averaging_with_weights

October 12, 2020

1 Averaging non-stationary functions with weights.

Steven Murray, on behalf of the EDGES collaboration (Oct. 2020).

1.1 Introduction

To average a set of data d_i, each with a weight $w_i = 1/\sigma_i^2$, we usually use a simple weighted average:

$$\bar{d} = \frac{\sum_i^N d_i w_i}{\sum_i^N w_i}. \tag{1}$$

Supposing that d_i are some random values of a *stationary process*, i.e. $d_i = \mu + \mathcal{N}(0, \sigma_i^2)$, this is ideal, since

$$\langle \bar{d} \rangle = \frac{\mu \sum w_i}{\sum w_i} = \mu. \tag{2}$$

(i.e., this is an unbiased estimate of μ, the "true average" in the data). Furthermore of course it is easy to show that the variance of such an estimator reduces as

$$\mathrm{Var}(\bar{d}) = \frac{1}{\sum 1/\sigma_i^2}, \tag{3}$$

i.e. the variance of the estimated mean is mostly dependent on the samples with *least variance*, and so the total variance is typically lower than $\bar{\sigma^2}/N$. Note that these formulas work just as well for "flagged" data, for which $\sigma \to \infty$.

1.2 The Problem

In EDGES spectra (as a function of GHA and Frequency), we *do not have* stationary processes. Within a single bin that we wish to average, there is not a single μ, rather a moving function $f(x)$ (x could be frequency or GHA).

So, let our data be $d_i = f(x_i) + n_i$. In the absence of noise, the average should be

$$\mu = \sum_i f(x_i)/N. \tag{4}$$

If we however perform a weighted average:

$$\langle \bar{d} \rangle = \left\langle \frac{\sum (f(x_i) + n_i) w_i}{\sum w_i} \right\rangle = \frac{\sum f(x_i) w_i}{\sum w_i} \neq \mu. \tag{5}$$

Here again we see that if $f(x_i) = c$ (i.e. a constant), then we get the right answer back. Similarly, if the weight function is a constant, we get the right answer. However, if the weight function is for example proportional to f, we get back

$$\langle \bar{d} \rangle = \frac{\sum f^2(x_i)}{\sum f(x_i)} \neq \mu. \tag{6}$$

1.3 A Solution

This problem can be resolved if we are able to determine a function, $f'(x)$, which models $f(x)$. This is of course never truly possible, but if we get reasonably close, it can be used like this:

$$\bar{d} = \frac{\sum f'(x_i)}{N} + \frac{\sum (d_i - f'(x_i)) w_i}{\sum w_i}. \tag{7}$$

If $f' = f$, then this has

$$\langle \bar{d} \rangle = \frac{\sum f'(x_i)}{N} = \mu, \tag{8}$$

as we require (since the second term consists only of random noise about zero).

Furthermore, the variance of this estimator is

$$\mathrm{Var}(\bar{d}) = \frac{1}{\sum 1/\sigma_i^2}, \tag{9}$$

as we would like. Notice again that this is true for flagged data as well.

Note that this is unbiased *only* if the fit itself is unbiased (at each x). It is hard to ascertain how true this is. However, it is reasonably simple perhaps to obtain an estimate of how much uncertainty is introduced by performing the fit, by estimating the covariance about the result at each x_i from the fit. I am not yet sure how to combine this with the noise purely obtained from the estimator.

Note that biases introduced here will not average down with more data, and should be considered carefully in the full averaged estimate.

1.4 Numerical Comparisons

Numpy offers a simple weighted average (for stationary data), and we've implemented a non-stationary average in `edges-analysis`. In this section, we compare results for these two approaches, for examples of increasing complexity. For all these examples, we'll be taking the case of a spectrum over frequency with some weights/flags.

[1]:
```
import numpy as np
from edges_analysis.analysis import tools

import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
f = np.linspace(50, 100, 400)
bins = np.linspace(50,100, 100)

centres = (bins[:-1]+ bins[1:])/2
```

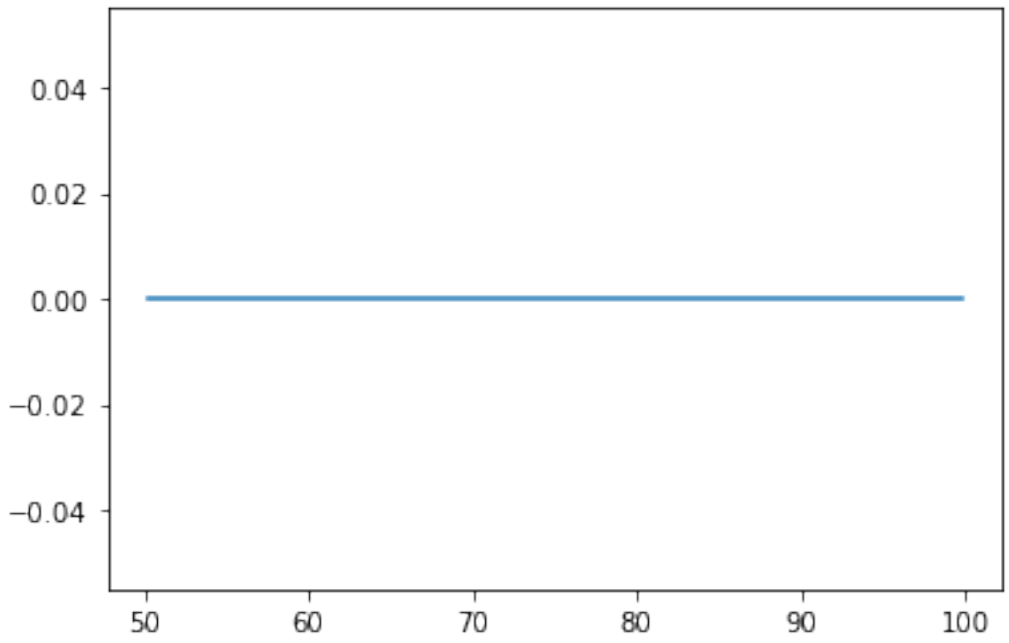
[3]:
```
def naive_weighted_bin(data, x, bins, weights=None):
    out = np.zeros(len(bins)-1)

    if weights is None:
        weights = np.ones_like(data)

    for i in range(len(out)):
        mask = (x >= bins[i]) & (x < bins[i+1])
        if np.any(weights[mask] > 0):
            out[i] = np.average(data[mask], weights=weights[mask])
        else:
            out[i] = np.nan

    return out
```

1.4.1 No Noise, No Weights, Power-Law

[4]:
```
spec_form = lambda f : f**-2.5
```

[5]:
```
spec = spec_form(f)

w_avg = naive_weighted_bin(spec, f, bins)
ns_w_avg = tools.non_stationary_bin_avg(spec, f, bins=bins, model_fit=spec_form)
```

[6]:
```
plt.plot(centres, w_avg-ns_w_avg);
```



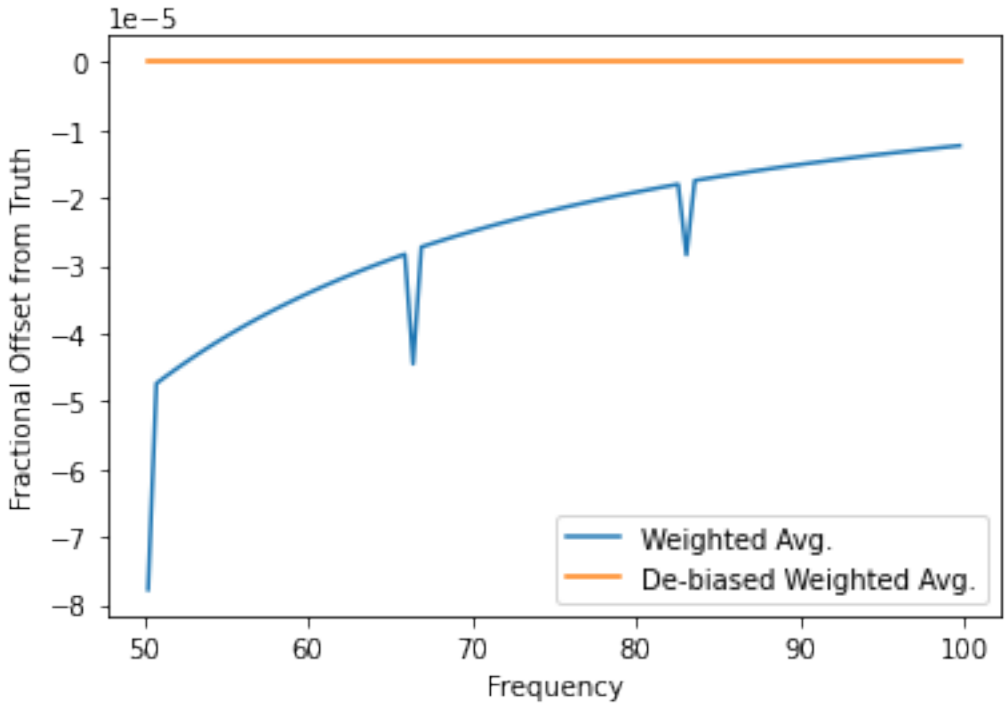
As we expected, if there's no noise and no weights, and we model the function perfectly, we get exactly the same result.

1.4.2 No Noise, Weights

[7]:
```
weights = 1/spec

w_avg = naive_weighted_bin(spec, f, bins, weights=weights)
ns_w_avg = tools.non_stationary_bin_avg(spec, f, bins=bins,␣
 ↪model_fit=spec_form, weights=weights)

true = naive_weighted_bin(spec, f, bins)
```

[8]:
```
plt.plot(centres, (w_avg - true) / spec_form(centres), label="Weighted Avg.")
plt.plot(centres, (ns_w_avg - true) / spec_form(centres), label='De-biased␣
 ↪Weighted Avg.')
plt.legend()
plt.xlabel('Frequency')
plt.ylabel("Fractional Offset from Truth")
```

[8]:
```
Text(0, 0.5, 'Fractional Offset from Truth')
```

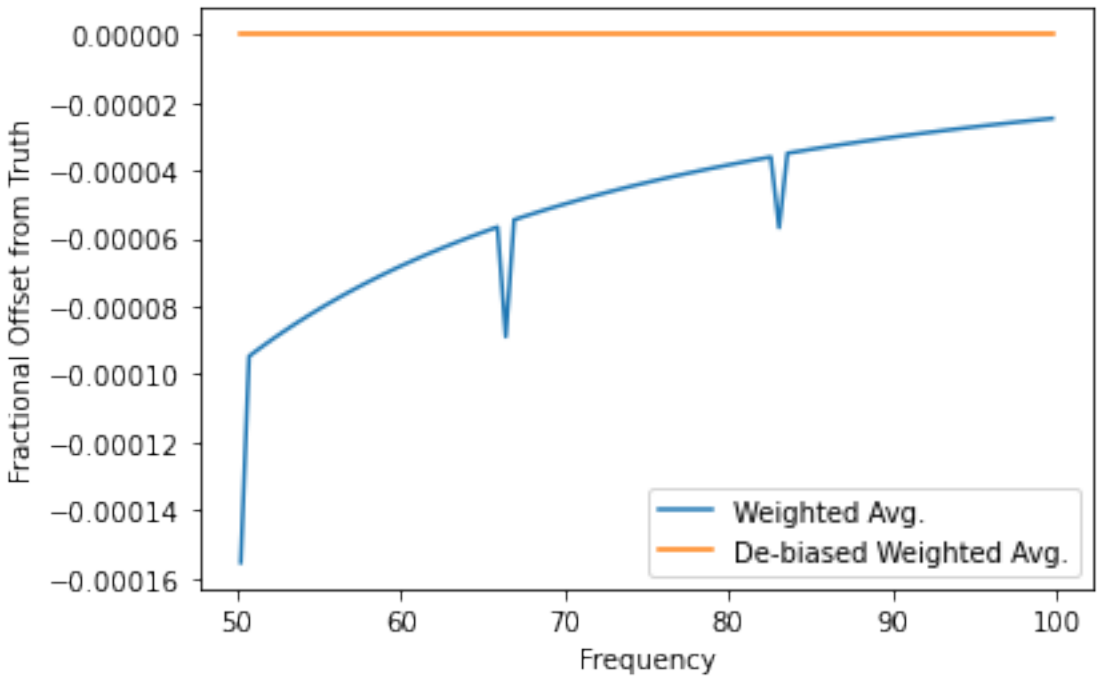


Here we see that the simple weighted averaged gets a very small offset from the true answer for this binning scheme.

[9]:
```
weights = 1/spec**2

w_avg = naive_weighted_bin(spec, f, bins, weights=weights)
ns_w_avg = tools.non_stationary_bin_avg(spec, f, bins=bins,␣
 ↪model_fit=spec_form, weights=weights)

true = naive_weighted_bin(spec, f, bins)
```

[10]:
```
plt.plot(centres, (w_avg - true) / spec_form(centres), label="Weighted Avg.")
plt.plot(centres, (ns_w_avg - true) / spec_form(centres), label='De-biased␣
 ↪Weighted Avg.')
plt.legend()
plt.xlabel('Frequency')
plt.ylabel("Fractional Offset from Truth")
```

[10]:
```
Text(0, 0.5, 'Fractional Offset from Truth')
```



The offset from the truth gets a bit worse when the weights are proportional to $1/T^2$, which happens to be about what we expect.

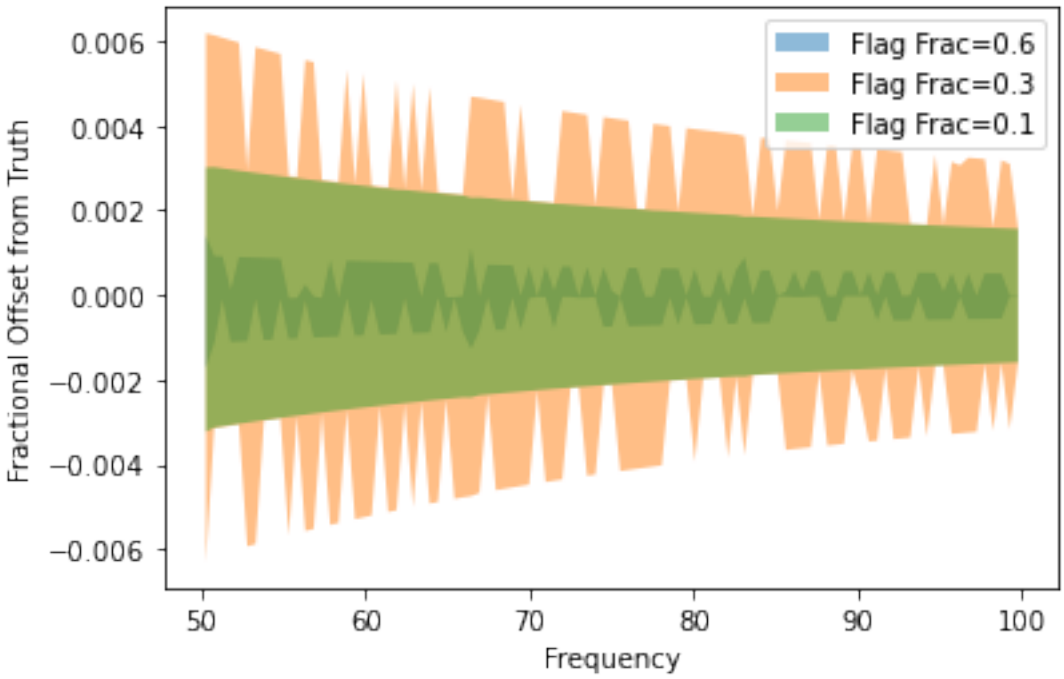
1.5 Flag-type weights, No Noise

By definition here, since the input model is perfect, we expect that the de-biased binning will still have a discrepancy of zero (since there is no noise). We omit it in this case.

[11]:
```
true = naive_weighted_bin(spec, f, bins)

ntrials = 500
flag_fracs = [0.1, 0.3, 0.6]
w_avg = np.zeros((len(flag_fracs), ntrials, len(bins)-1))
 #ns_w_avg = np.zeros((ntrials, len(bins)-1))

weights = 1/spec**2

for j, flag_frac in enumerate(flag_fracs):
    for i in range(ntrials):
        flags = np.random.binomial(1, flag_frac, size=len(f)).astype(bool)
        wght = np.where(flags, 0, weights)

        w_avg[j, i] = naive_weighted_bin(spec, f, bins, weights=wght)
```

[12]:
```
w_avg_perc = np.nanpercentile((w_avg - true) / spec_form(centres), [16, 84],␣
↪axis=1)
#ns_w_avg_perc = np.nanpercentile((ns_w_avg - true) / spec_form(centres), [16,␣
↪84], axis=0)

for j, ff in enumerate(flag_fracs[::-1]):
    plt.fill_between(centres, w_avg_perc[0][-j], w_avg_perc[1][-j],␣
↪label=f"Flag Frac={ff}", alpha=0.5)
#plt.fill_between(centres, ns_w_avg_perc[0], ns_w_avg_perc[1], label='De-biased␣
↪Weighted Avg.', alpha=0.5)
plt.legend()
plt.xlabel('Frequency')
plt.ylabel("Fractional Offset from Truth")
```

[12]:
```
Text(0, 0.5, 'Fractional Offset from Truth')
```



Thus there can be a bias of up to ~0.6% for flagging fractions of ~30% if the non-debiased averaging is used (for 0.5 MHz bins).

1.5.1 Noise and flag-type weights

[13]:
```
true = naive_weighted_bin(spec, f, bins)

ntrials = 500
flag_fracs = [0.1, 0.3, 0.6]
w_avg = np.zeros((len(flag_fracs), ntrials, len(bins)-1))
ns_w_avg = np.zeros((len(flag_fracs), ntrials, len(bins)-1))

weights = 1/spec**2

for j, flag_frac in enumerate(flag_fracs):
    for i in range(ntrials):
        flags = np.random.binomial(1, flag_frac, size=len(f)).astype(bool)
        noise = np.random.normal(0, scale=1/np.sqrt(weights)/100)
        wght = np.where(flags, 0, weights)

        w_avg[j, i] = naive_weighted_bin(spec + noise, f, bins, weights=wght)
        ns_w_avg[j, i] = tools.non_stationary_bin_avg(spec + noise, f,␣
↪bins=bins, model_fit=spec_form, weights=wght)
```

```
/data4/smurray/Projects/radio/EOR/Edges/edges-
analysis/src/edges_analysis/analysis/tools.py:99: RuntimeWarning: invalid value
encountered in double_scalars
  out[indx] = np.mean(m) + np.sum(res * this_weight) / np.sum(this_weight)
```

[14]:
```
w_avg_perc[0][0] - ns_w_avg_perc[0][0]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-14-81f9382f5360> in <module>
----> 1 w_avg_perc[0][0] - ns_w_avg_perc[0][0]

NameError: name 'ns_w_avg_perc' is not defined
```

[ ]:
```
fix, ax = plt.subplots(1, len(flag_fracs), sharex=True, figsize=(18, 6))

w_avg_perc = np.nanpercentile((w_avg - true) / spec_form(centres), [16, 84],␣
↪axis=1)
ns_w_avg_perc = np.nanpercentile((ns_w_avg - true) / spec_form(centres), [16,␣
↪84], axis=1)

for j, ff in enumerate(flag_fracs[::-1]):

    ax[j].fill_between(centres, w_avg_perc[0][-j], w_avg_perc[1][-j],␣
↪label=f"Weighted Avg.", alpha=0.5)
```

```
    ax[j].fill_between(centres, ns_w_avg_perc[0][-j], ns_w_avg_perc[1][-j],␣
↪label=f"De-Biased", alpha=0.5)
    ax[j].set_xlabel('Frequency')
    ax[j].set_ylabel("Fractional Offset from Truth")

ax[0].legend()
```

Here, the added noise dominates the fractional offsets, making the two procedures essentially identical (though this will depend on the relative signal-to-noise in the bin). As the signal-to-noise increases in the bin, we expect the simple weighted averaged to settle on a biased fractional offset.

1.6 Non-perfect model

All the above has assumed we can fit a perfect model to the spectrum in the first place. Let's relax that assumption:

[ ]:
```
true = naive_weighted_bin(spec, f, bins)

ntrials = 500
flag_fracs = [0.1, 0.3, 0.6]
w_avg = np.zeros((len(flag_fracs), ntrials, len(bins)-1))
ns_w_avg = np.zeros((len(flag_fracs), ntrials, len(bins)-1))

weights = 1/spec**2

for j, flag_frac in enumerate(flag_fracs):
    for i in range(ntrials):
        flags = np.random.binomial(1, flag_frac, size=len(f)).astype(bool)
        noise = np.random.normal(0, scale=1/np.sqrt(weights)/100)
        wght = np.where(flags, 0, weights)

        w_avg[j, i] = naive_weighted_bin(spec + noise, f, bins, weights=wght)
        ns_w_avg[j, i] = tools.non_stationary_bin_avg(spec + noise, f,␣
↪bins=bins, model='polynomial', weights=wght, n_terms=5)
```

[ ]:
```
fig, ax = plt.subplots(1, len(flag_fracs), sharex=True, figsize=(18, 6))

w_avg_perc = np.nanpercentile((w_avg - true) / spec_form(centres), [16, 84],␣
↪axis=1)
ns_w_avg_perc = np.nanpercentile((ns_w_avg - true) / spec_form(centres), [16,␣
↪84], axis=1)

for j, ff in enumerate(flag_fracs[::-1]):

    ax[j].fill_between(centres, w_avg_perc[0][-j], w_avg_perc[1][-j],␣
↪label=f"Weighted Avg.", alpha=0.5)
```

```
    ax[j].fill_between(centres, ns_w_avg_perc[0][-j], ns_w_avg_perc[1][-j],␣
↪label=f"De-Biased", alpha=0.5)
    ax[j].set_xlabel('Frequency')
    ax[j].set_ylabel("Fractional Offset from Truth")

ax[0].legend()
```

For a smooth intrinsic model with large sample noise, the de-biased still does better. But let's make the model a bit worse:

[ ]:
```
complex_form = lambda f : (f/75)**-2.5  +  (f/75)**-4.5
complex_spec = complex_form(f)
true = naive_weighted_bin(complex_spec, f, bins)

ntrials = 500
flag_fracs = [0.1, 0.3, 0.6]
w_avg = np.zeros((len(flag_fracs), ntrials, len(bins)-1))
ns_w_avg = np.zeros((len(flag_fracs), ntrials, len(bins)-1))

weights = 1/spec**2

for j, flag_frac in enumerate(flag_fracs):
    for i in range(ntrials):
        flags = np.random.binomial(1, flag_frac, size=len(f)).astype(bool)
        noise = np.random.normal(0, scale=1/np.sqrt(weights)/100)
        wght = np.where(flags, 0, weights)

        w_avg[j, i] = naive_weighted_bin(complex_spec + noise, f, bins,␣
↪weights=wght)
        ns_w_avg[j, i] = tools.non_stationary_bin_avg(complex_spec + noise, f,␣
↪bins=bins, model='polynomial', weights=wght, n_terms=3)
```

[ ]:
```
fig, ax = plt.subplots(1, len(flag_fracs), sharex=True, figsize=(18, 6))

w_avg_perc = np.nanpercentile((w_avg - true) / complex_form(centres), [16, 84],␣
↪axis=1)
ns_w_avg_perc = np.nanpercentile((ns_w_avg - true) / complex_form(centres),␣
↪[16, 84], axis=1)

for j, ff in enumerate(flag_fracs):

    ax[j].fill_between(centres, w_avg_perc[0][j], w_avg_perc[1][j],␣
↪label=f"Weighted Avg.", alpha=0.5)
    ax[j].fill_between(centres, ns_w_avg_perc[0][j], ns_w_avg_perc[1][j],␣
↪label=f"De-Biased", alpha=0.5)
    ax[j].set_xlabel('Frequency')
    ax[j].set_ylabel("Fractional Offset from Truth")
```

```
ax[0].legend()
```

Here we see that the model doesn't quite capture the underlying model, and towards the band edges, the fractional offset can become quite large (larger than the simple weighted average). We can probably account for this by using a per-bin-model:

[ ]:
```
complex_form = lambda f : (f/75)**-2.5  +  (f/75)**-4.5
complex_spec = complex_form(f)
true = naive_weighted_bin(complex_spec, f, bins)

ntrials = 500
flag_fracs = [0.1, 0.3, 0.6]
w_avg = np.zeros((len(flag_fracs), ntrials, len(bins)-1))
ns_w_avg = np.zeros((len(flag_fracs), ntrials, len(bins)-1))

weights = 1/spec**2

for j, flag_frac in enumerate(flag_fracs):
    for i in range(ntrials):
        flags = np.random.binomial(1, flag_frac, size=len(f)).astype(bool)
        noise = np.random.normal(0, scale=1/np.sqrt(weights)/100)
        wght = np.where(flags, 0, weights)

        w_avg[j, i] = naive_weighted_bin(complex_spec + noise, f, bins,␣
 ↪weights=wght)
        ns_w_avg[j, i] = tools.non_stationary_bin_avg(complex_spec + noise, f,␣
 ↪bins=bins, model='polynomial', weights=wght, n_terms=3, per_bin_model=True)
```

[ ]:
```
fig, ax = plt.subplots(1, len(flag_fracs), sharex=True, figsize=(18, 6))

w_avg_perc = np.nanpercentile((w_avg - true) / complex_form(centres), [16, 84],␣
 ↪axis=1)
ns_w_avg_perc = np.nanpercentile((ns_w_avg - true) / complex_form(centres),␣
 ↪[16, 84], axis=1)

for j, ff in enumerate(flag_fracs):

    ax[j].fill_between(centres, w_avg_perc[0][j], w_avg_perc[1][j],␣
 ↪label=f"Weighted Avg.", alpha=0.5)
    ax[j].fill_between(centres, ns_w_avg_perc[0][j], ns_w_avg_perc[1][j],␣
 ↪label=f"De-Biased", alpha=0.5)
    ax[j].set_xlabel('Frequency')
    ax[j].set_ylabel("Fractional Offset from Truth")

ax[0].legend()
```

This clearly reduces the amount of bias.