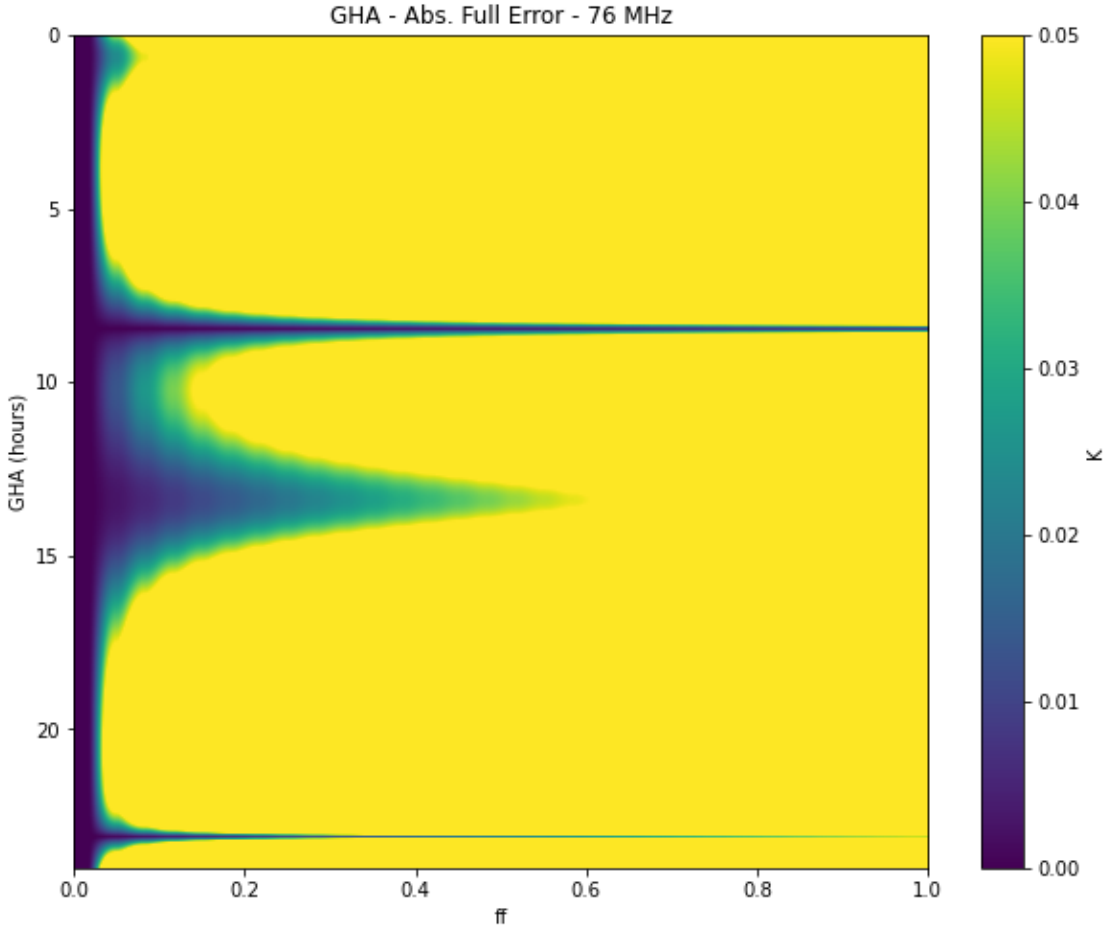
This time binning process was repeated for 90 MHz, well within the FM band, a source of considerable noise and flagging. 90 MHz showed a slight improvement in error, with the smallest bin size and flag fraction (above 0) showing a max error below 80 mK and rms error below 60mK. However, above 3.2% flagging it also moves into errors above 100mK.

3.1.4 Error change with 39s bins

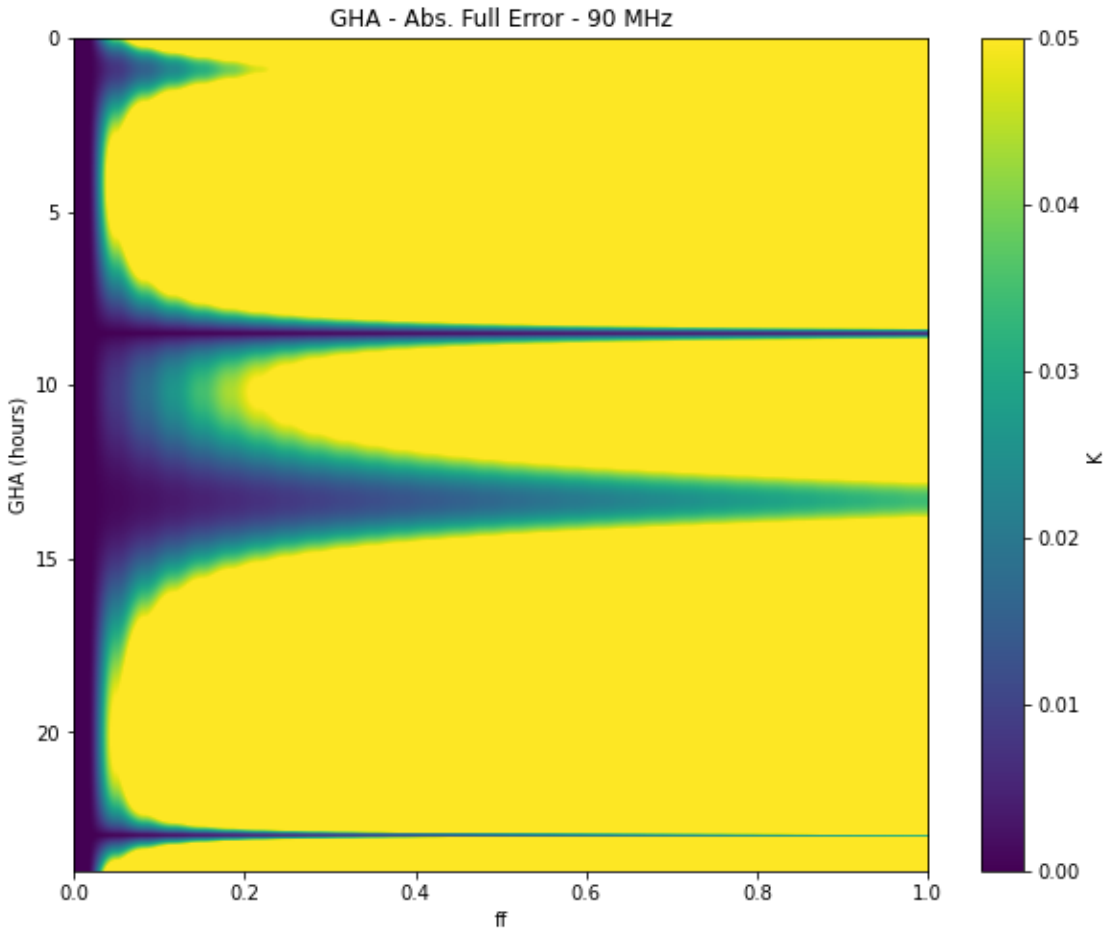
Since we are unable to substantially increase bin times without a corresponding increase in error, the error was calculated using the minimum bin size of the EDGES instrument (39 seconds) for both 76 and 90MHz.

[ ]:
```
#use parameters to calculate error function
#t = np.linspace(0, 24, 48) #hours
ff = np.linspace(0, 0.95, 30)
#base_bin = 39 #seconds
dt = 39/(60*60) #hours (39 sec)
```

```
t = np.linspace(0, 24, np.int(24/dt))
# Different metrics
#full_error = np.zeros((len(dt), len(ff), len(t)))
#max_error = np.zeros((len(t), len(ff)))
#rms_error = np.zeros((len(t), len(ff)))
full_error76 = np.zeros((len(t), len(ff)))


for j, fff in enumerate(ff):
    errors = error_time(t,dt,fff)
    for i, err in enumerate(errors):
        full_error76[i,j] = err
    #max_error[i,j] = np.max(abs(errors))
    #rms_error[i,j] = np.sqrt(np.mean(np.square(errors)))
```



Above: Absolute Error values for 76 MHz across LST. Note that only times with a slowly varying sky show low error in our default bin size.



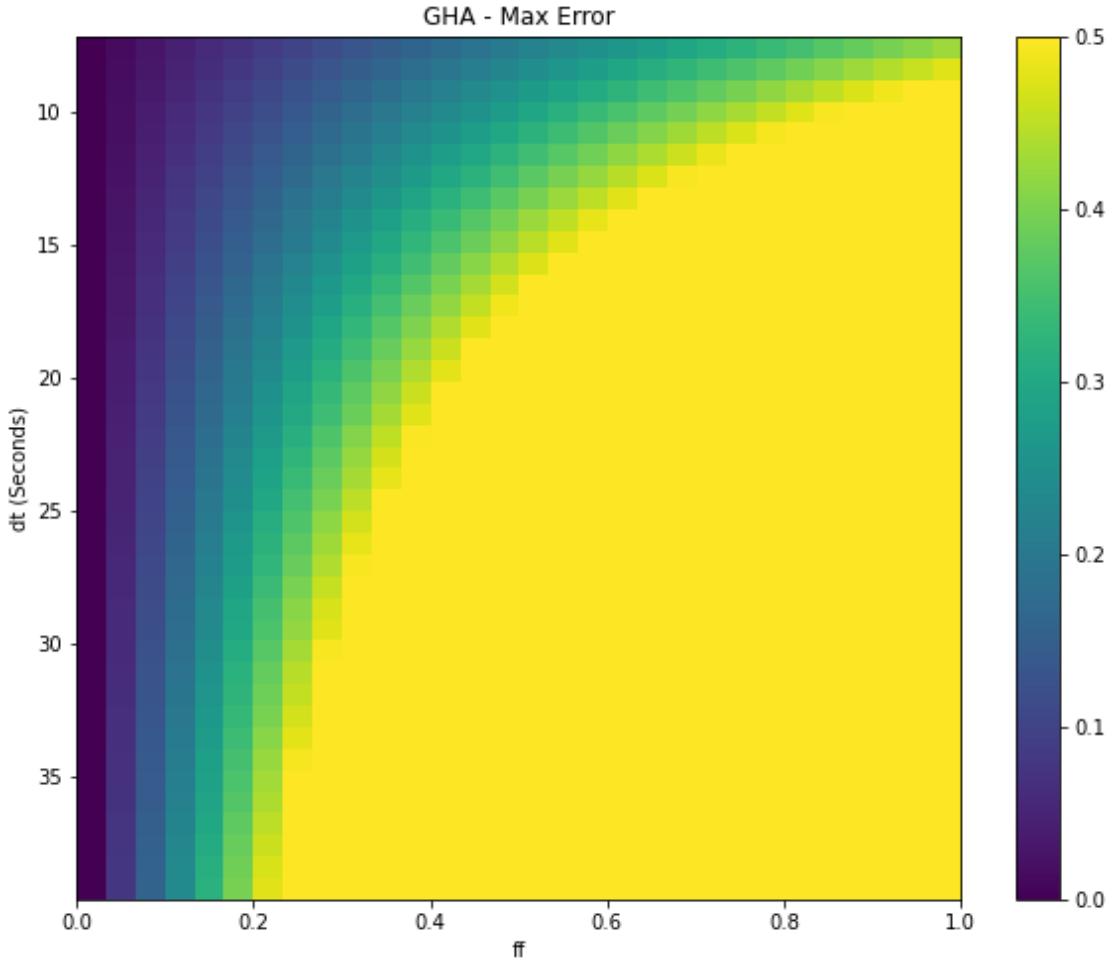
Above: Absolute Error values for 90 MHz across LST. As in the 76 MHz error, only LST with low temperature changes show desired error ranges.

3.1.5 Smallest time bins

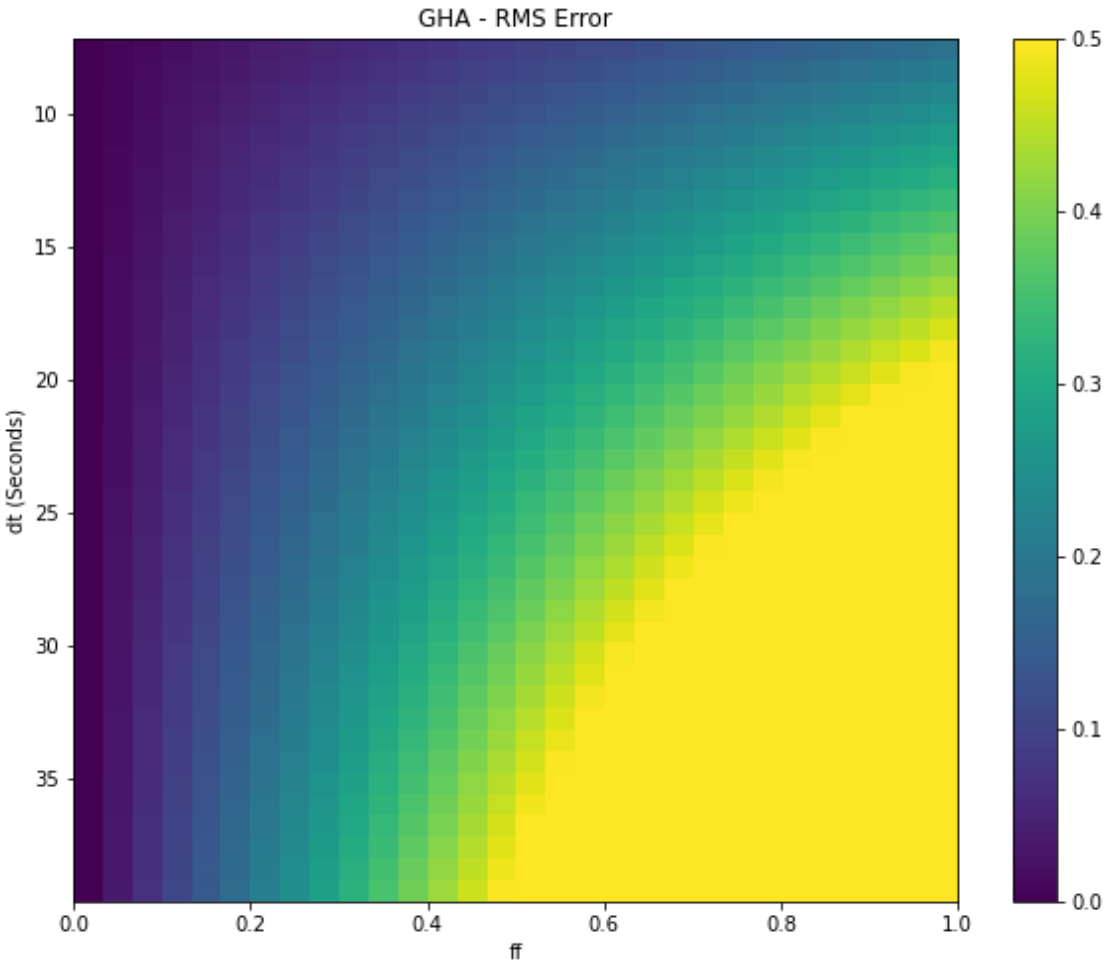
So it currently looks like we are currently unable to reach an error in the mK range. But could we? What kind of dt is required?

[69]:
```
#use parameters to calculate error function
ff = np.linspace(0, 0.95, 30)
dt = np.linspace(0.002,0.011,40) #hours (7.1s to 39s)


# Different metrics
max_error = np.zeros((len(dt), len(ff)))
rms_error = np.zeros((len(dt), len(ff)))
```

```
for i, dtt in enumerate(dt):
    t = np.linspace(0, 24, np.int(24/dtt))
    for j, fff in enumerate(ff):
        errors = error_time(t,dtt,fff)
        #full_error[i,j] = errors
        max_error[i,j] = np.max(abs(errors))
        rms_error[i,j] = np.sqrt(np.mean(np.square(errors)))
```



Above: Absolute Maximum Error for 76 MHz bins from 7.1 seconds to 39 seconds.



Above: RMS Error for 76 MHz bins from 7.1 seconds to 39 seconds

Although this is much improved from our default bin size, none of these values fall under 10mk, though a few bins in the smaller flag fractions and bin sizes come close. This suggests that to obtain a desireable error level, bin sizes must be severely reduced.

3.1.6 Theoretical Maximum Bin Size

With our above equations, we can derive the bin size required to produce errors smaller than our desired 10mK. If we assume a linear model and two consecutive bins, we find our formula is:

$$\epsilon = b/2 * ff * dt \tag{9}$$

where b is the TSky derivative between the bins, ff is the flag fraction, and dt is our bin size.

$$dt = \frac{2\epsilon}{bff} \tag{10}$$

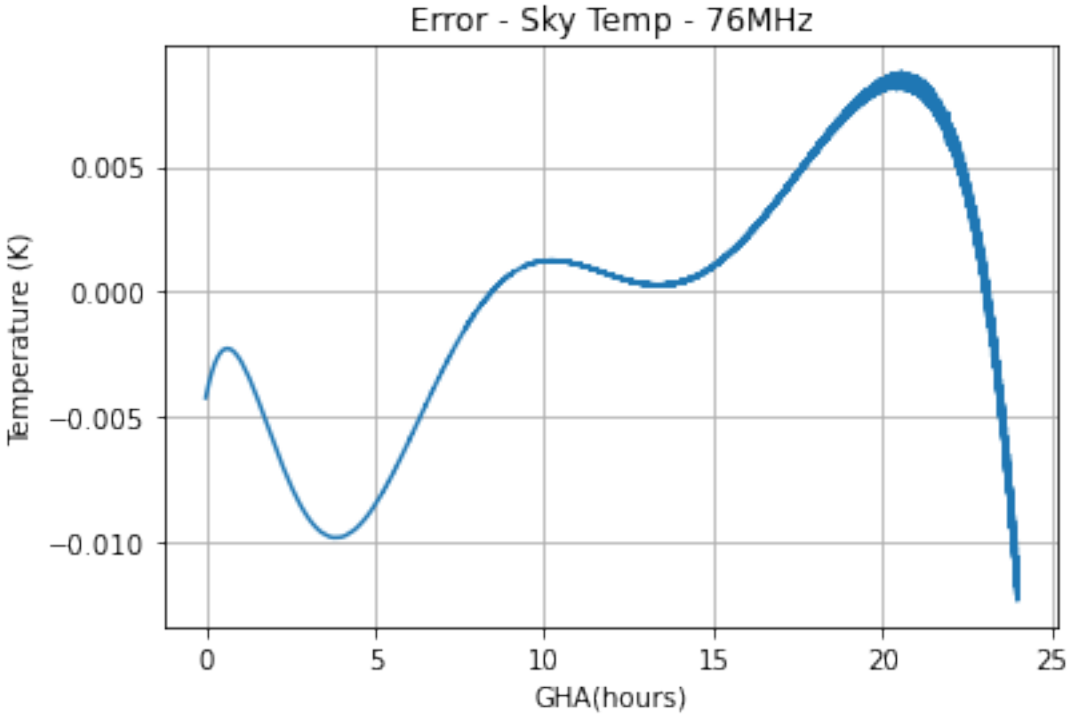
From the 76 MHz derivative seen above, the maximum change in sky temperature is approaching 600 K. Using b~600 K/h, ff= 0.5, and

$$\epsilon = 10^{-2}$$

:

$$dt = \frac{2 * 10^{-2}}{600 * 0.5} = 6.67e^{-5} hrs = 0.24s \tag{11}$$

[ ]:
```
ff=0.5
dt = (2* 10**-2)/(600*0.5) #hours (39 sec)
t = np.linspace(0, 24, np.int(24/dt))
errors = error_time(t,dt,ff)
full_error28=errors
```



Above: Error for 76MHz with 0.24s bin sizes. Note that we remain within 10mK error in all places except approaching the discontinuity.

We can see above that with 0.24 s bin size, we finally keep our error under 10mK. This is much smaller than our current default bin size of 39 seconds, and a reduction of bin size is not currently feasible with current experimental equipment.

4 Conclusion

In conclusion, we created an error model to analyze the possible error due to biasing that may be present due to the binning of our data into larger bin sizes. We found that while the biasing error due to frequency binning is below 10 mK at all frequency bin sizes below 2 kHz, with errors rising above this threshold at higher flagging fractions using frequency bins larger than this threshold. Additionally, lower frequencies show higher error values due to higher temperature changes. The error using the same method for time binning is considerably larger, with nearly all flag fractions producing error above 100 mK. This remains true both within and outside of the FM band. This suggests that we will need to remain in our default smallest time and frequency, and also that other methods of debiasing will likely be needed during our analysis.