prior_montecarlo

May 10, 2019

1 Run and process the prior monte carlo and pick a "truth" realization

A great advantage of exploring a synthetic model is that we can enforce a "truth" and then evaluate how our various attempts to estimate it perform. One way to do this is to run a monte carlo ensemble of multiple parameter realizations and then choose one of them to represent the "truth". That will be accomplished in this notebook.

```
In [1]: import os
        import shutil
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import matplotlib as mpl
        plt.rcParams['font.size']=12
        import flopy
        import pyemu
```

```
flopy is installed in /Users/jeremyw/Dev/gw1876/activities_2day_mfm/notebooks/flopy
```

1.0.1 set the `t_d` or "template directory" variable to point at the template folder and read in the PEST control file

```
In [2]: t_d = "template"
        pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
```

1.0.2 Decide what pars are uncertain in the truth

We need to decide what our truth looks like - should the pilot points or the grid-scale pars be the source of spatial variability? or both?

```
In [3]: par = pst.parameter_data
        # grid pars
        #should_fix = par.loc[par.pargp.apply(lambda x: "gr" in x),"parnme"]
        # pp pars
        #should_fix = par.loc[par.pargp.apply(lambda x: "pp" in x),"parnme"]
        #pst.npar - should_fix.shape[0]
```

```
In [4]: pe = pyemu.ParameterEnsemble.from_binary(pst=pst,filename=os.path.join(t_d,"prior.jcb")
        #pe.loc[:,should_fix] = 1.0
        pe.to_csv(os.path.join(t_d,"sweep_in.csv"))
```

```
new binary format detected...
```

1.0.3 run the prior ensemble in parallel locally

This takes advantage of the program `pestpp-swp` which runs a parameter sweep through a set of parameters. By default, `pestpp-swp` reads in the ensemble from a file called `sweep_in.csv` which in this case we made just above.

```
In [5]: m_d = "master_prior_sweep"
        pyemu.os_utils.start_slaves(t_d,"pestpp-swp","freyberg.pst",num_slaves=20,slave_root="
```

1.0.4 Load the output ensemble and plot a few things

```
In [6]: obs_df = pd.read_csv(os.path.join(m_d,"sweep_out.csv"),index_col=0)
        print('number of realization in the ensemble before dropping: ' + str(obs_df.shape[0]))
```

```
number of realization in the ensemble before dropping: 200
```

drop any failed runs

```
In [7]: obs_df = obs_df.loc[obs_df.failed_flag==0,:]
        print('number of realization in the ensemble **after** dropping: ' + str(obs_df.shape[
```

```
number of realization in the ensemble **after** dropping: 200
```

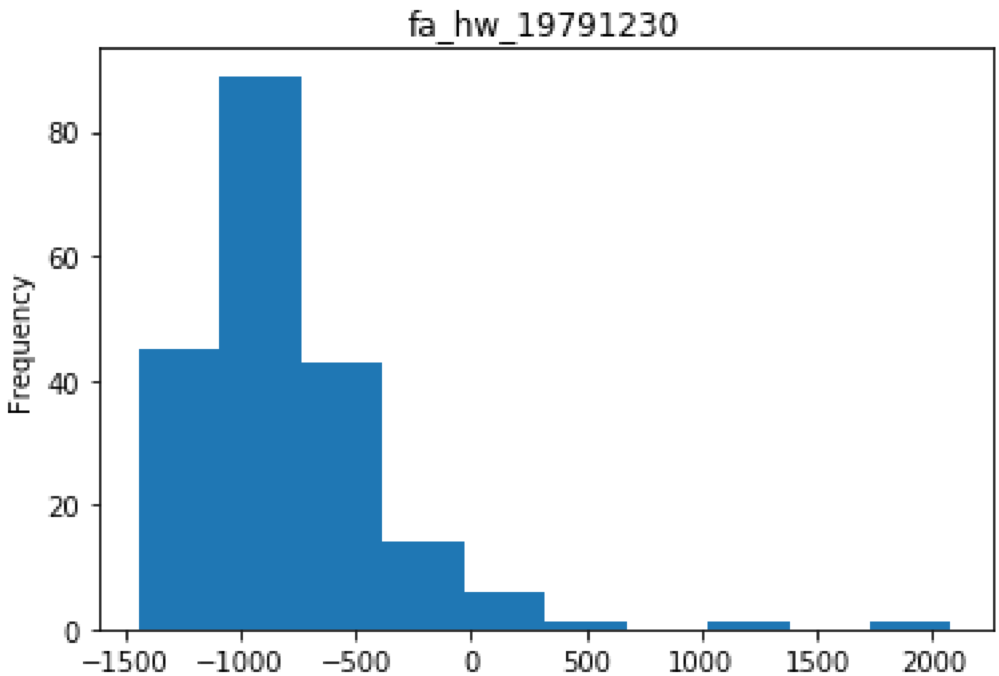
1.0.5 confirm which quantities were identified as forecasts

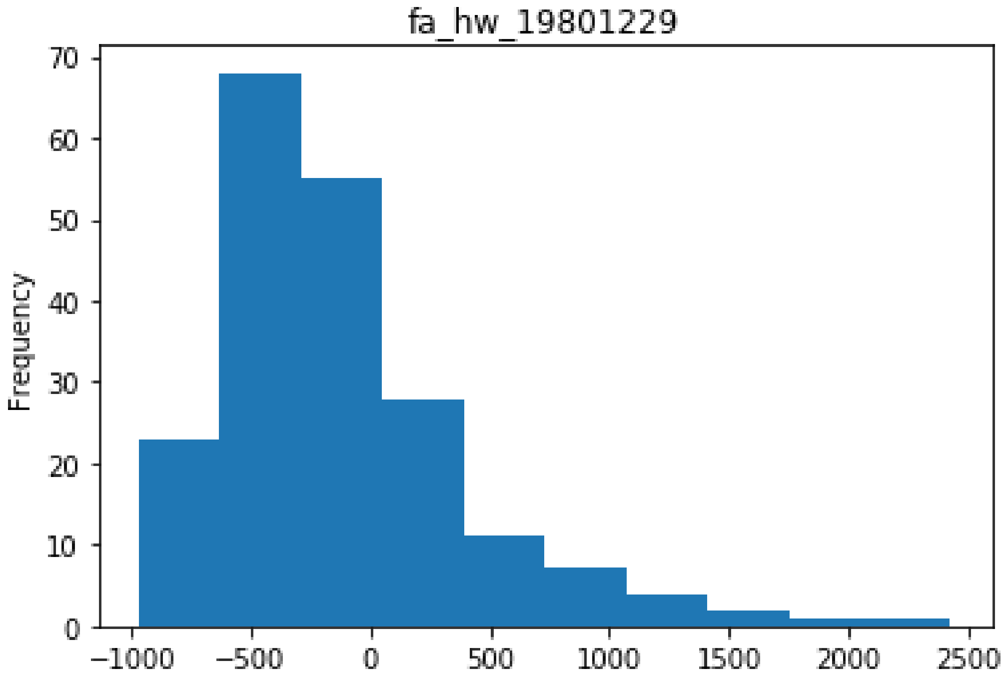
```
In [8]: fnames = pst.pestpp_options["forecasts"].split(',')
        fnames
```

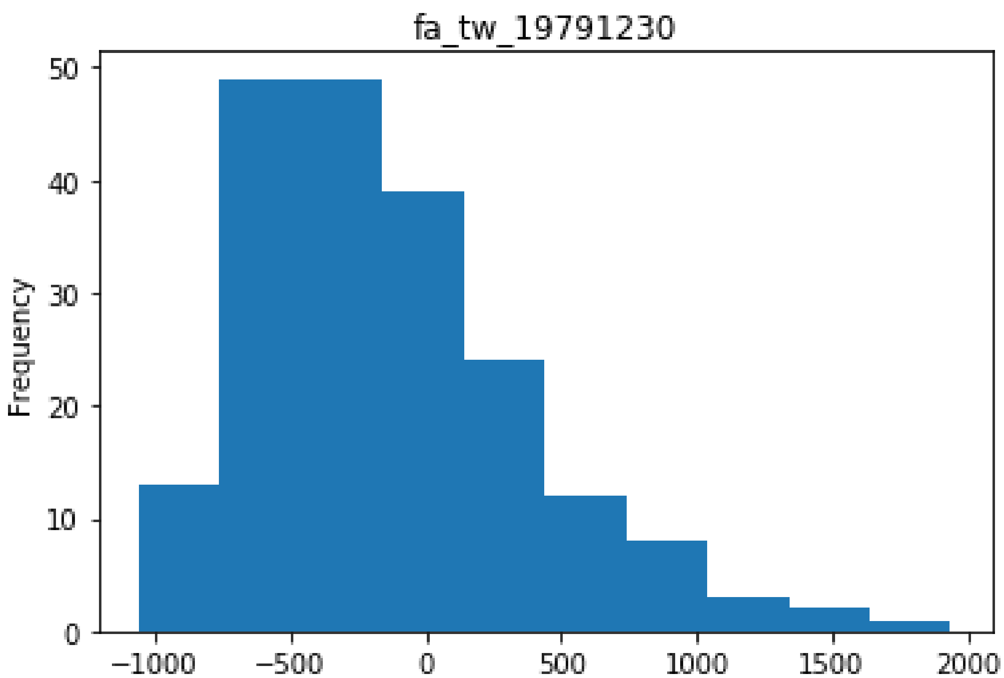
```
Out[8]: ['fa_hw_19791230',
         'fa_hw_19801229',
         'fa_tw_19791230',
         'fa_tw_19801229',
         'hds_00_013_002_000',
         'hds_00_013_002_001',
         'part_time',
         'part_status']
```

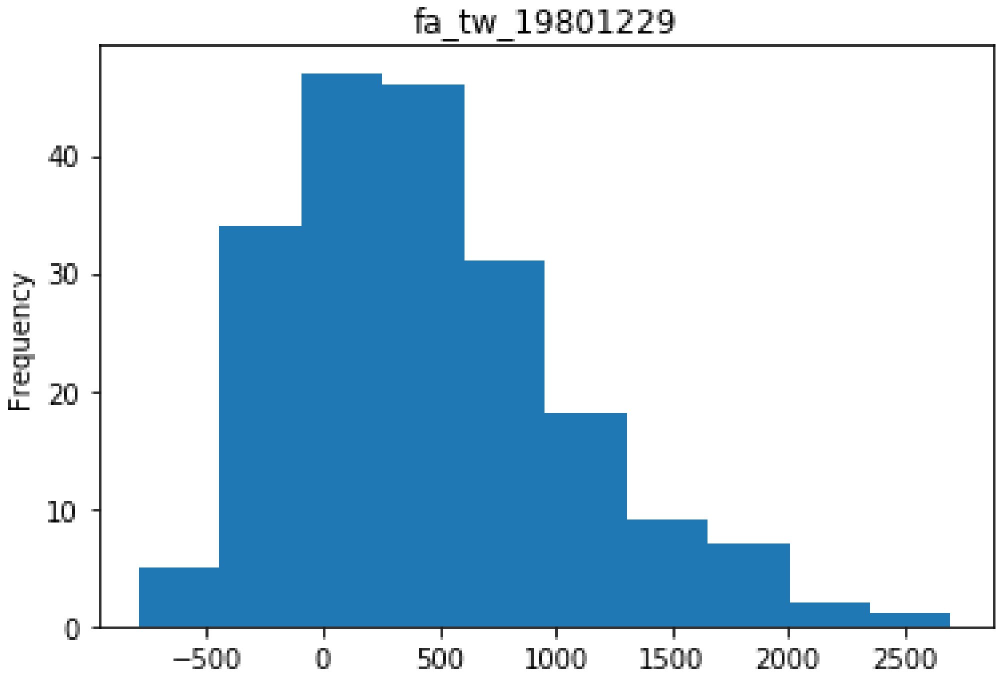
1.0.6 now we can plot the distributions of each forecast

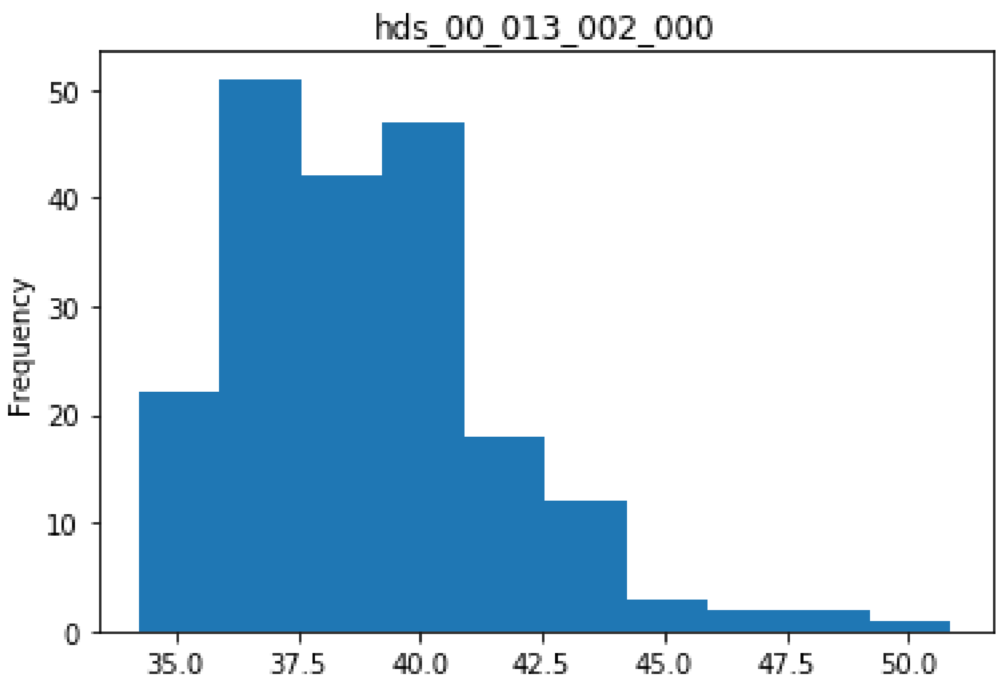
```
In [9]: for forecast in fnames:
            plt.figure()
            ax = obs_df.loc[:,forecast].plot(kind="hist")
            ax.set_title(forecast)
        plt.show()
```

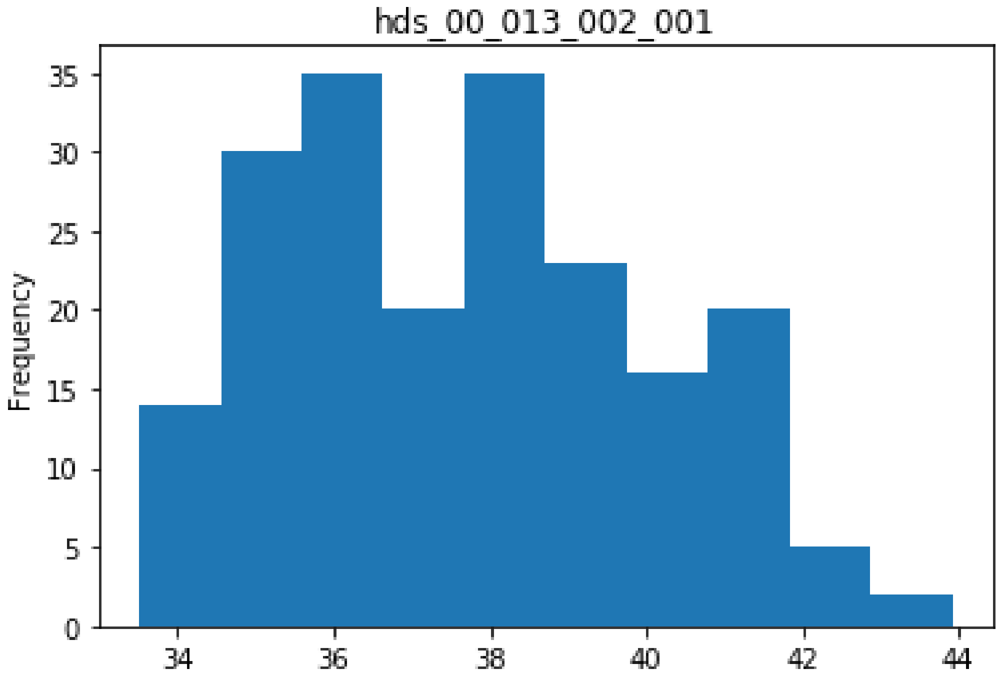


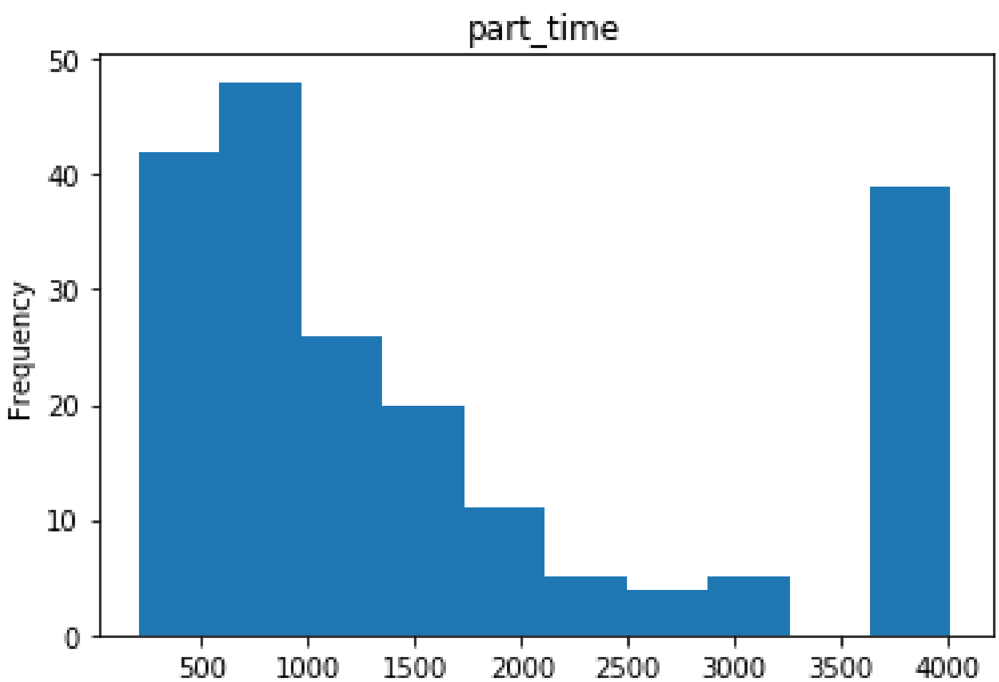

fa_hw_19801229
Frequency
0
10
20
30
40
50
60
70
−1000
−500
0
500
1000
1500
2000
2500

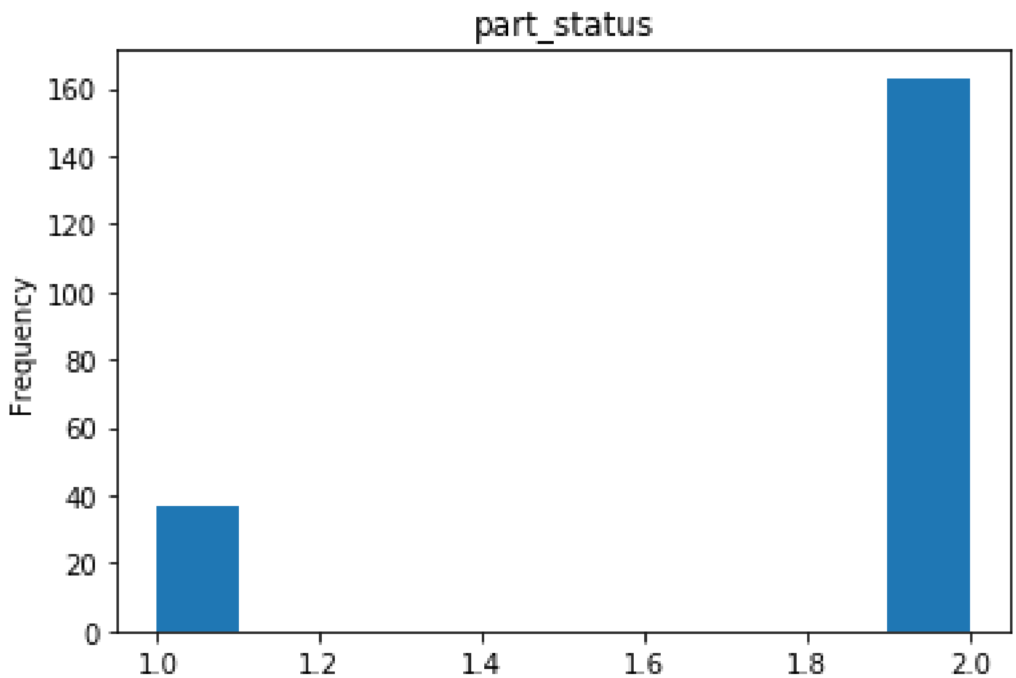

fa_tw_19791230
Frequency
0
10
20
30
40
50
−1000
−500
0
500
1000
1500
2000


fa_tw_19801229
Frequency
0
10
20
30
40
−500
0
500
1000
1500
2000
2500


hds_00_013_002_000
Frequency
0
10
20
30
40
50
35.0
37.5
40.0
42.5
45.0
47.5
50.0

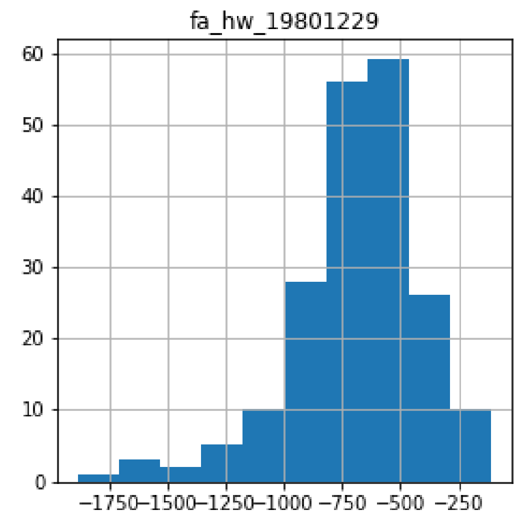

hds_00_013_002_001
Frequency
0
5
10
15
20
25
30
35
34
36
38
40
42
44

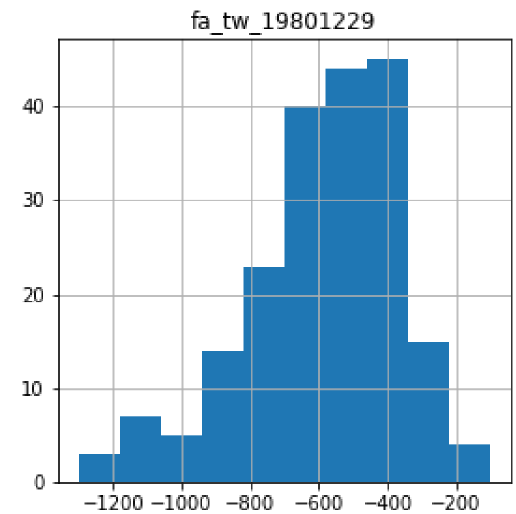

part_time
Frequency
0
10
20
30
40
50
500
1000
1500
2000
2500
3000
3500
4000

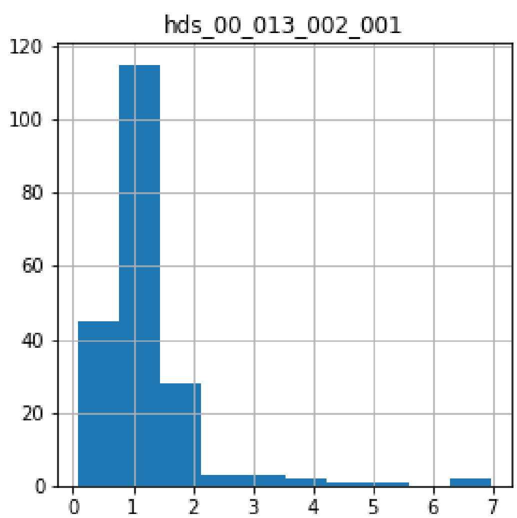


We see that under scenario conditions, many more realizations for the flow to the aquifer in the headwaters are postive (as expected). Lets difference these two:

```
In [10]: sfnames = [f for f in fnames if "1980" in f or "_001" in f]
         hfnames = [f for f in fnames if "1979" in f or "_000" in f]
         diff = obs_df.loc[:,hfnames].values - obs_df.loc[:,sfnames].values
         diff = pd.DataFrame(diff,columns=sfnames)
         diff.hist(figsize=(10,10))
         plt.show()
```







We now see that the most extreme scenario yields a large decrease in flow from the aquifer to the headwaters (the most negative value)

1.0.7 setting the "truth"

We just need to replace the observed values (`obsval`) in the control file with the outputs for one of the realizations on `obs_df`. In this way, we now have the nonzero values for history matching, but also the `truth` values for comparing how we are doing with other unobserved quantities. I'm going to pick a realization that yields an "average" variability of the observed gw levels:

In [11]:
```
# choose the realization with a low historic gw to sw headwater flux
#hist_swgw = obs_df.loc[:,"fa_hw_19791230"].sort_values()
hist_swgw = obs_df.loc[:,"part_time"].sort_values()
idx = hist_swgw.index[20]
```

```
idx
hist_swgw
```

Out[11]:

```
run_id
149       207.9040
165       237.0716
191       280.1461
97        286.6308
77        289.1803
16        304.0466
173       326.0275
186       332.5973
180       355.7679
145       357.3499
78        359.7271
107       378.5330
17        391.6625
156       399.4463
76        410.1112
0         415.2489
96        437.2630
46        439.3421
2         442.4801
35        442.4812
64        452.2543
128       453.5696
141       464.6295
93        469.9462
21        480.9774
117       482.3959
6         489.8026
73        492.7455
11        501.1917
62        517.6446
            ...
8        4015.0000
118      4015.0000
190      4015.0000
69       4015.0000
193      4015.0000
194      4015.0000
12       4015.0000
47       4015.0000
75       4015.0000
27       4015.0000
82       4015.0000
104      4015.0000
153      4015.0000
```

```
138    4015.0000
136    4015.0000
157    4015.0000
159    4015.0000
41     4015.0000
125    4015.0000
74     4015.0000
84     4015.0000
166    4015.0000
29     4015.0000
169    4015.0000
171    4015.0000
172    4015.0000
144    4015.0000
175    4015.0000
131    4015.0000
42     4015.0000
Name: part_time, Length: 200, dtype: float64
```

```
In [12]: obs_df.loc[idx,pst.nnz_obs_names]
```

```
Out[12]: fo_39_19791230        10555.000000
hds_00_002_009_000       34.847549
hds_00_002_015_000       34.639633
hds_00_003_008_000       34.861164
hds_00_009_001_000       35.161556
hds_00_013_010_000       34.661835
hds_00_015_016_000       34.569942
hds_00_021_010_000       34.571648
hds_00_022_015_000       34.439877
hds_00_024_004_000       34.609158
hds_00_026_006_000       34.521420
hds_00_029_015_000       34.297997
hds_00_033_007_000       34.187607
hds_00_034_010_000       33.842358
Name: 64, dtype: float64
```

Lets see how our selected truth does with the sw/gw forecasts:

```
In [13]: obs_df.loc[idx,fnames]
```

```
Out[13]: fa_hw_19791230       -992.855500
fa_hw_19801229        -73.877930
fa_tw_19791230       -256.953300
fa_tw_19801229        368.869266
hds_00_013_002_000     35.032547
hds_00_013_002_001     34.416519
part_time             452.254300
part_status             2.000000
Name: 64, dtype: float64
```

Assign some initial weights. Now, it is custom to add noise to the observed values…we will use the classic Gaussian noise…zero mean and standard deviation of 1 over the weight

```
In [14]: pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
         obs = pst.observation_data
         obs.loc[:,"obsval"] = obs_df.loc[idx,pst.obs_names]
         obs.loc[obs.obgnme=="calhead","weight"] = 10.0
         obs.loc[obs.obgnme=="calflux","weight"] = 1.0
```

here we just get a sample from a random normal distribution with mean=0 and std=1. The argument indicates how many samples we want - and we choose `pst.nnz_obs` which is the the number of nonzero-weighted observations in the PST file

```
In [15]: np.random.seed(seed=0)
         snd = np.random.randn(pst.nnz_obs)
         noise = snd * 1./obs.loc[pst.nnz_obs_names,"weight"]
         pst.observation_data.loc[noise.index,"obsval"] += noise
         noise
```

```
Out[15]: obsnme
         fo_39_19791230         1.764052
         hds_00_002_009_000     0.040016
         hds_00_002_015_000     0.097874
         hds_00_003_008_000     0.224089
         hds_00_009_001_000     0.186756
         hds_00_013_010_000    -0.097728
         hds_00_015_016_000     0.095009
         hds_00_021_010_000    -0.015136
         hds_00_022_015_000    -0.010322
         hds_00_024_004_000     0.041060
         hds_00_026_006_000     0.014404
         hds_00_029_015_000     0.145427
         hds_00_033_007_000     0.076104
         hds_00_034_010_000     0.012168
         Name: weight, dtype: float64
```

Then we write this out to a new file and run `pestpp-ies` to see how the objective function looks

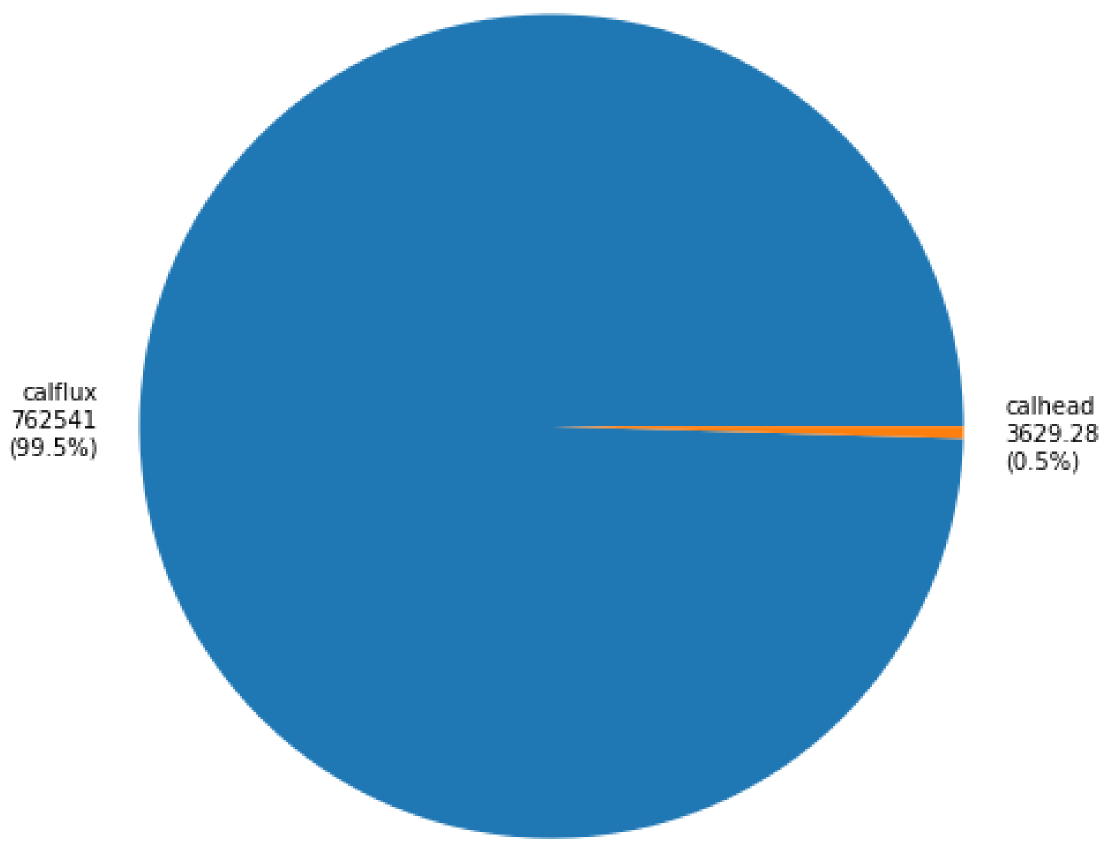
```
In [16]: pst.write(os.path.join(t_d,"freyberg.pst"))
         pyemu.os_utils.run("pestpp-ies freyberg.pst",cwd=t_d)
```

```
noptmax:0, npar_adj:14819, nnz_obs:14
```

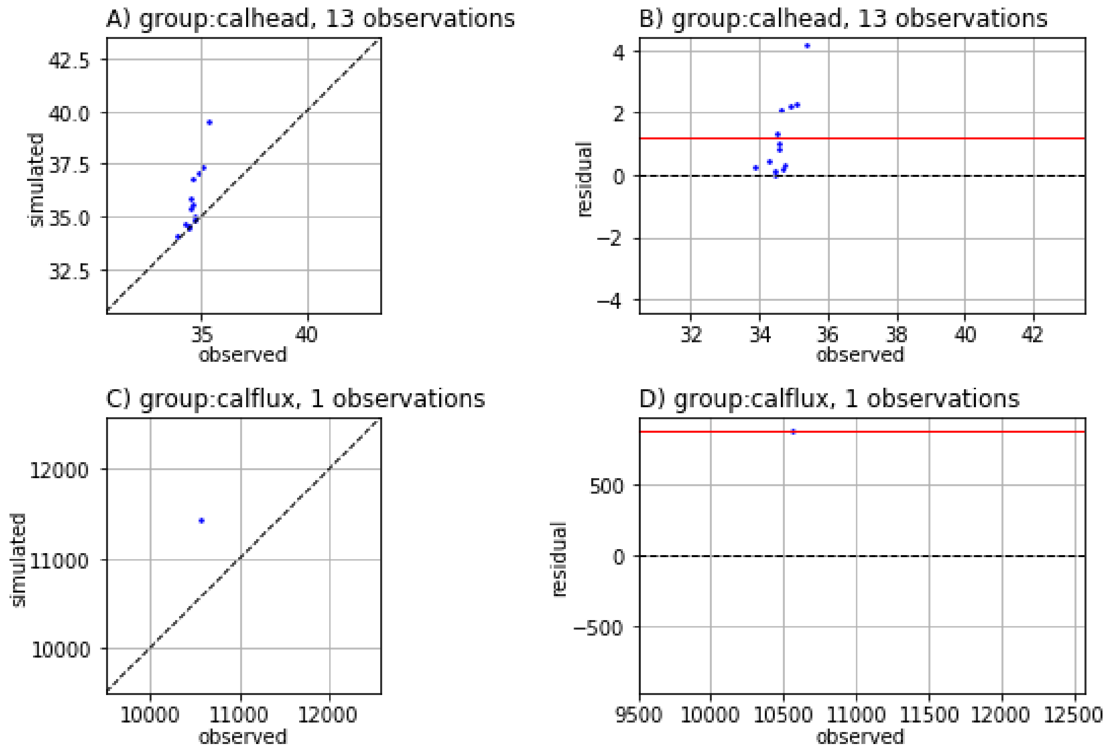
Now we can read in the results and make some figures showing residuals and the balance of the objective function

```
In [17]: pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
         print(pst.phi)
         plt.figure()
         pst.plot(kind='phi_pie')
         print('Here are the non-zero weighted observation names')

         figs = pst.plot(kind="1to1")
         plt.show()
         pst.res.loc[pst.nnz_obs_names,:]
```

```
766170.2973169514
Here are the non-zero weighted observation names


<Figure size 432x288 with 0 Axes>
```



<Figure size 576x756 with 0 Axes>



Out[17]: name group measured modelled \

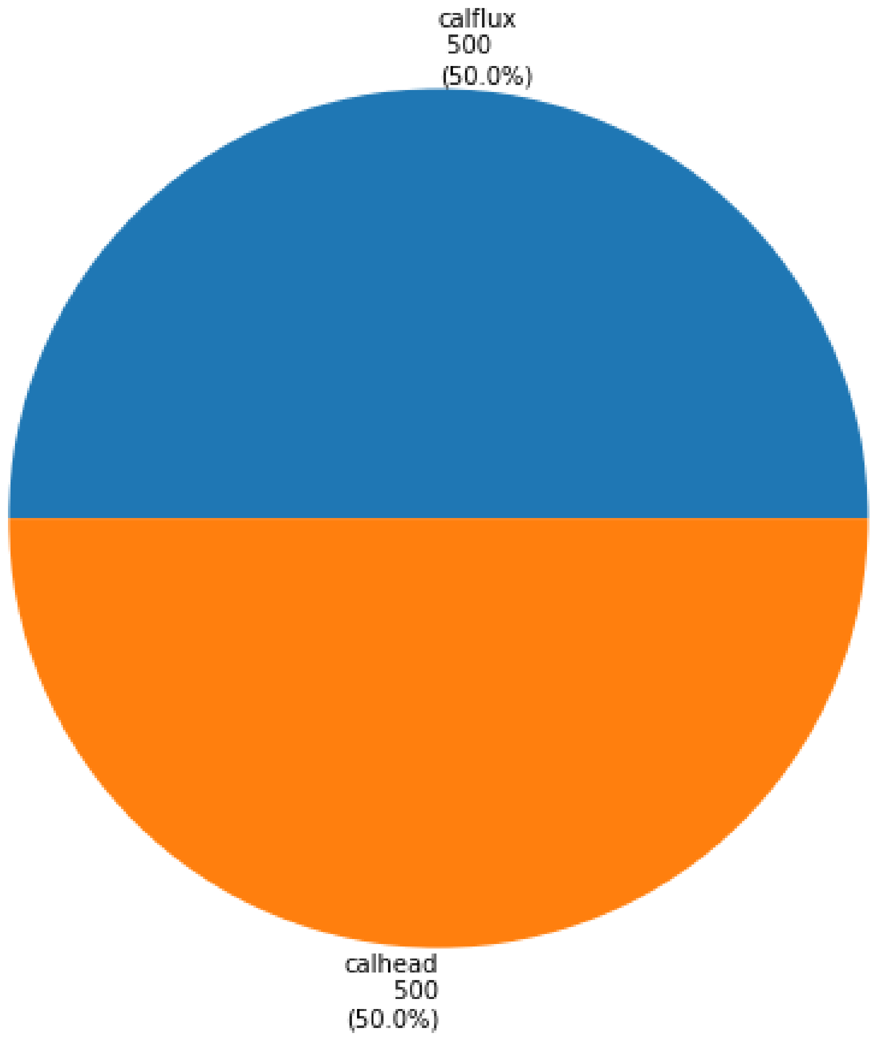
```
name
fo_39_19791230          fo_39_19791230  calflux  10556.764052  11430.000000
hds_00_002_009_000  hds_00_002_009_000  calhead     34.887565     37.107498
hds_00_002_015_000  hds_00_002_015_000  calhead     34.737507     35.045185
hds_00_003_008_000  hds_00_003_008_000  calhead     35.085253     37.397289
hds_00_009_001_000  hds_00_009_001_000  calhead     35.348312     39.546417
hds_00_013_010_000  hds_00_013_010_000  calhead     34.564107     35.571774
hds_00_015_016_000  hds_00_015_016_000  calhead     34.664951     34.835716
hds_00_021_010_000  hds_00_021_010_000  calhead     34.556512     35.386250
hds_00_022_015_000  hds_00_022_015_000  calhead     34.429555     34.577492
hds_00_024_004_000  hds_00_024_004_000  calhead     34.650217     36.760464
hds_00_026_006_000  hds_00_026_006_000  calhead     34.535824     35.896149
hds_00_029_015_000  hds_00_029_015_000  calhead     34.443424     34.453842
hds_00_033_007_000  hds_00_033_007_000  calhead     34.263711     34.678810
hds_00_034_010_000  hds_00_034_010_000  calhead     33.854525     34.118073

                      residual  weight
name
fo_39_19791230     -873.235948     1.0
hds_00_002_009_000   -2.219933    10.0
hds_00_002_015_000   -0.307678    10.0
hds_00_003_008_000   -2.312036    10.0
hds_00_009_001_000   -4.198105    10.0
hds_00_013_010_000   -1.007667    10.0
hds_00_015_016_000   -0.170765    10.0
hds_00_021_010_000   -0.829738    10.0
hds_00_022_015_000   -0.147937    10.0
hds_00_024_004_000   -2.110246    10.0
hds_00_026_006_000   -1.360325    10.0
hds_00_029_015_000   -0.010418    10.0
hds_00_033_007_000   -0.415100    10.0
hds_00_034_010_000   -0.263547    10.0
```

Publication ready figs - oh snap!

Depending on the truth you chose, we may have a problem - we set the weights for both the heads and the flux to reasonable values based on what we expect for measurement noise. But the contributions to total phi might be out of balance - if contribution of the flux measurement to total phi is too low, the history matching excersizes (coming soon!) will focus almost entirely on minimizing head residuals. So we need to balance the objective function. This is a subtle but very important step, especially since some of our forecasts deal with sw-gw exchange

In [18]:
```
pc = pst.phi_components
#target = {"calflux":0.3 * pc["calhead"]}
target = {"calhead":500,"calflux":500}
pst.adjust_weights(obsgrp_dict=target)
pst.plot(kind='phi_pie')
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x10ec8f2b0>
```



Lets see what the new flux observation weight is:

```
In [19]: pst.observation_data.loc[pst.nnz_obs_names,"weight"]
```

```
Out[19]: obsnme
         fo_39_19791230        0.025607
         hds_00_002_009_000    3.711718
         hds_00_002_015_000    3.711718
         hds_00_003_008_000    3.711718
         hds_00_009_001_000    3.711718
         hds_00_013_010_000    3.711718
         hds_00_015_016_000    3.711718
         hds_00_021_010_000    3.711718
         hds_00_022_015_000    3.711718
```

```
hds_00_024_004_000    3.711718
hds_00_026_006_000    3.711718
hds_00_029_015_000    3.711718
hds_00_033_007_000    3.711718
hds_00_034_010_000    3.711718
Name: weight, dtype: float64
```

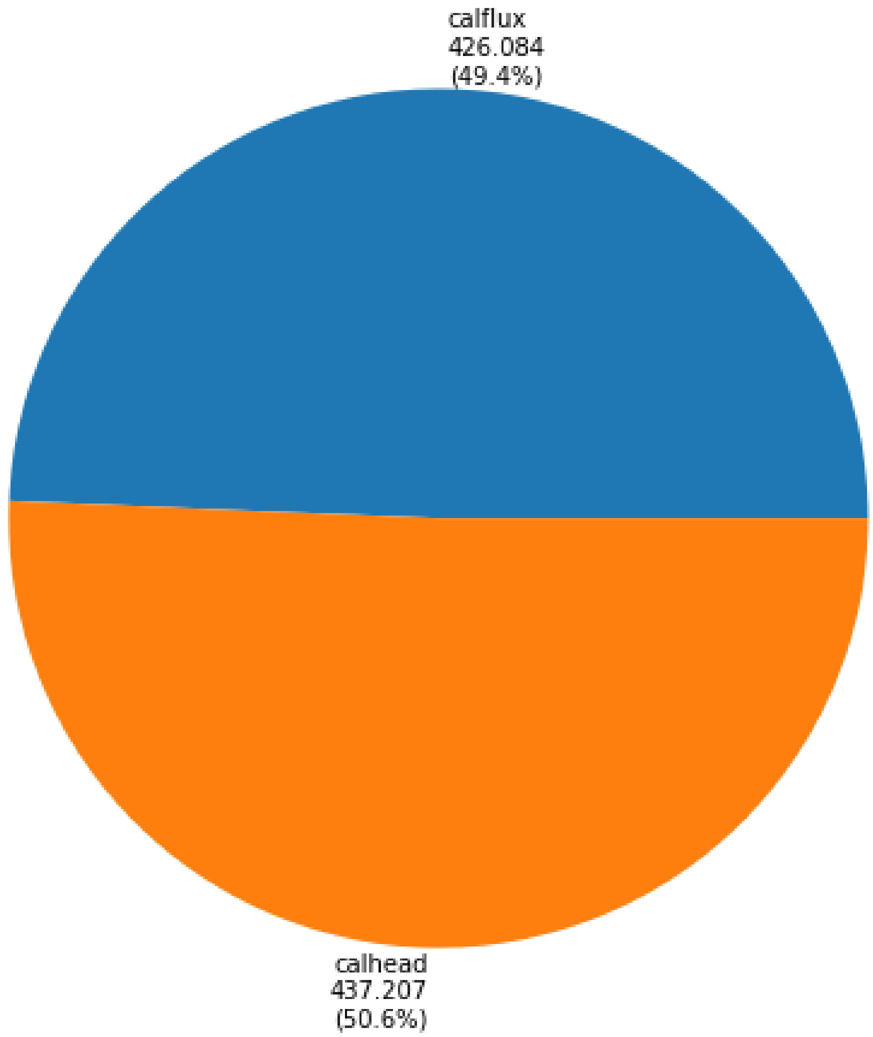
Now, for some super trickery: since we changed the weight, we need to generate the observation noise using these new weights for the error model (so meta!)

```
In [20]: obs = pst.observation_data
         np.random.seed(seed=0)
         snd = np.random.randn(pst.nnz_obs)
         noise = snd * 1./obs.loc[pst.nnz_obs_names,"weight"]
         obs.loc[:,"obsval"] = obs_df.loc[idx,pst.obs_names]
         pst.observation_data.loc[noise.index,"obsval"] += noise
         noise
```

```
Out[20]: obsnme
         fo_39_19791230        68.890299
         hds_00_002_009_000     0.107809
         hds_00_002_015_000     0.263689
         hds_00_003_008_000     0.603735
         hds_00_009_001_000     0.503152
         hds_00_013_010_000    -0.263295
         hds_00_015_016_000     0.255970
         hds_00_021_010_000    -0.040778
         hds_00_022_015_000    -0.027809
         hds_00_024_004_000     0.110622
         hds_00_026_006_000     0.038808
         hds_00_029_015_000     0.391806
         hds_00_033_007_000     0.205037
         hds_00_034_010_000     0.032781
         Name: weight, dtype: float64
```

```
In [21]: pst.write(os.path.join(t_d,"freyberg.pst"))
         pyemu.os_utils.run("pestpp-ies freyberg.pst",cwd=t_d)
         pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
         print(pst.phi)
         pst.plot(kind='phi_pie')
         plt.show()
```

```
noptmax:0, npar_adj:14819, nnz_obs:14
863.2909159111605
```



Whew! confused yet? Ok, let's leave all this confusion behind…its mostly academic, just to make sure we are using weights that are in harmony with the noise we added to the truth…Just to make sure we have everything working right, we should be able to load the truth parameters, run the model once and have a `phi` equivalent to the noise vector:

```
In [22]: par_df = pd.read_csv(os.path.join(m_d,"sweep_in.csv"),index_col=0)
         pst.parameter_data.loc[:,"parval1"] = par_df.loc[idx,pst.par_names]
         pst.write(os.path.join(m_d,"test.pst"))
```

```
noptmax:0, npar_adj:14819, nnz_obs:14
```

we will run this with `noptmax=0` to preform a single run. Pro-tip: you can use any of the `pestpp-###` binaries/executables to run `noptmax=0`

```
In [23]: pyemu.os_utils.run("pestpp-ies.exe test.pst",cwd=m_d)
         pst = pyemu.Pst(os.path.join(m_d,"test.pst"))
         print(pst.phi)
         pst.res.loc[pst.nnz_obs_names,:]
```

```
17.528847196782618
```

```
Out[23]:                                  name    group      measured      modelled  \
         name
         fo_39_19791230          fo_39_19791230  calflux  10623.890299  10555.000000
         hds_00_002_009_000  hds_00_002_009_000  calhead     34.955359     34.847549
         hds_00_002_015_000  hds_00_002_015_000  calhead     34.903322     34.639633
         hds_00_003_008_000  hds_00_003_008_000  calhead     35.464899     34.861164
         hds_00_009_001_000  hds_00_009_001_000  calhead     35.664708     35.161556
         hds_00_013_010_000  hds_00_013_010_000  calhead     34.398539     34.661835
         hds_00_015_016_000  hds_00_015_016_000  calhead     34.825912     34.569942
         hds_00_021_010_000  hds_00_021_010_000  calhead     34.530869     34.571648
         hds_00_022_015_000  hds_00_022_015_000  calhead     34.412068     34.439877
         hds_00_024_004_000  hds_00_024_004_000  calhead     34.719780     34.609158
         hds_00_026_006_000  hds_00_026_006_000  calhead     34.560227     34.521420
         hds_00_029_015_000  hds_00_029_015_000  calhead     34.689803     34.297997
         hds_00_033_007_000  hds_00_033_007_000  calhead     34.392643     34.187607
         hds_00_034_010_000  hds_00_034_010_000  calhead     33.875139     33.842358

                              residual    weight
         name
         fo_39_19791230      68.890299  0.025607
         hds_00_002_009_000   0.107809  3.711718
         hds_00_002_015_000   0.263689  3.711718
         hds_00_003_008_000   0.603735  3.711718
         hds_00_009_001_000   0.503152  3.711718
         hds_00_013_010_000  -0.263295  3.711718
         hds_00_015_016_000   0.255970  3.711718
         hds_00_021_010_000  -0.040778  3.711718
         hds_00_022_015_000  -0.027809  3.711718
         hds_00_024_004_000   0.110622  3.711718
         hds_00_026_006_000   0.038808  3.711718
         hds_00_029_015_000   0.391806  3.711718
         hds_00_033_007_000   0.205037  3.711718
         hds_00_034_010_000   0.032781  3.711718
```

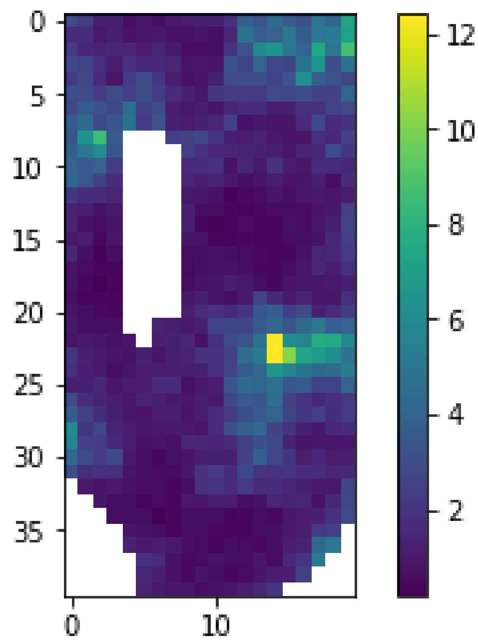
The residual should be exactly the noise values from above.Lets load the model (that was just run using the true pars) and check some things

```
In [24]: m = flopy.modflow.Modflow.load("freyberg.nam",model_ws=m_d)
```

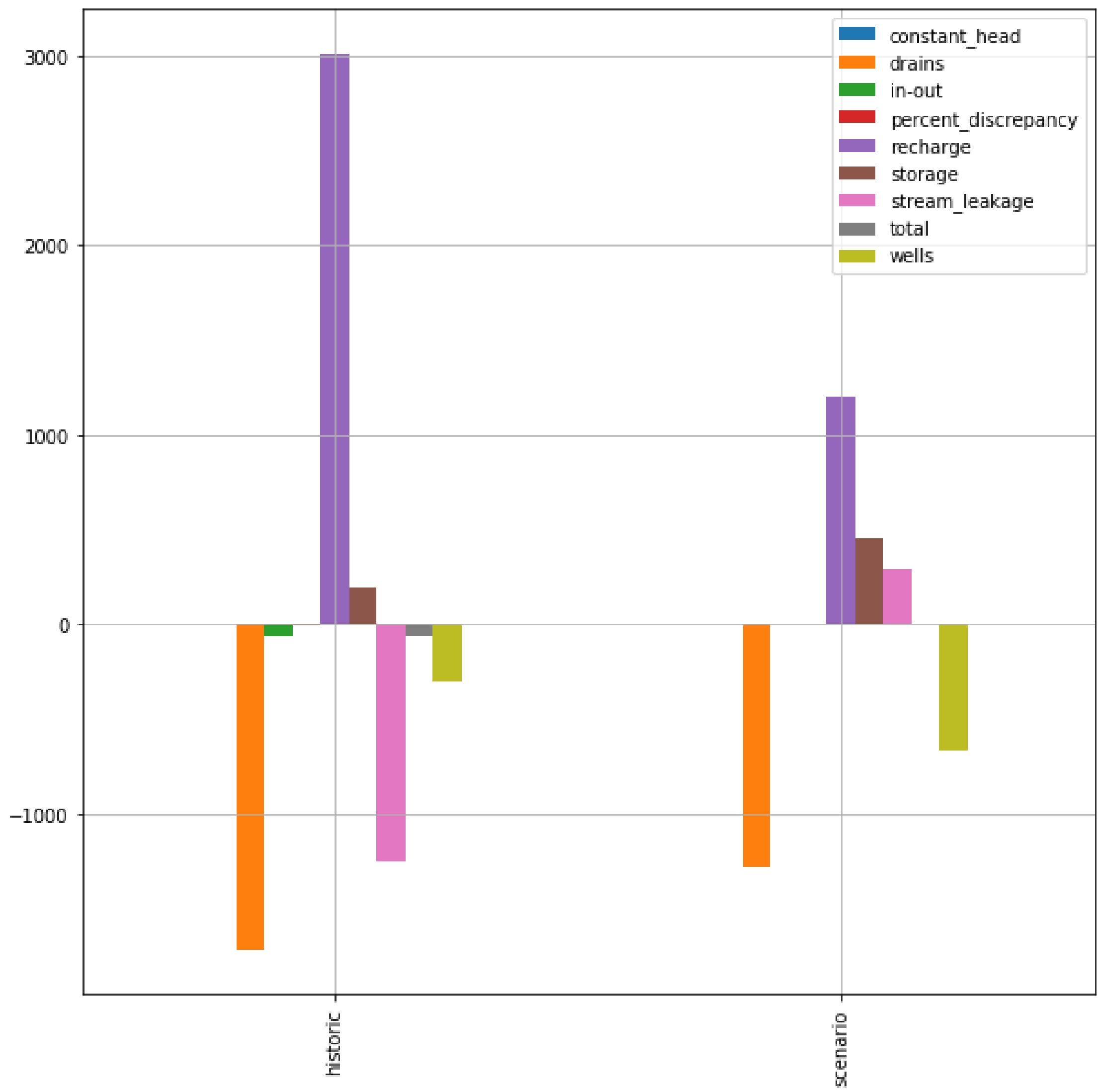
```
In [25]: a = m.upw.hk[0].array
         #a = m.rch.rech[0].array
```

```
a = np.ma.masked_where(m.bas6.ibound[0].array==0,a)
print(a.min(),a.max())
c = plt.imshow(a)
plt.colorbar()
plt.show()
```

```
0.1996812 12.40834
```



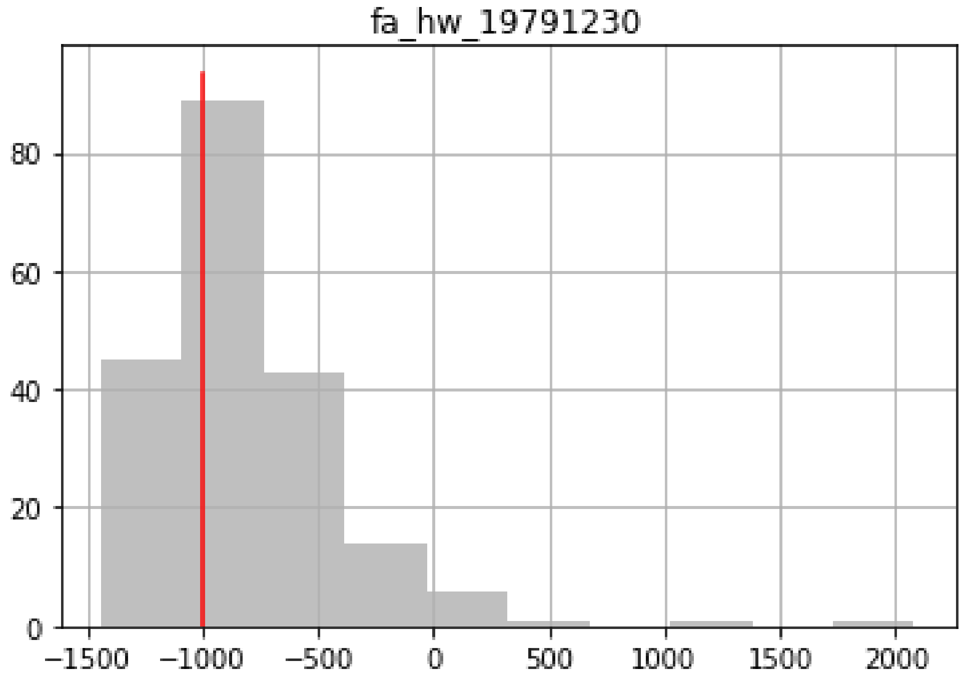

```
In [26]: lst = flopy.utils.MfListBudget(os.path.join(m_d,"freyberg.list"))
         df = lst.get_dataframes(diff=True)[0]
         ax = df.plot(kind="bar",figsize=(10,10), grid=True)
         a = ax.set_xticklabels(["historic","scenario"],rotation=90)
         plt.show(ax)
```

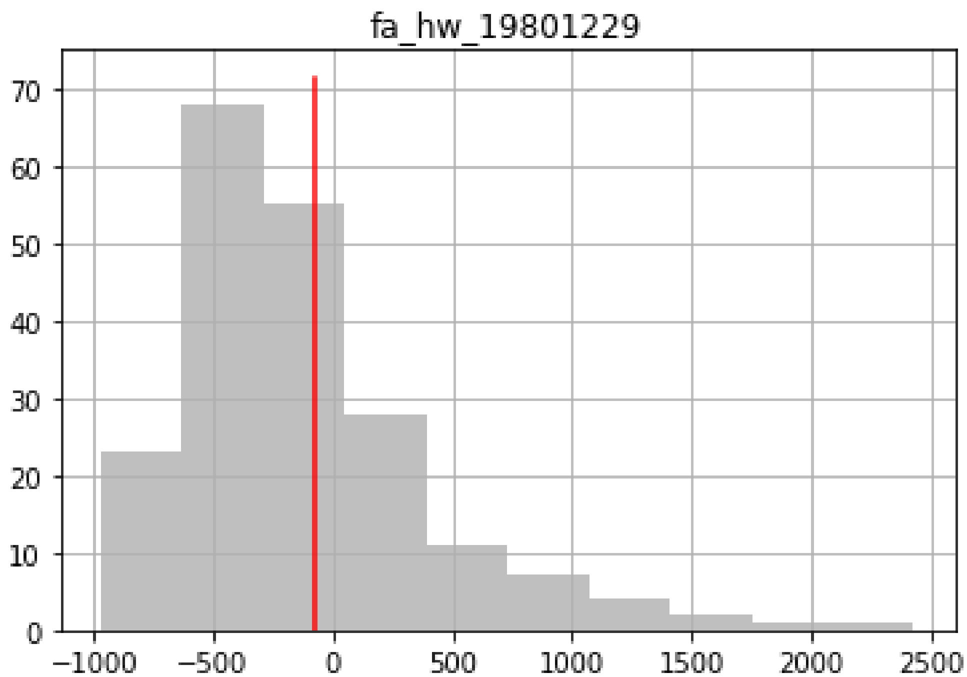


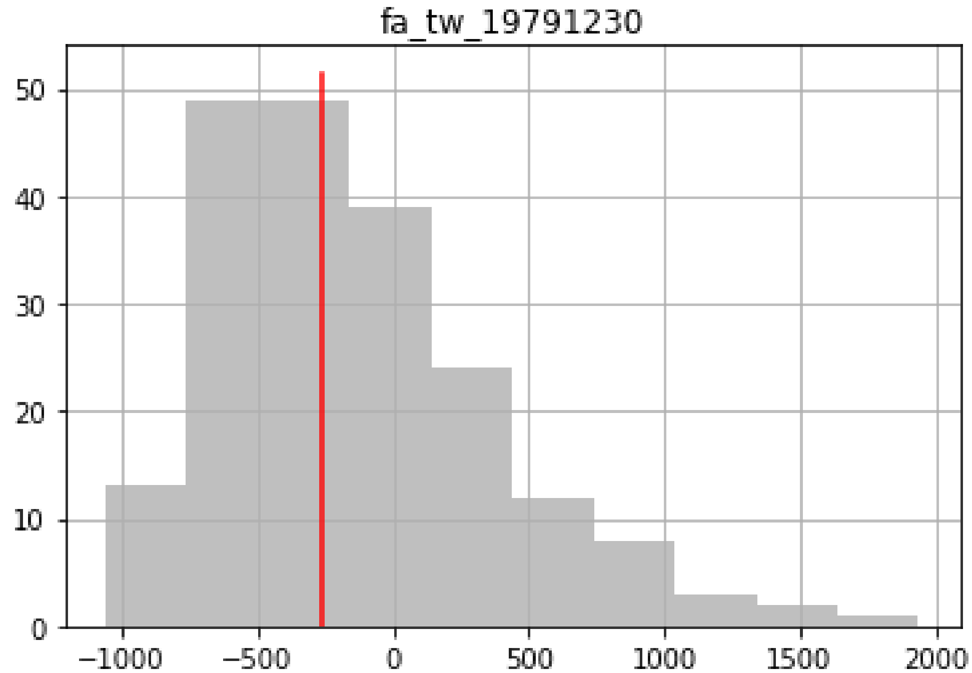
1.0.8 see how our existing observation ensemble compares to the truth

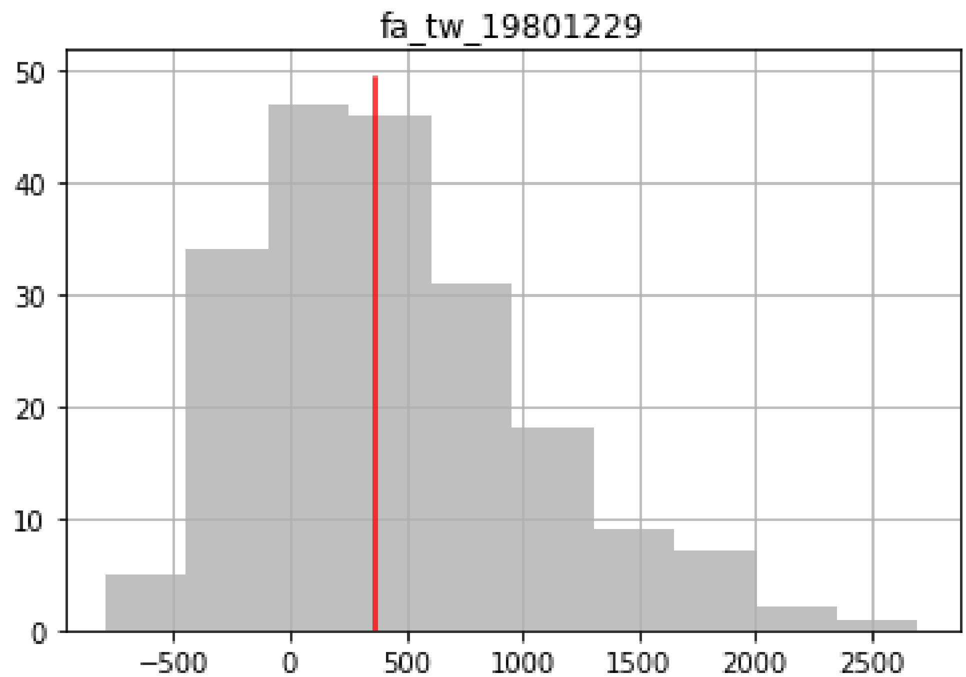
forecasts:

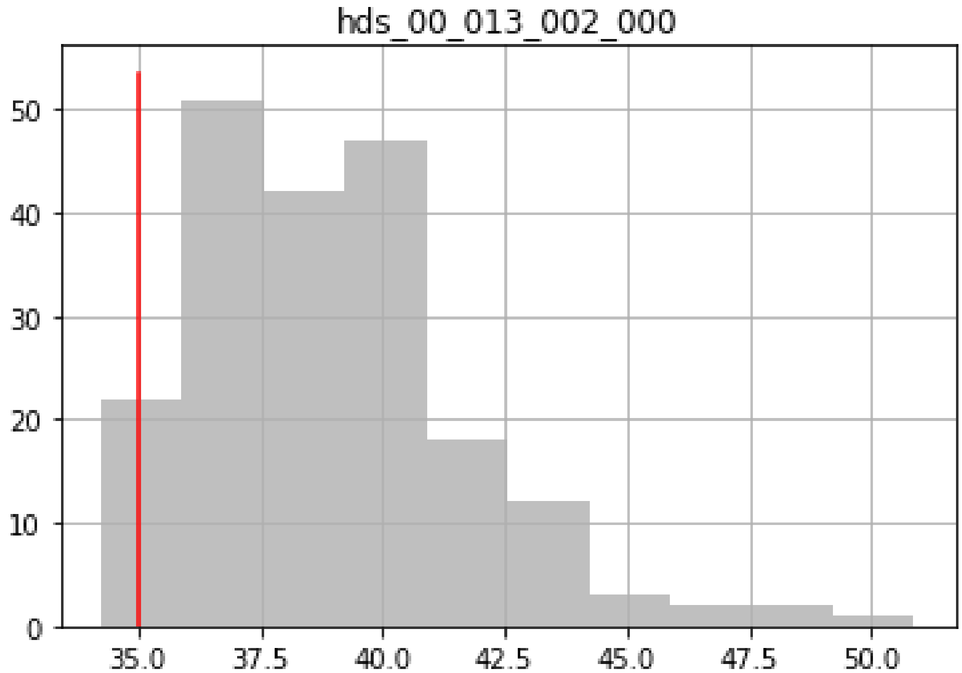
```
In [27]: obs = pst.observation_data
         plt.figure()
         for forecast in fnames:
             ax = plt.subplot(111)
             obs_df.loc[:,forecast].hist(ax=ax,color="0.5",alpha=0.5)
             ax.plot([obs.loc[forecast,"obsval"],obs.loc[forecast,"obsval"]],ax.get_ylim(),"r"
             ax.set_title(forecast)
             plt.show()
```

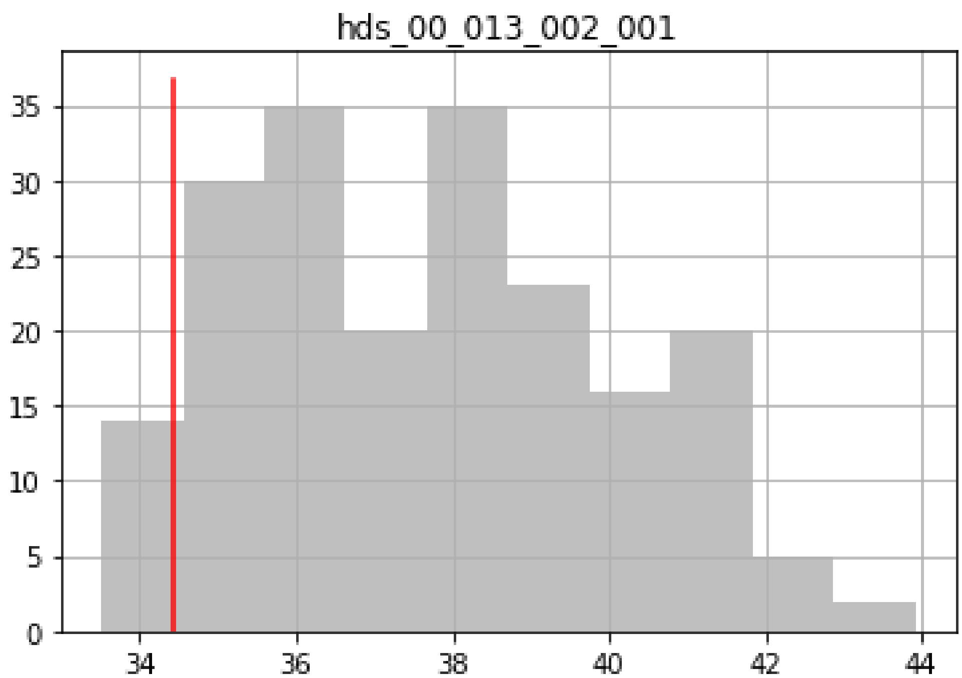

fa_hw_19791230
0
20
40
60
80
−1500
−1000
−500
0
500
1000
1500
2000

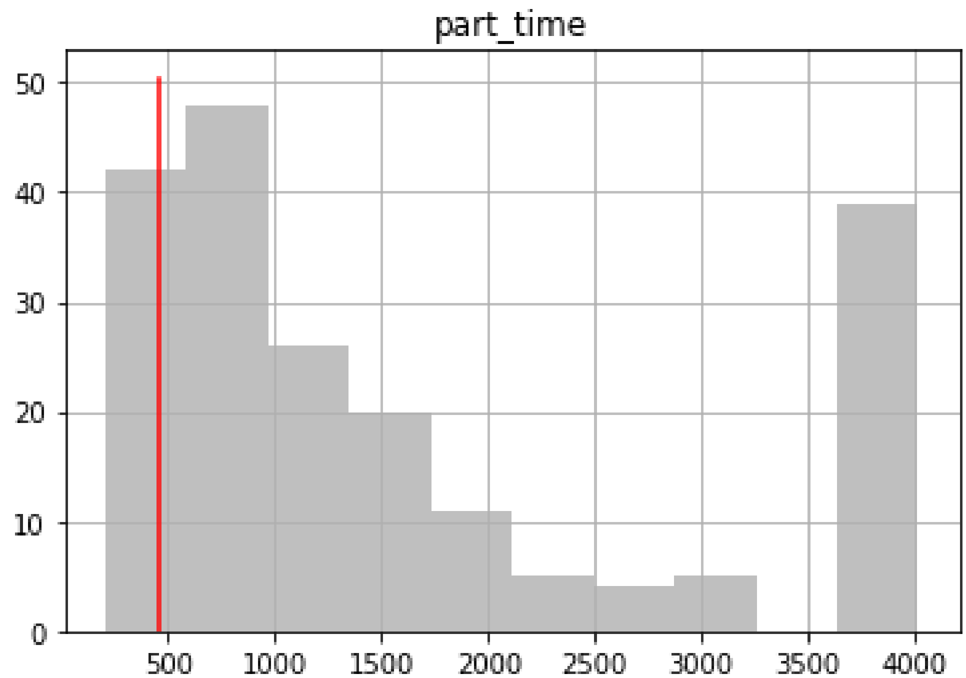

fa_hw_19801229
0
10
20
30
40
50
60
70
−1000
−500
0
500
1000
1500
2000
2500

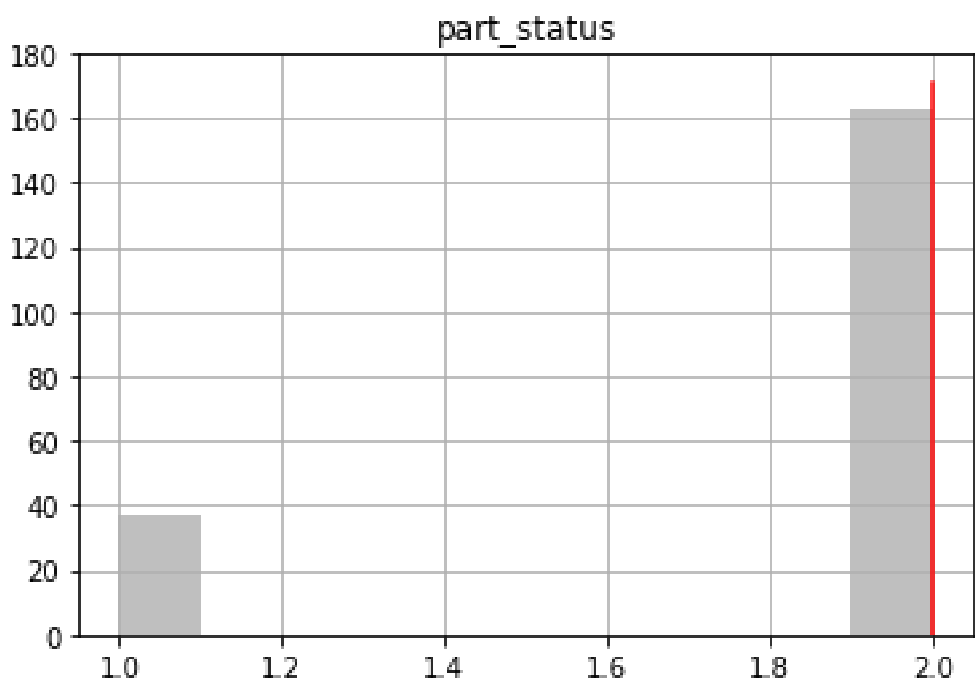

fa_tw_19791230
50
40
30
20
10
0
−1000
−500
0
500
1000
1500
2000


fa_tw_19801229
50
40
30
20
10
0
−500
0
500
1000
1500
2000
2500


hds_00_013_002_000
0
10
20
30
40
50
35.0
37.5
40.0
42.5
45.0
47.5
50.0

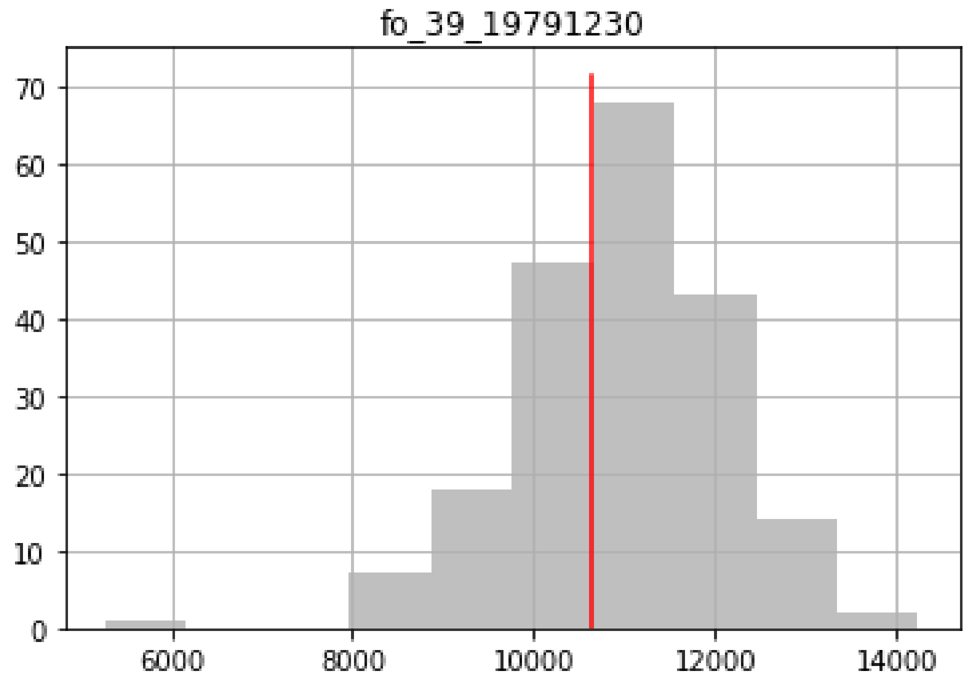

hds_00_013_002_001
0
5
10
15
20
25
30
35
34
36
38
40
42
44

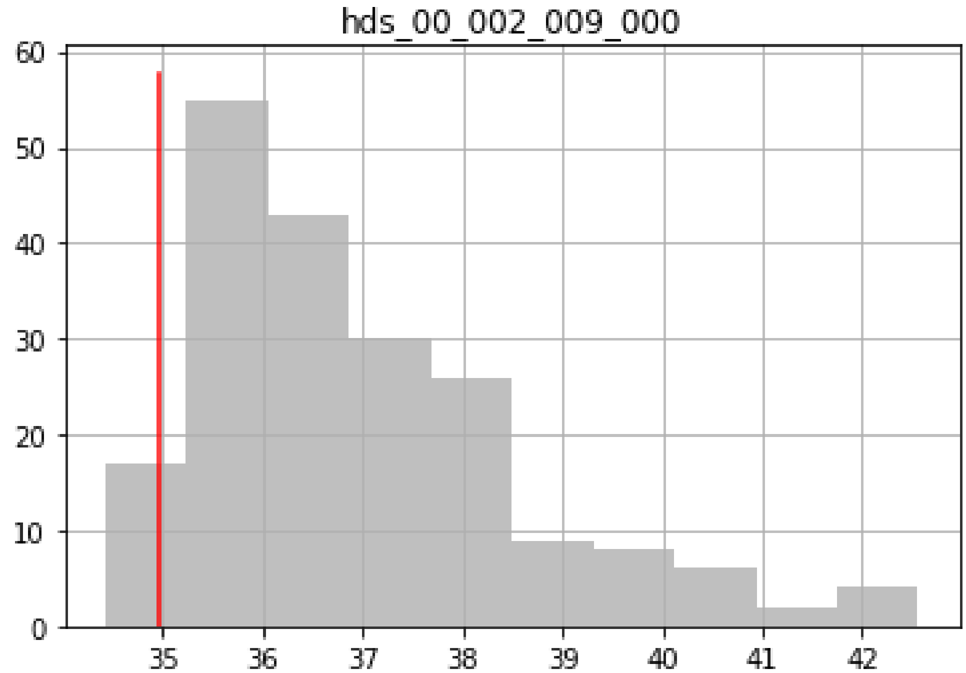


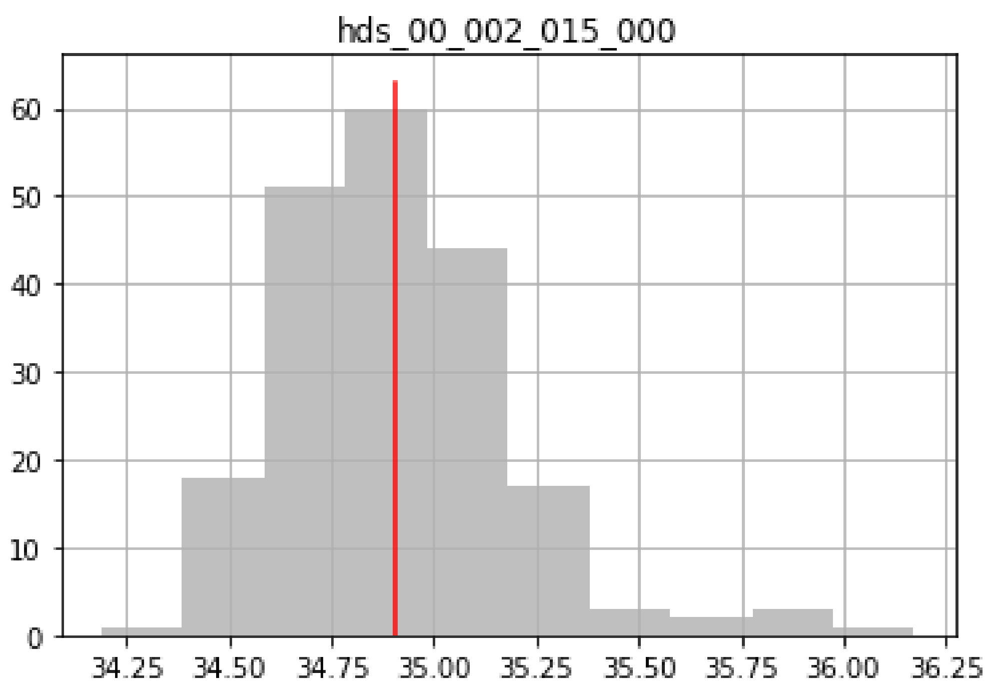


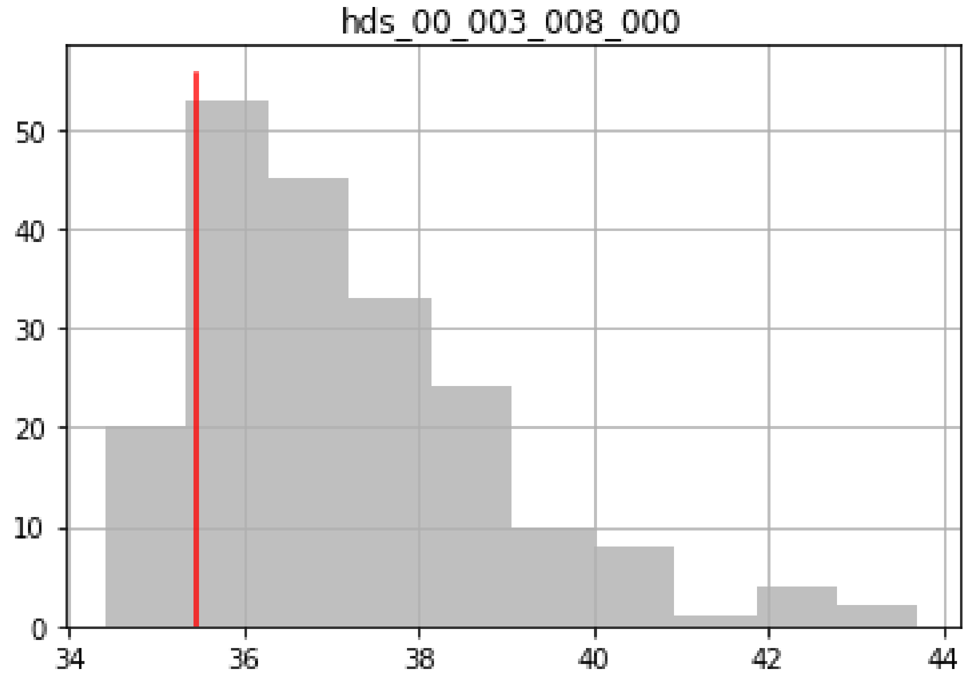
observations:

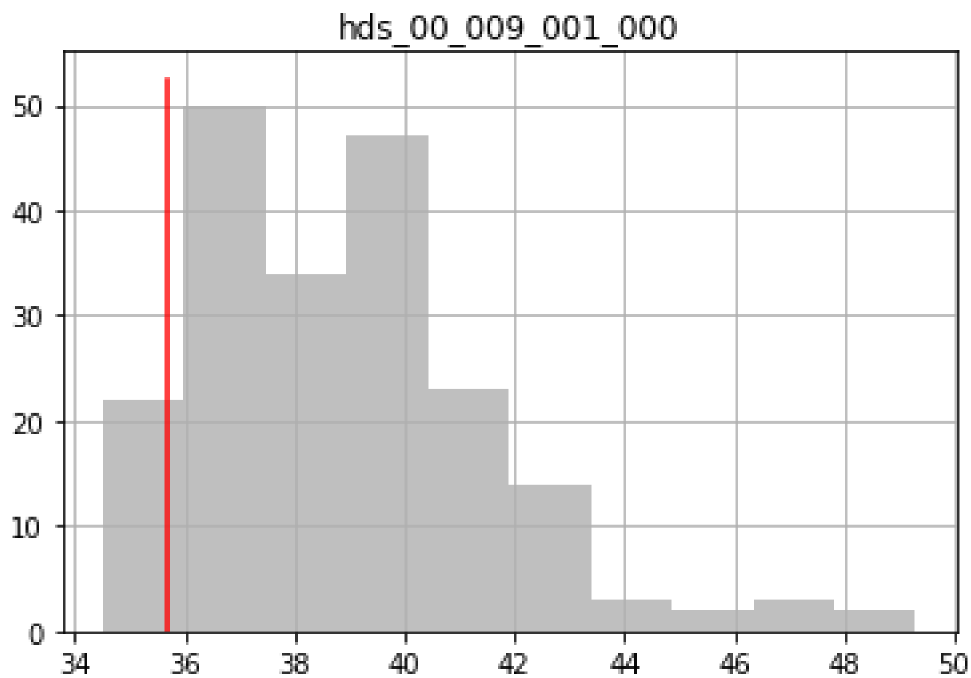
In [28]:
```
for oname in pst.nnz_obs_names:
    ax = plt.subplot(111)
    obs_df.loc[:,oname].hist(ax=ax,color="0.5",alpha=0.5)
    ax.plot([obs.loc[oname,"obsval"],obs.loc[oname,"obsval"]],ax.get_ylim(),"r")
    ax.set_title(oname)
    plt.show()
```

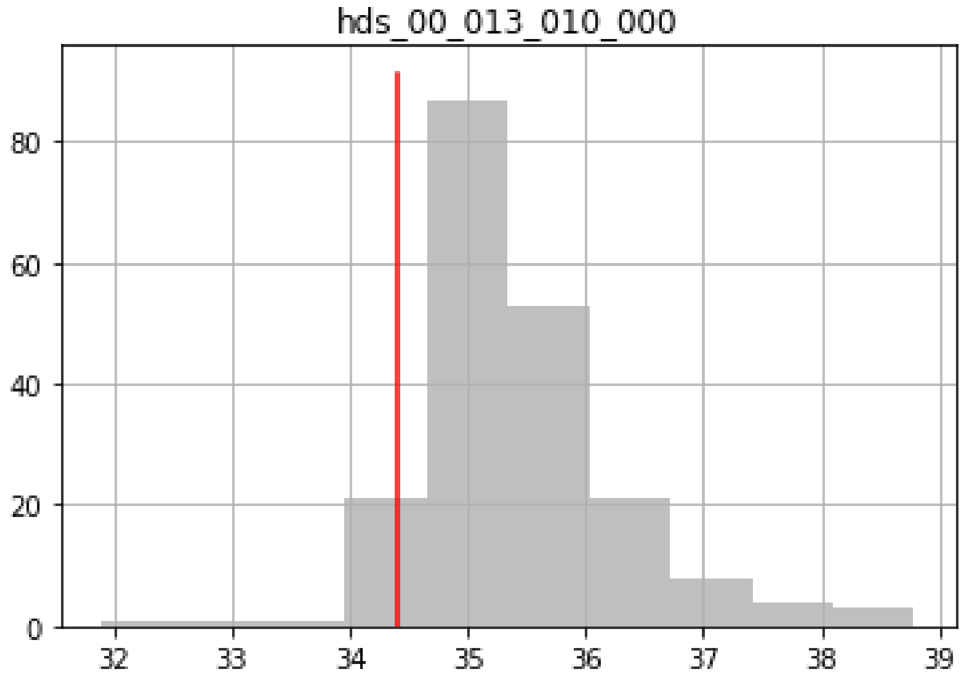


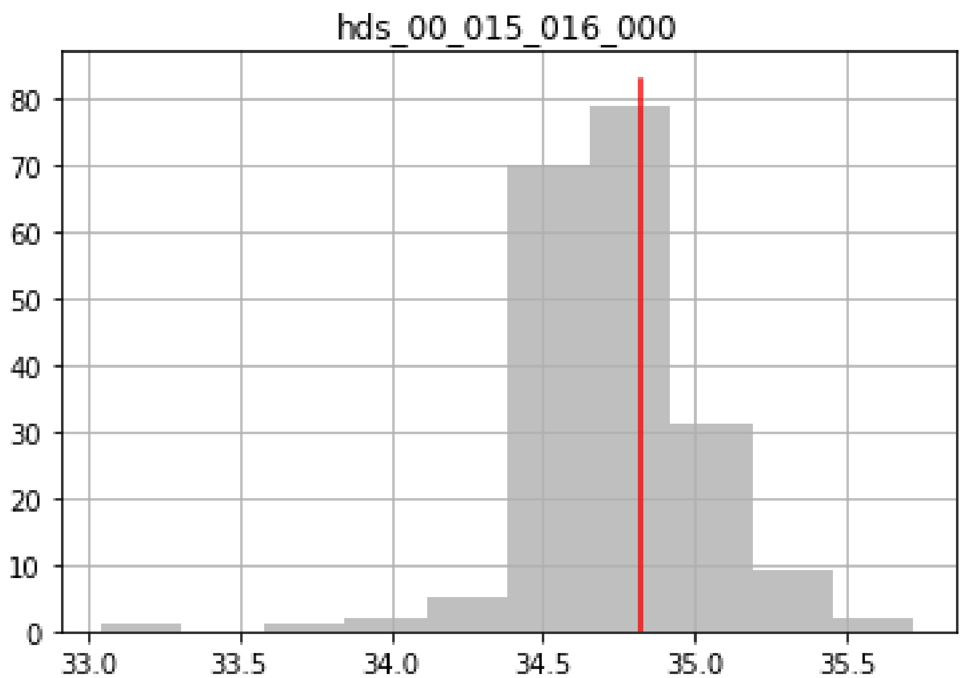

hds_00_002_009_000
0
10
20
30
40
50
60
35
36
37
38
39
40
41
42

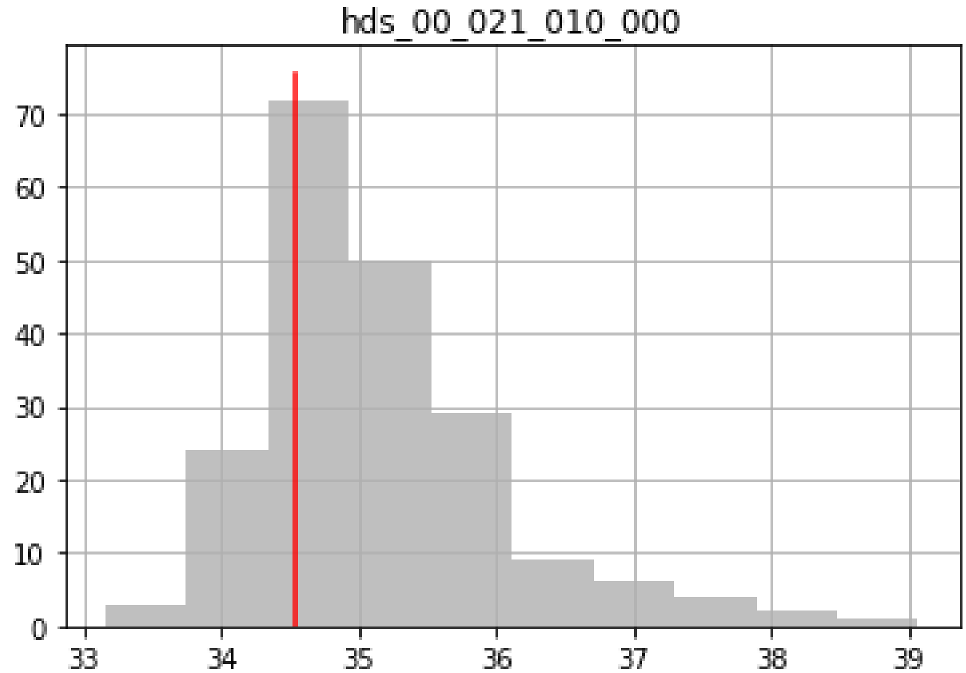

hds_00_002_015_000
0
10
20
30
40
50
60
34.25
34.50
34.75
35.00
35.25
35.50
35.75
36.00
36.25

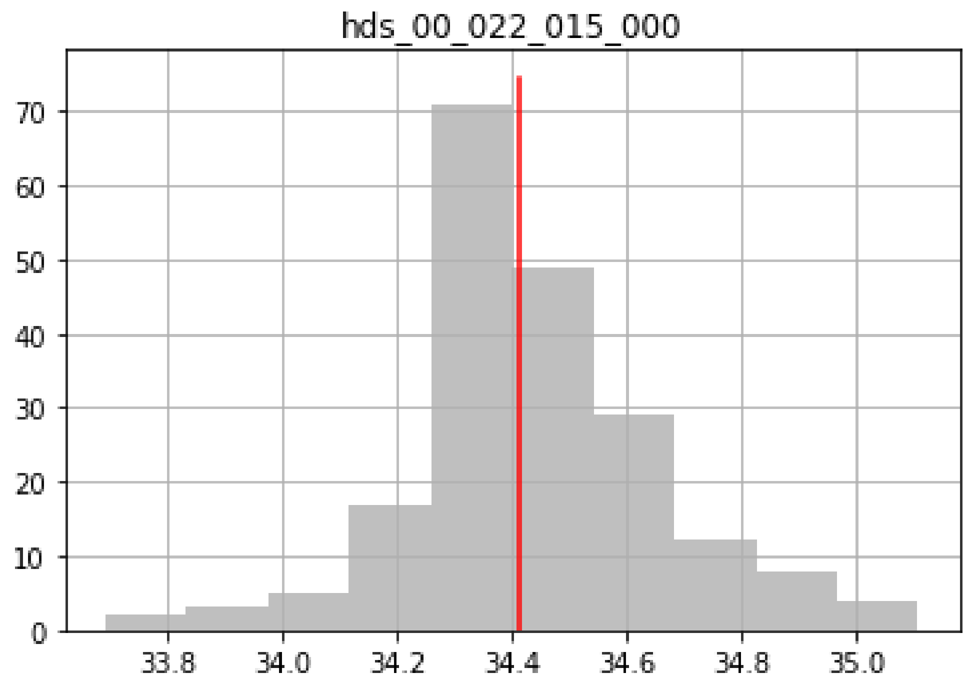

hds_00_003_008_000
0
10
20
30
40
50
34
36
38
40
42
44

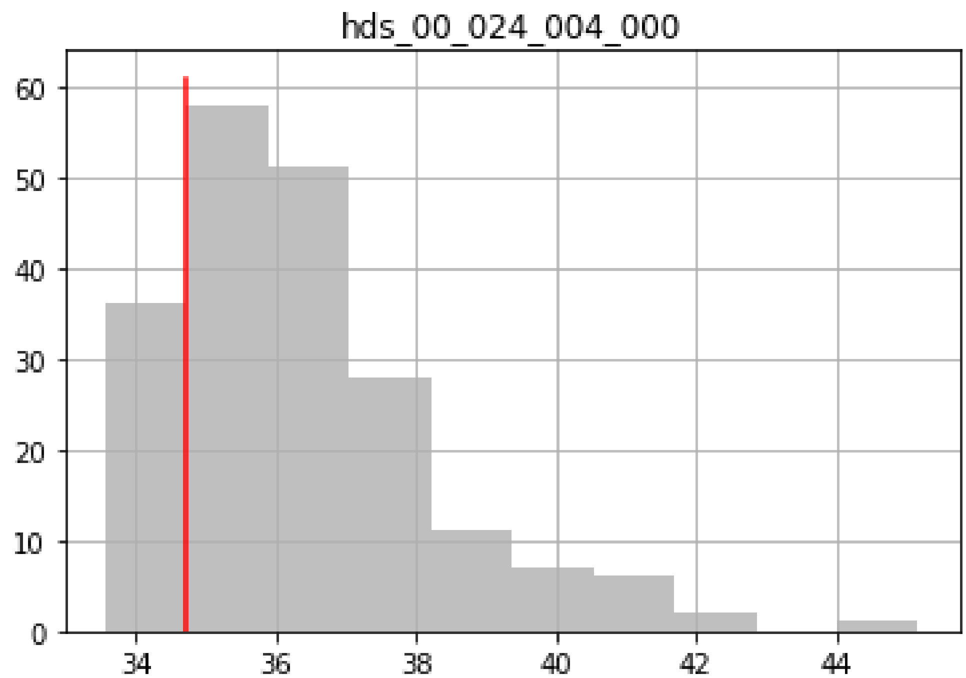

hds_00_009_001_000
0
10
20
30
40
50
34
36
38
40
42
44
46
48
50

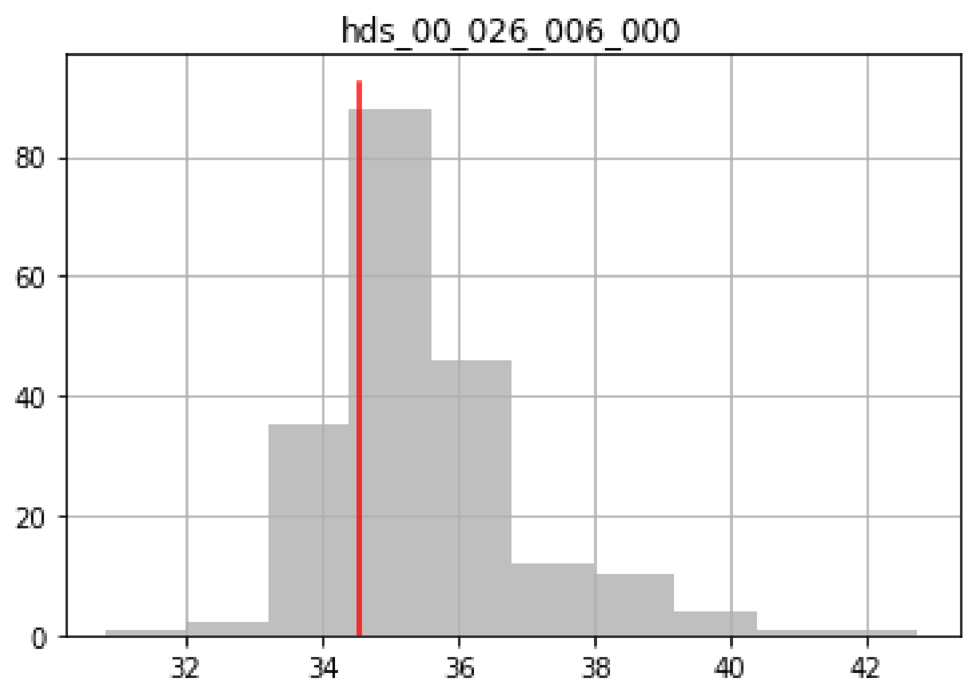

hds_00_013_010_000
0
20
40
60
80
32
33
34
35
36
37
38
39

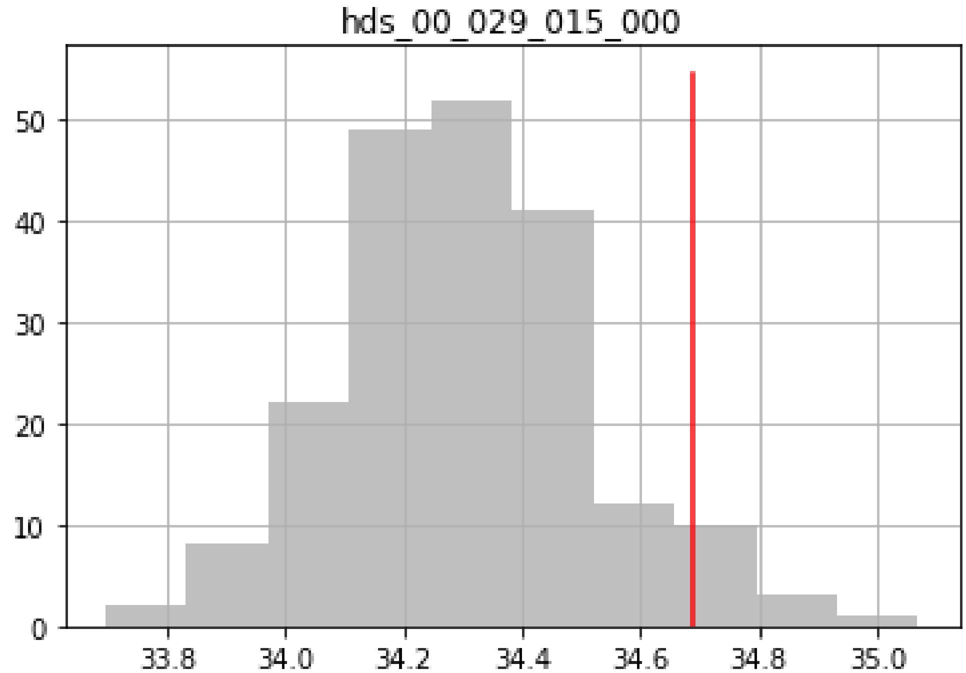

hds_00_015_016_000
0
10
20
30
40
50
60
70
80
33.0
33.5
34.0
34.5
35.0
35.5

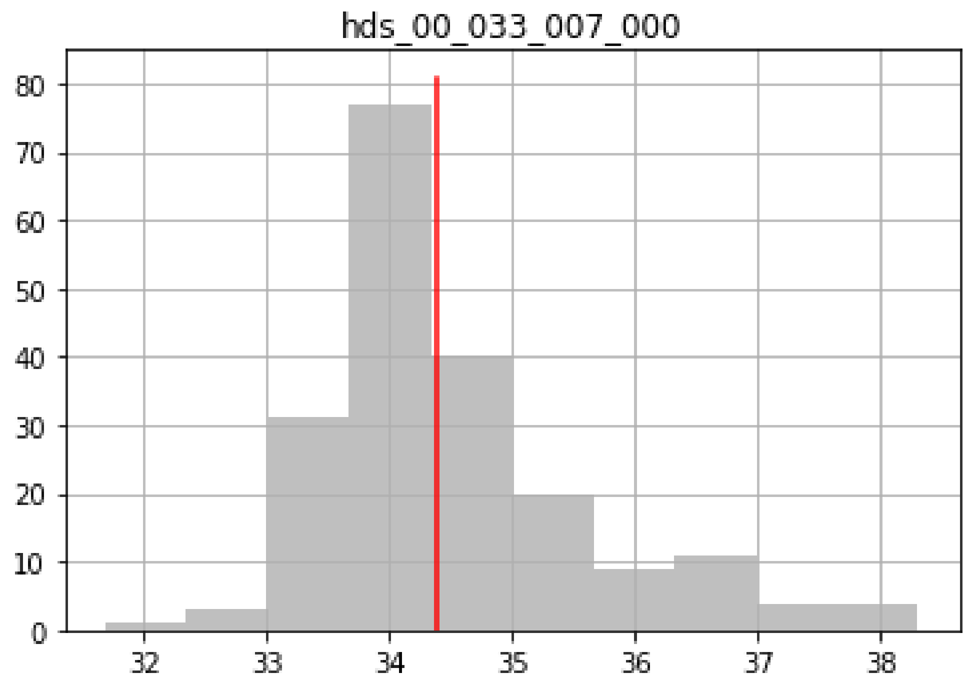

hds_00_021_010_000
0
10
20
30
40
50
60
70
33
34
35
36
37
38
39

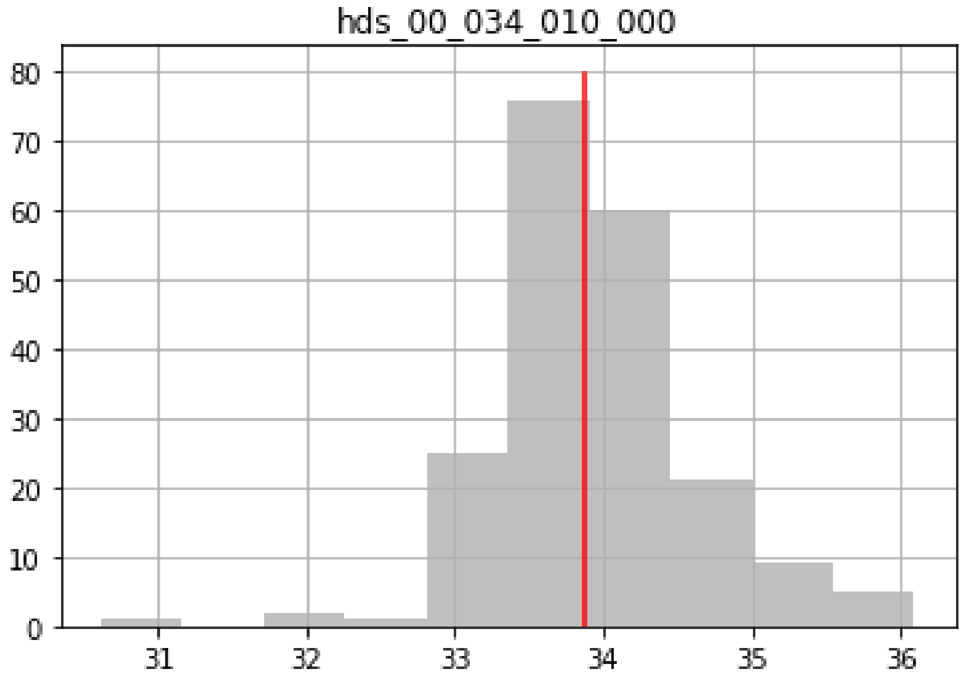

hds_00_022_015_000
0
10
20
30
40
50
60
70
33.8
34.0
34.2
34.4
34.6
34.8
35.0


hds_00_024_004_000
0
10
20
30
40
50
60
34
36
38
40
42
44


hds_00_026_006_000
0
20
40
60
80
32
34
36
38
40
42


hds_00_029_015_000
0
10
20
30
40
50
33.8
34.0
34.2
34.4
34.6
34.8
35.0


hds_00_033_007_000
0
10
20
30
40
50
60
70
80
32
33
34
35
36
37
38


hds_00_034_010_000
80
70
60
50
40
30
20
10
0
31
32
33
34
35
36