prior_montecarlo

May 11, 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
```

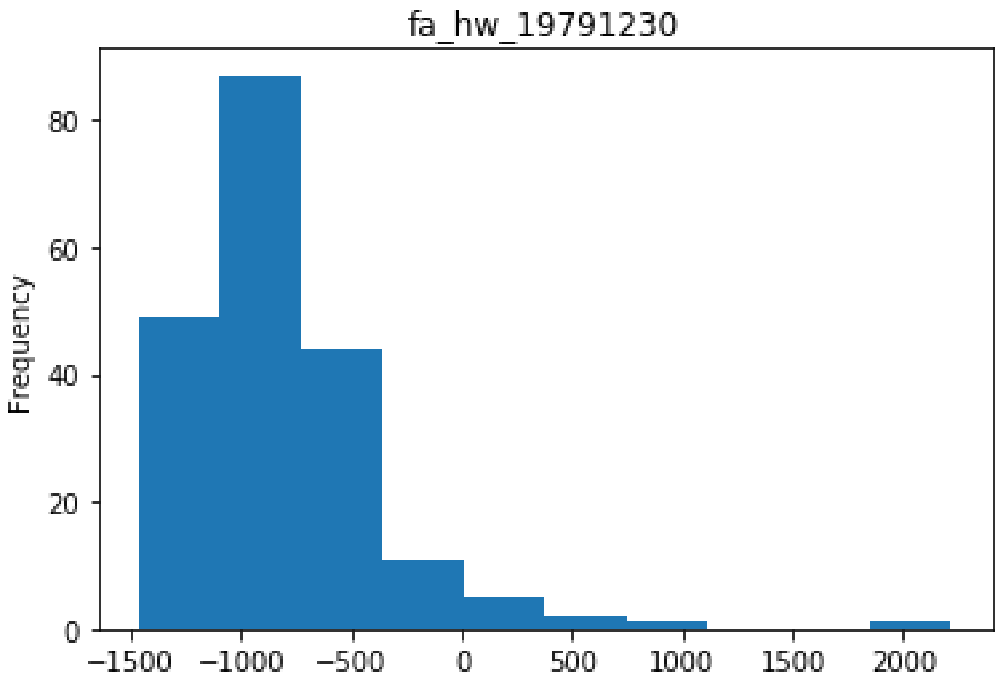
1.0.5 confirm which quantities were identified as forecasts

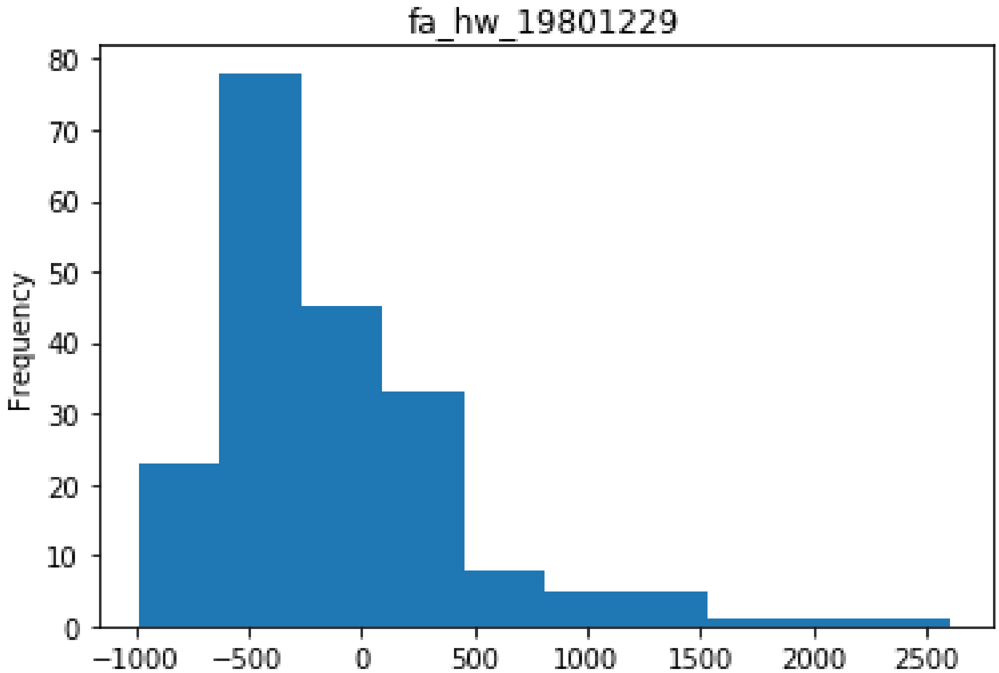
```
In [8]: fnames = pst.pestpp_options["forecasts"].split(',')
        fnames
```

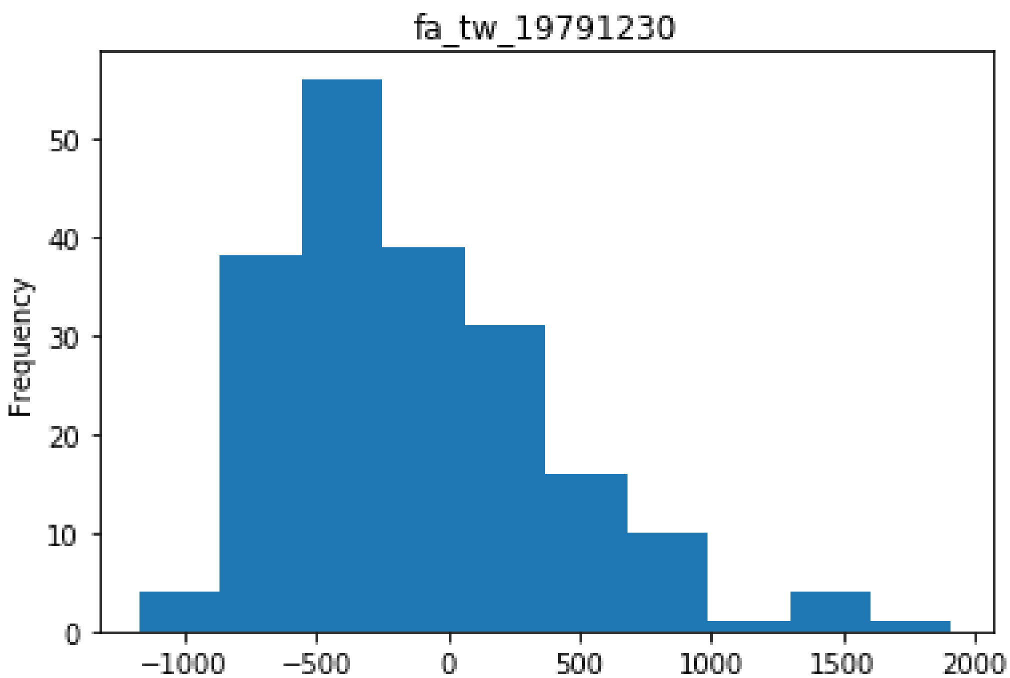
```
Out[8]: ['fa_hw_19791230',
         'fa_hw_19801229',
         'fa_tw_19791230',
         'fa_tw_19801229',
         'hds_00_013_002_000',
         'hds_00_013_002_001',
         'part_time',
         'part_status']
```

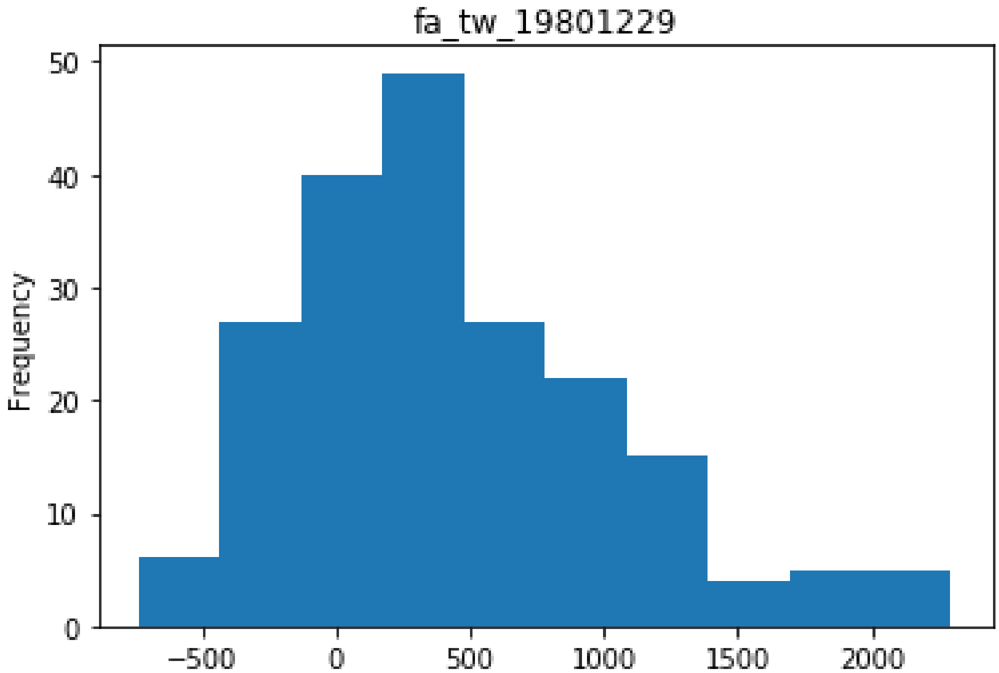
1.0.6 now we can plot the distributions of each forecast

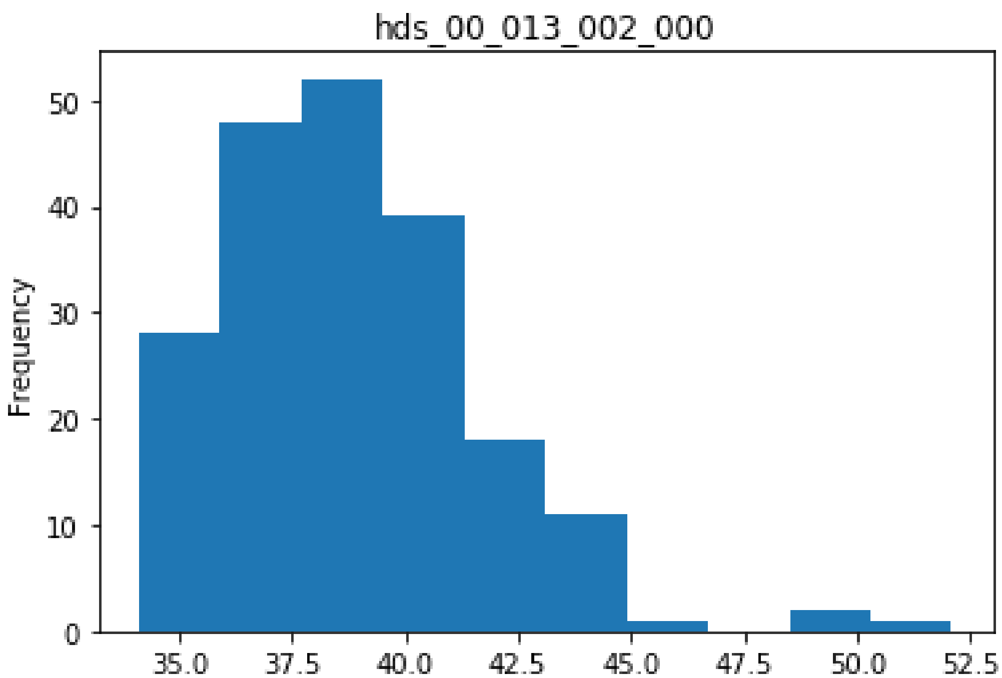
```
In [9]: for forecast in fnames:
            plt.figure()
            ax = obs_df.loc[:,forecast].plot(kind="hist")
            ax.set_title(forecast)
        plt.show()
```

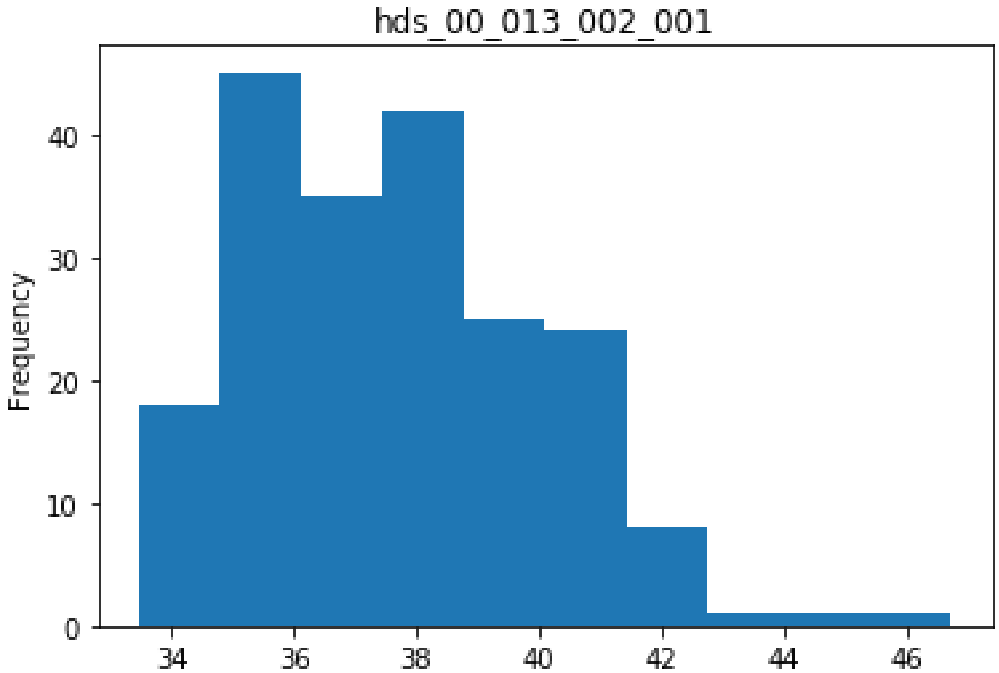


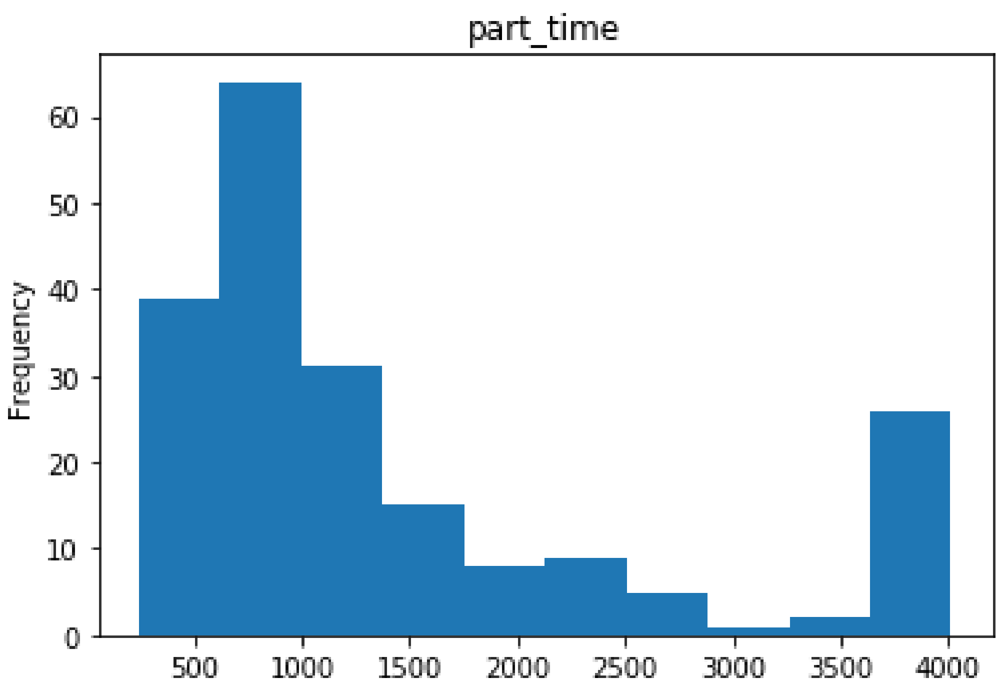

fa_hw_19801229
Frequency
80
70
60
50
40
30
20
10
0
−1000
−500
0
500
1000
1500
2000
2500

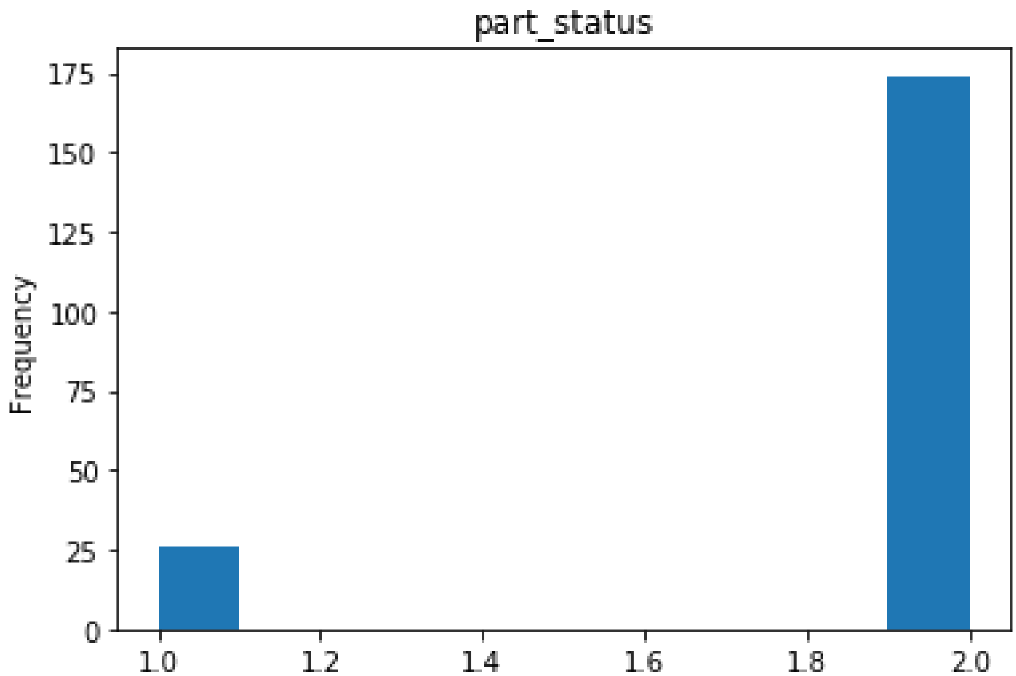

fa_tw_19791230
Frequency
50
40
30
20
10
0
−1000
−500
0
500
1000
1500
2000


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


hds_00_013_002_000
Frequency
50
40
30
20
10
0
35.0
37.5
40.0
42.5
45.0
47.5
50.0
52.5

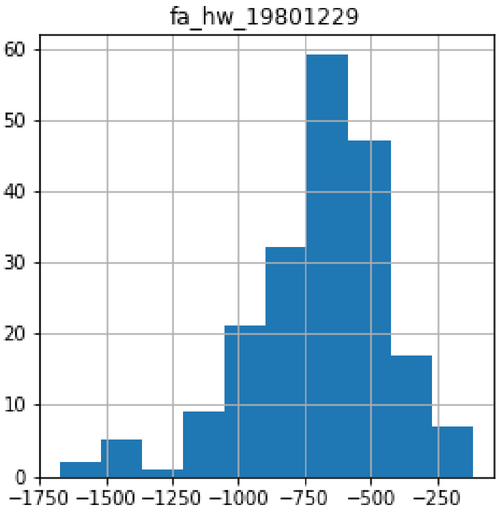

hds_00_013_002_001
Frequency
0
10
20
30
40
34
36
38
40
42
44
46

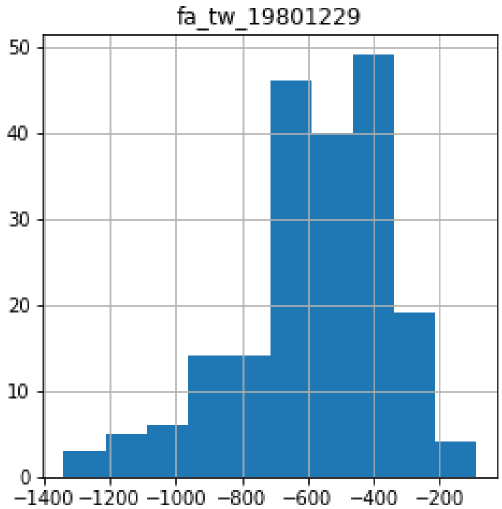

part_time
Frequency
0
10
20
30
40
50
60
500
1000
1500
2000
2500
3000
3500
4000

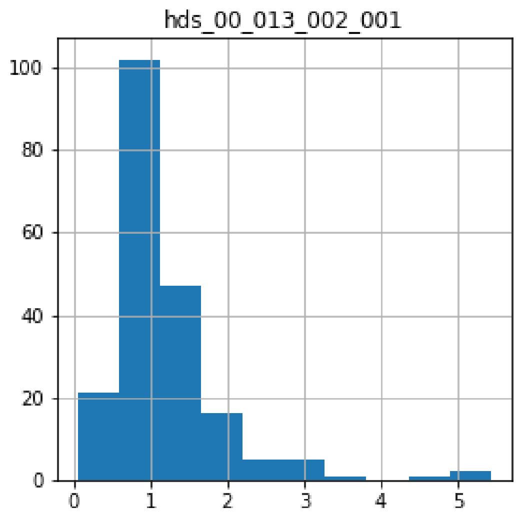


We see that under scenario conditions, many more realizations for the flow to the aquifer in the headwaters are postive (as expected). Lets difference these two:

```
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
        76       243.0713
        165      245.4841
        149      253.9859
        186      302.7784
        161      316.2889
        51       334.9378
        187      339.2126
        40       352.3095
        39       353.5649
        191      375.6852
        78       388.5967
        97       396.5381
        132      398.9588
        30       402.3243
        9        413.6731
        37       415.8716
        77       436.9160
        120      442.8409
        180      452.2230
        196      461.4423
        100      484.2079
        3        487.8409
        56       492.3636
        57       495.2174
        46       500.2236
        43       505.0375
        75       510.1112
        138      510.2362
        61       511.1681
        130      512.5389
                   ...
        42      2794.8200
        29      3083.3470
        179     3299.9120
        47      3306.0520
        27      4015.0000
        71      4015.0000
        36      4015.0000
        16      4015.0000
        194     4015.0000
        12      4015.0000
        6       4015.0000
        185     4015.0000
        19      4015.0000
```

```
159    4015.0000
82     4015.0000
160    4015.0000
107    4015.0000
108    4015.0000
117    4015.0000
173    4015.0000
118    4015.0000
125    4015.0000
131    4015.0000
169    4015.0000
153    4015.0000
154    4015.0000
166    4015.0000
157    4015.0000
73     4015.0000
0      4015.0000
Name: part_time, Length: 200, dtype: float64
```

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

```
Out[12]: fo_39_19791230        11693.000000
hds_00_002_009_000       41.737926
hds_00_002_015_000       34.961693
hds_00_003_008_000       42.291515
hds_00_009_001_000       43.447071
hds_00_013_010_000       37.594856
hds_00_015_016_000       34.993023
hds_00_021_010_000       37.685677
hds_00_022_015_000       34.358593
hds_00_024_004_000       41.239319
hds_00_026_006_000       39.924770
hds_00_029_015_000       34.695927
hds_00_033_007_000       36.626369
hds_00_034_010_000       34.268597
Name: 100, 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       -872.033000
fa_hw_19801229       -275.564050
fa_tw_19791230       -801.765690
fa_tw_19801229       -293.516600
hds_00_013_002_000     43.294212
hds_00_013_002_001     40.301617
part_time             484.207900
part_status             2.000000
Name: 100, 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

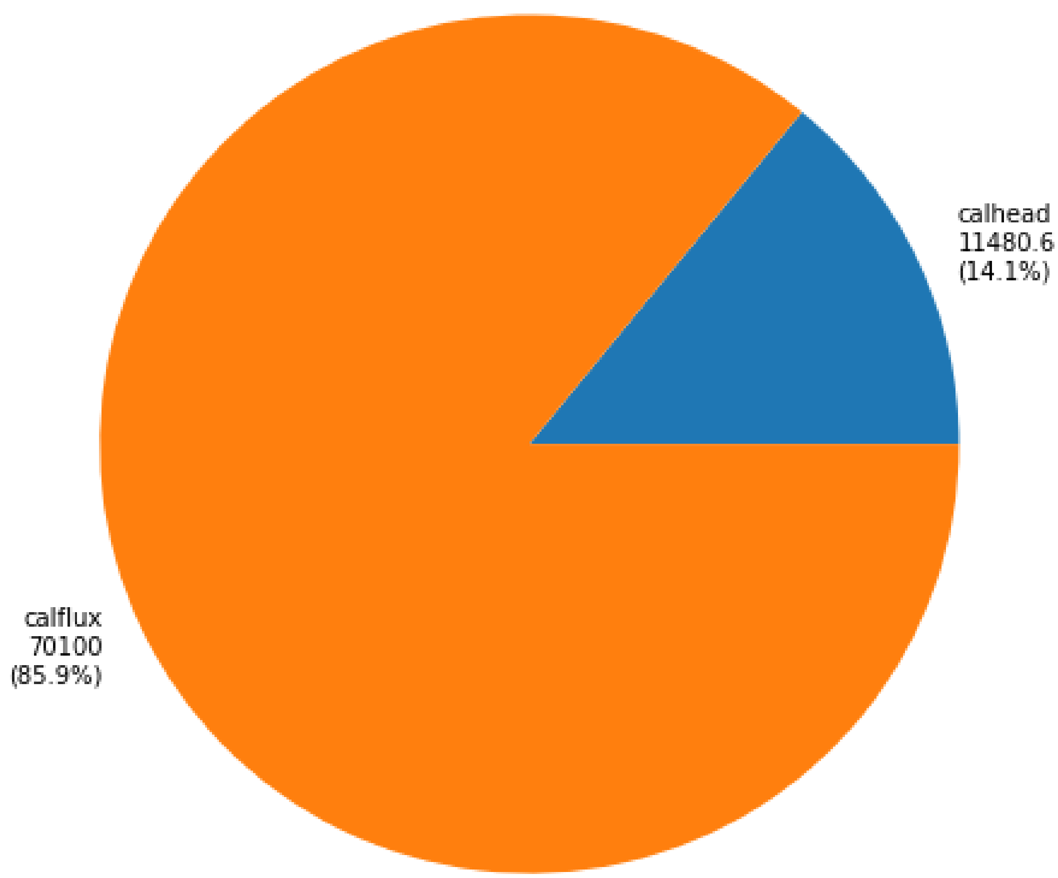
```
In [16]: pst.write(os.path.join(t_d,"freyberg.pst"))
         pyemu.os_utils.run("pestpp-ies freyberg.pst",cwd=t_d)

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

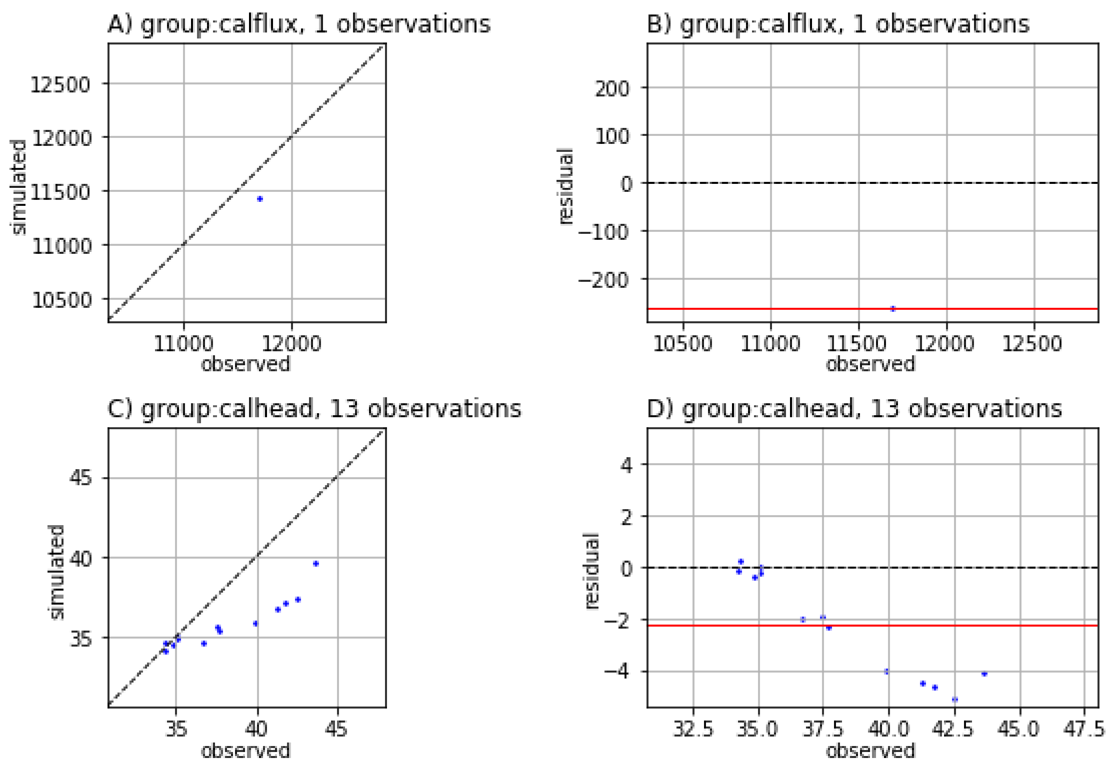
```
In [17]: pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
         print(pst.phi)
         plt.figure()
         pst.plot(kind='phi_pie')
         print('Here are the non-zero weighted observation names')

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

```
81580.60777411229
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  11694.764052  11430.000000
hds_00_002_009_000  hds_00_002_009_000  calhead     41.777942     37.107498
hds_00_002_015_000  hds_00_002_015_000  calhead     35.059567     35.045185
hds_00_003_008_000  hds_00_003_008_000  calhead     42.515605     37.397289
hds_00_009_001_000  hds_00_009_001_000  calhead     43.633827     39.546417
hds_00_013_010_000  hds_00_013_010_000  calhead     37.497128     35.571774
hds_00_015_016_000  hds_00_015_016_000  calhead     35.088032     34.835716
hds_00_021_010_000  hds_00_021_010_000  calhead     37.670541     35.386250
hds_00_022_015_000  hds_00_022_015_000  calhead     34.348271     34.577492
hds_00_024_004_000  hds_00_024_004_000  calhead     41.280379     36.760464
hds_00_026_006_000  hds_00_026_006_000  calhead     39.939175     35.896149
hds_00_029_015_000  hds_00_029_015_000  calhead     34.841354     34.453842
hds_00_033_007_000  hds_00_033_007_000  calhead     36.702473     34.678810
hds_00_034_010_000  hds_00_034_010_000  calhead     34.280764     34.118073

                      residual  weight
name
fo_39_19791230      264.764052     1.0
hds_00_002_009_000    4.670444    10.0
hds_00_002_015_000    0.014382    10.0
hds_00_003_008_000    5.118315    10.0
hds_00_009_001_000    4.087410    10.0
hds_00_013_010_000    1.925355    10.0
hds_00_015_016_000    0.252316    10.0
hds_00_021_010_000    2.284291    10.0
hds_00_022_015_000   -0.229221    10.0
hds_00_024_004_000    4.519915    10.0
hds_00_026_006_000    4.043026    10.0
hds_00_029_015_000    0.387512    10.0
hds_00_033_007_000    2.023663    10.0
hds_00_034_010_000    0.162692    10.0
```

Publication ready figs - oh snap!

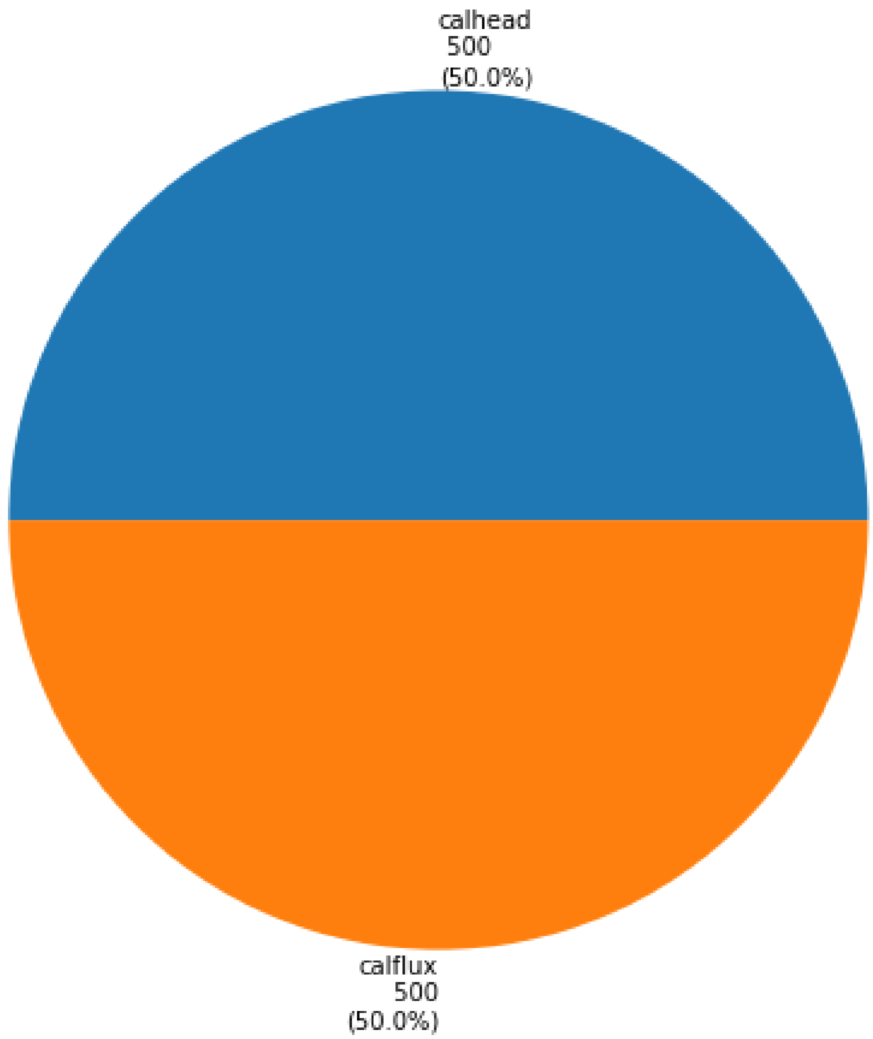
Depending on the truth you chose, we may have a problem - we set the weights for both the heads and the flux to reasonable values based on what we expect for measurement noise. But the contributions to total phi might be out of balance - if contribution of the flux measurement to total phi is too low, 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]:

```python
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 0x18194315c0>



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        0.084455
hds_00_002_009_000    2.086905
hds_00_002_015_000    2.086905
hds_00_003_008_000    2.086905
```

```
hds_00_009_001_000    2.086905
hds_00_013_010_000    2.086905
hds_00_015_016_000    2.086905
hds_00_021_010_000    2.086905
hds_00_022_015_000    2.086905
hds_00_024_004_000    2.086905
hds_00_026_006_000    2.086905
hds_00_029_015_000    2.086905
hds_00_033_007_000    2.086905
hds_00_034_010_000    2.086905
Name: weight, dtype: float64
```

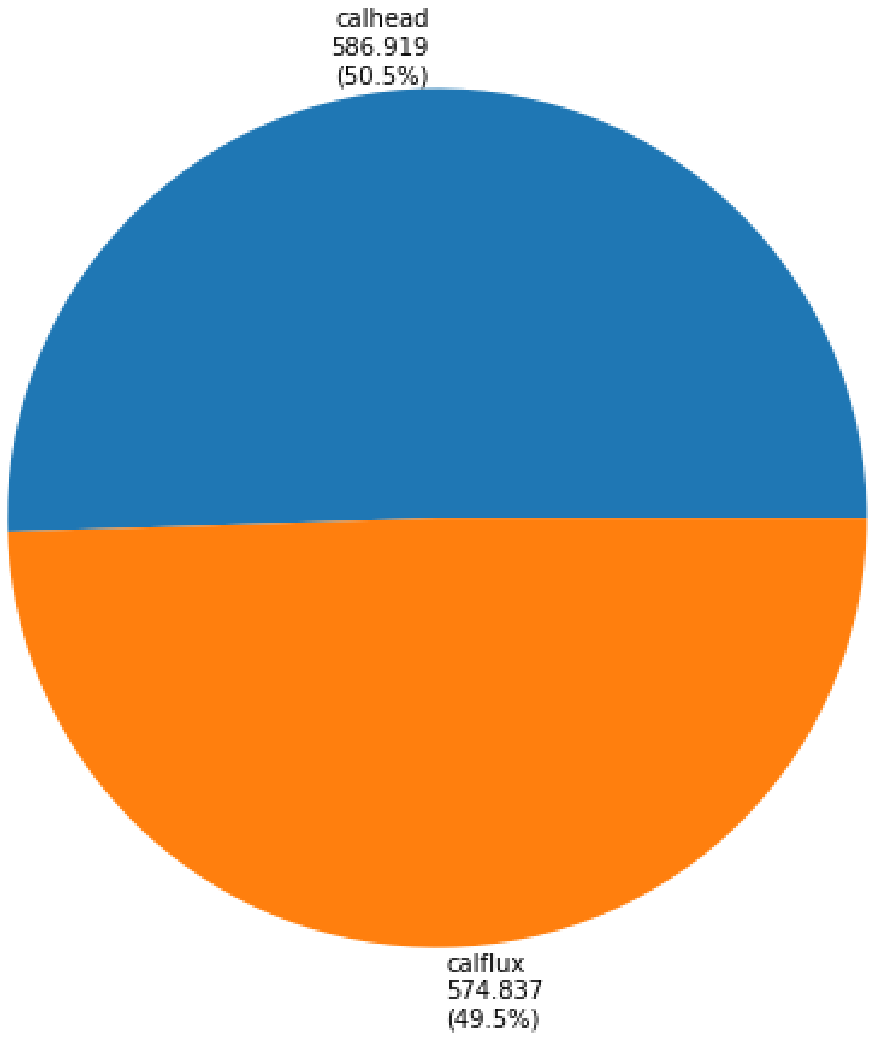
Now, for some super trickery: since we changed the weight, we need to generate the observation noise using these new weights for the error model (so meta!)

```
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        20.887453
         hds_00_002_009_000     0.191747
         hds_00_002_015_000     0.468990
         hds_00_003_008_000     1.073788
         hds_00_009_001_000     0.894894
         hds_00_013_010_000    -0.468291
         hds_00_015_016_000     0.455262
         hds_00_021_010_000    -0.072527
         hds_00_022_015_000    -0.049460
         hds_00_024_004_000     0.196750
         hds_00_026_006_000     0.069023
         hds_00_029_015_000     0.696857
         hds_00_033_007_000     0.364673
         hds_00_034_010_000     0.058304
         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
1161.7558803877937
```



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,:]
```

```
17.528847228089642
```

Out[23]:
```
                                name    group      measured      modelled  \
name
fo_39_19791230        fo_39_19791230  calflux  11713.887453  11693.000000
hds_00_002_009_000  hds_00_002_009_000  calhead     41.929673     41.737926
hds_00_002_015_000  hds_00_002_015_000  calhead     35.430683     34.961693
hds_00_003_008_000  hds_00_003_008_000  calhead     43.365303     42.291515
hds_00_009_001_000  hds_00_009_001_000  calhead     44.341965     43.447071
hds_00_013_010_000  hds_00_013_010_000  calhead     37.126566     37.594856
hds_00_015_016_000  hds_00_015_016_000  calhead     35.448285     34.993023
hds_00_021_010_000  hds_00_021_010_000  calhead     37.613149     37.685677
hds_00_022_015_000  hds_00_022_015_000  calhead     34.309133     34.358593
hds_00_024_004_000  hds_00_024_004_000  calhead     41.436069     41.239319
hds_00_026_006_000  hds_00_026_006_000  calhead     39.993793     39.924770
hds_00_029_015_000  hds_00_029_015_000  calhead     35.392783     34.695927
hds_00_033_007_000  hds_00_033_007_000  calhead     36.991042     36.626369
hds_00_034_010_000  hds_00_034_010_000  calhead     34.326901     34.268597

                     residual    weight
name
fo_39_19791230      20.887453  0.084455
hds_00_002_009_000   0.191747  2.086905
hds_00_002_015_000   0.468990  2.086905
hds_00_003_008_000   1.073788  2.086905
hds_00_009_001_000   0.894894  2.086905
hds_00_013_010_000  -0.468291  2.086905
hds_00_015_016_000   0.455262  2.086905
hds_00_021_010_000  -0.072527  2.086905
hds_00_022_015_000  -0.049460  2.086905
hds_00_024_004_000   0.196750  2.086905
hds_00_026_006_000   0.069023  2.086905
hds_00_029_015_000   0.696857  2.086905
hds_00_033_007_000   0.364673  2.086905
hds_00_034_010_000   0.058304  2.086905
```

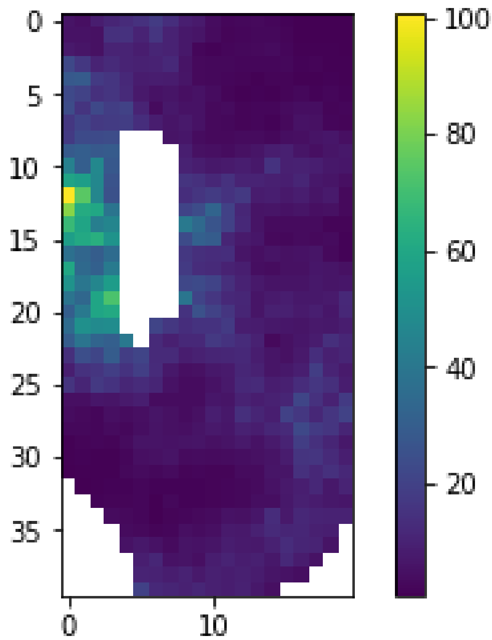
The residual should be exactly the noise values from above.Lets load the model (that was just run using the true pars) and check some things

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

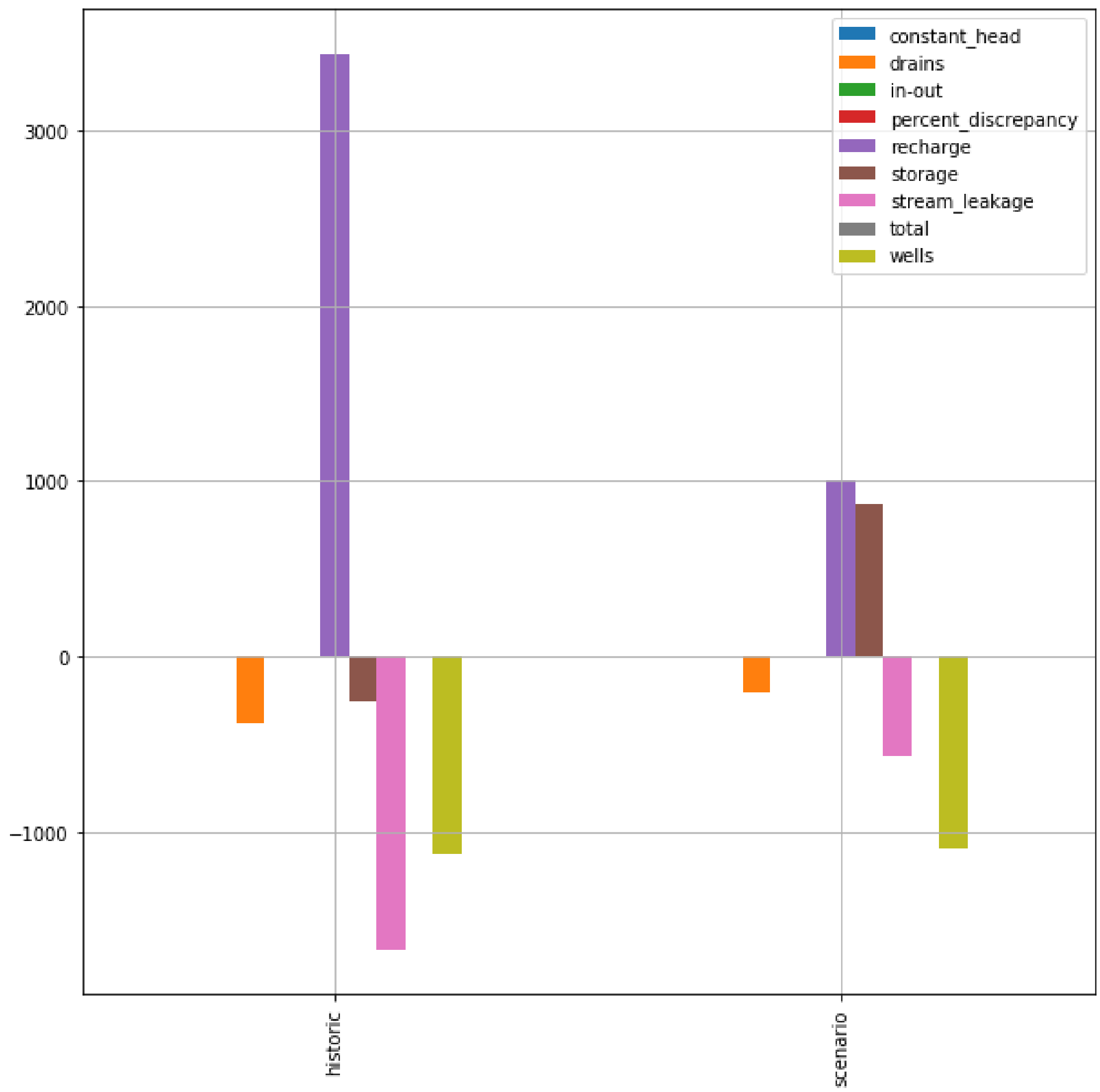
In [25]:
```
a = m.upw.hk[0].array
#a = m.rch.rech[0].array
```

```
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()
```

```
0.6918969 100.5342
```



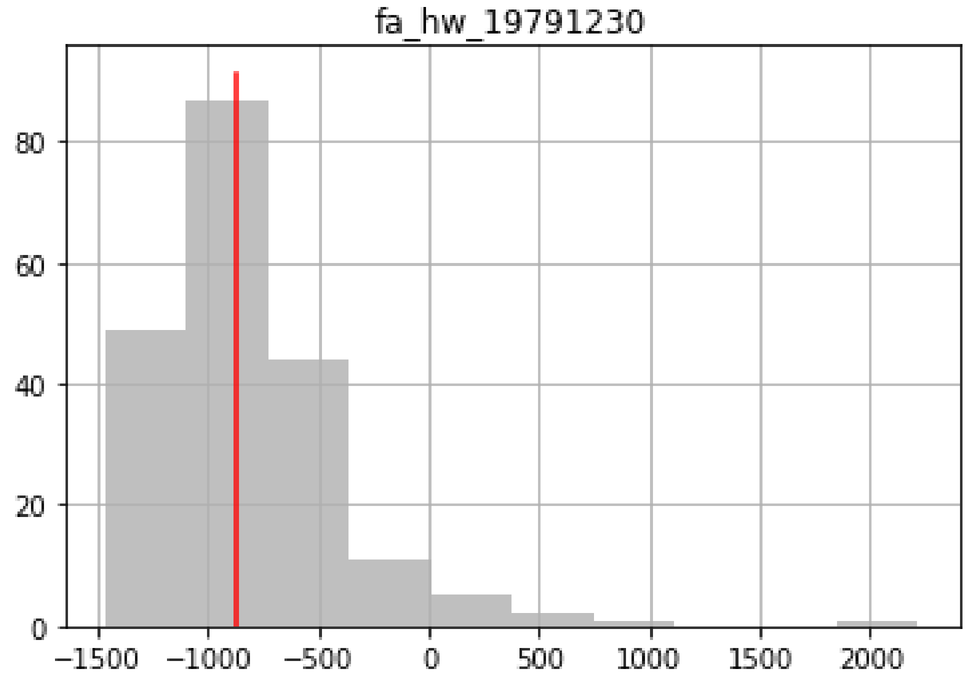

```
In [26]: lst = flopy.utils.MfListBudget(os.path.join(m_d,"freyberg.list"))
         df = lst.get_dataframes(diff=True)[0]
         ax = df.plot(kind="bar",figsize=(10,10), grid=True)
         a = ax.set_xticklabels(["historic","scenario"],rotation=90)
         plt.show(ax)
```

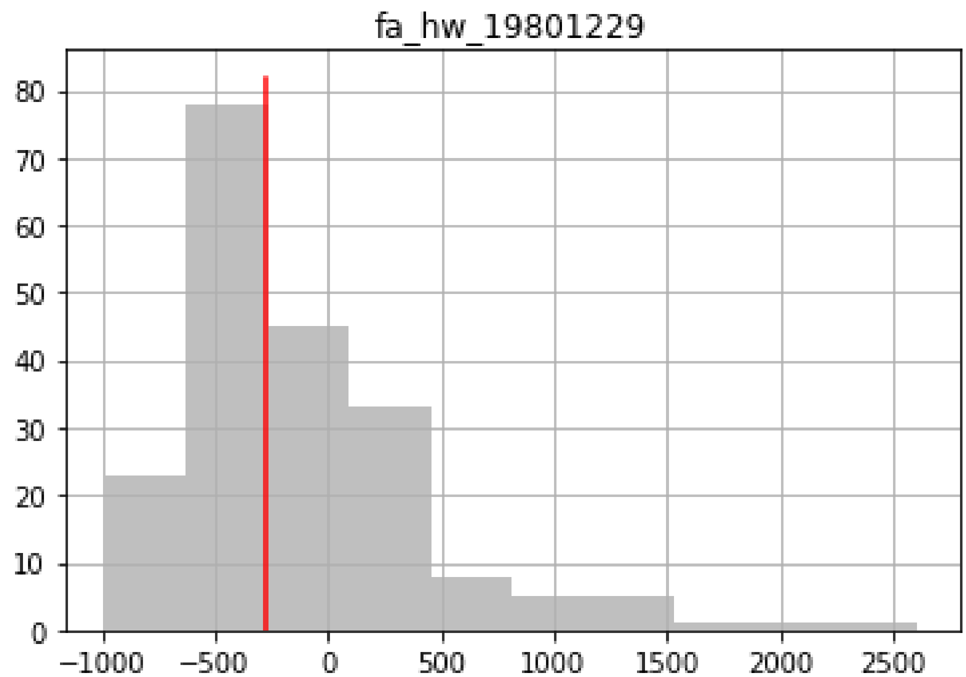


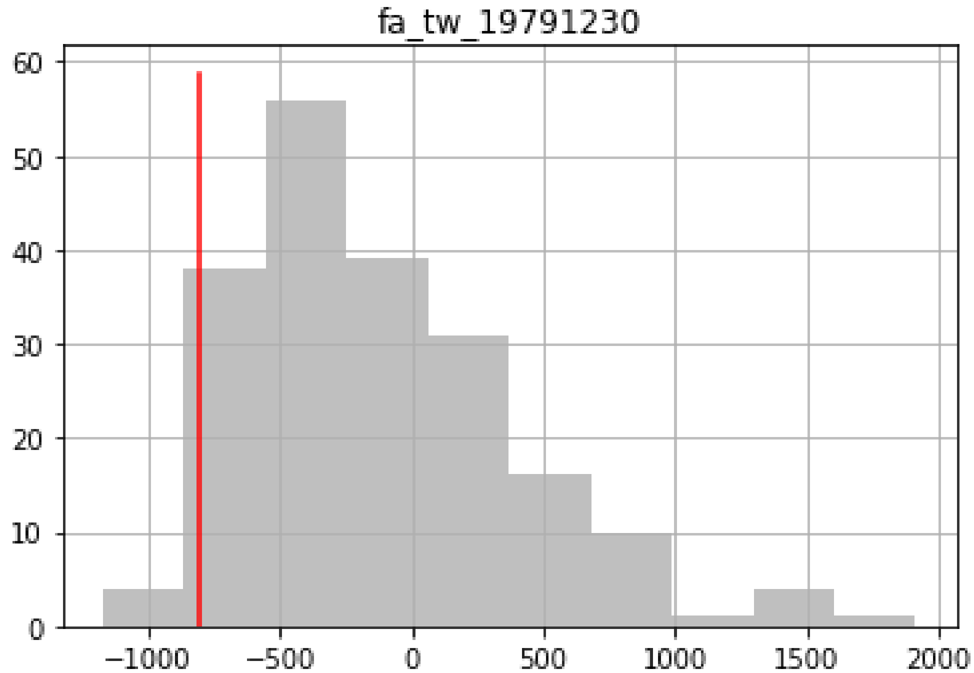
1.0.8 see how our existing observation ensemble compares to the truth

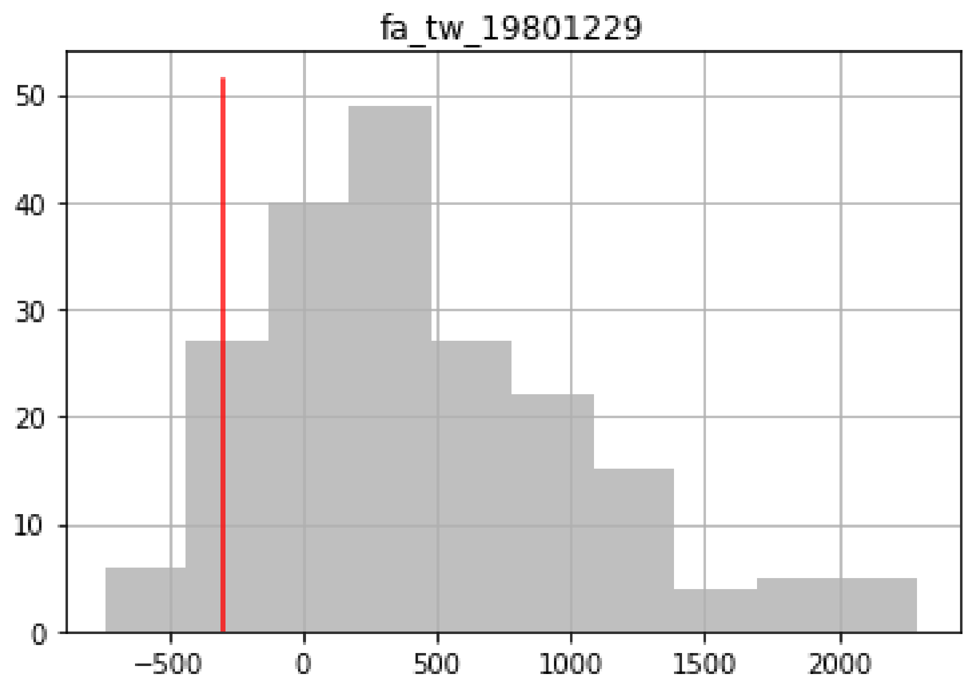
forecasts:

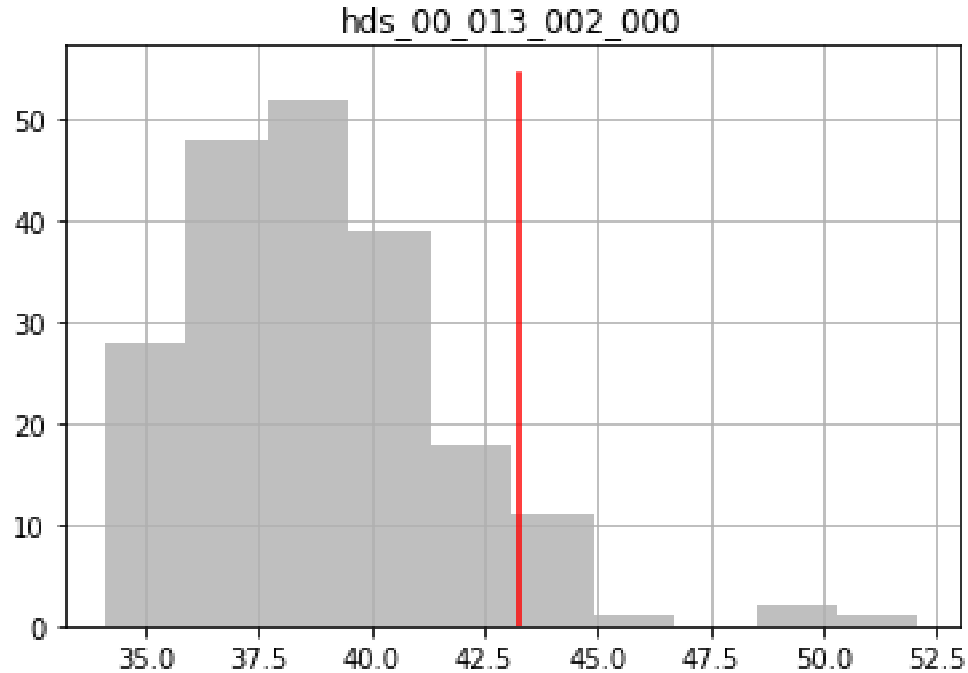
```
In [27]: obs = pst.observation_data
         plt.figure()
         for forecast in fnames:
             ax = plt.subplot(111)
             obs_df.loc[:,forecast].hist(ax=ax,color="0.5",alpha=0.5)
             ax.plot([obs.loc[forecast,"obsval"],obs.loc[forecast,"obsval"]],ax.get_ylim(),"r"
     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

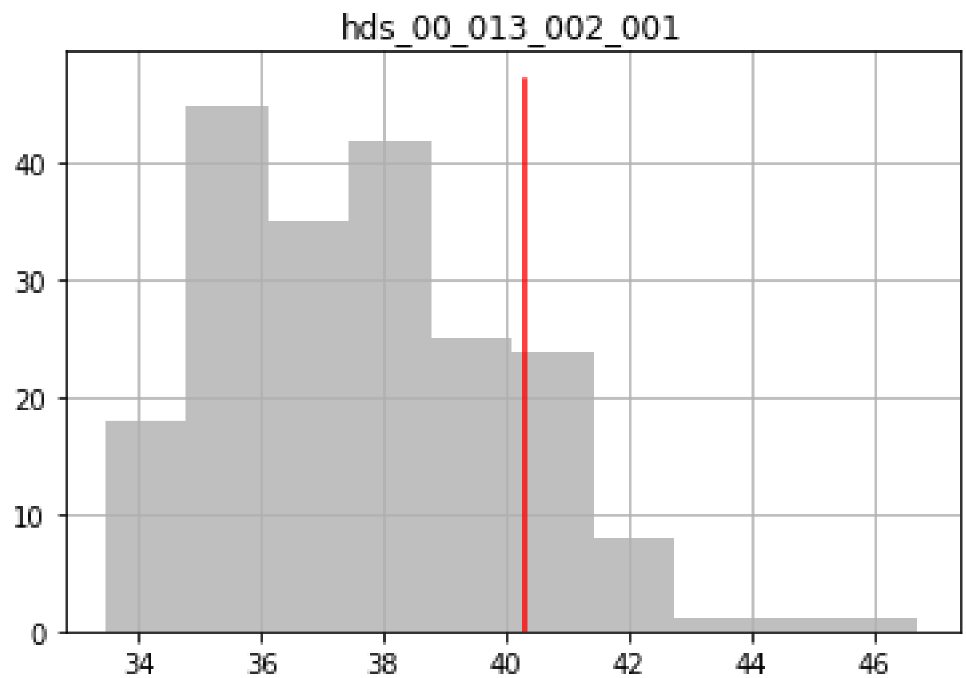

fa_hw_19791230
80
60
40
20
0
−1500
−1000
−500
0
500
1000
1500
2000

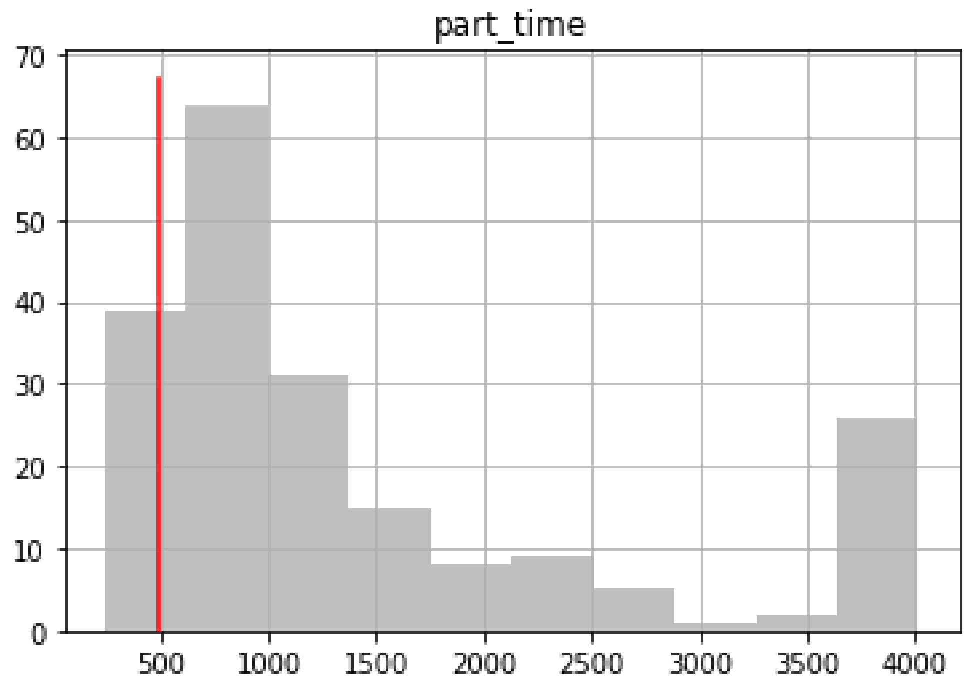

fa_hw_19801229
80
70
60
50
40
30
20
10
0
−1000
−500
0
500
1000
1500
2000
2500

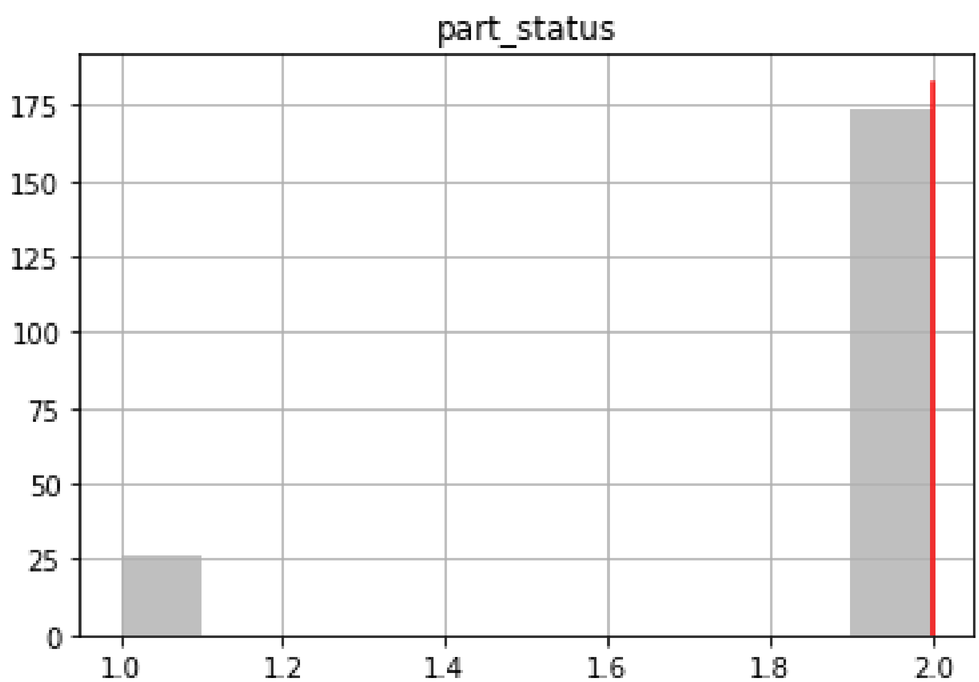

fa_tw_19791230
0
10
20
30
40
50
60
−1000
−500
0
500
1000
1500
2000


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


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
52.5

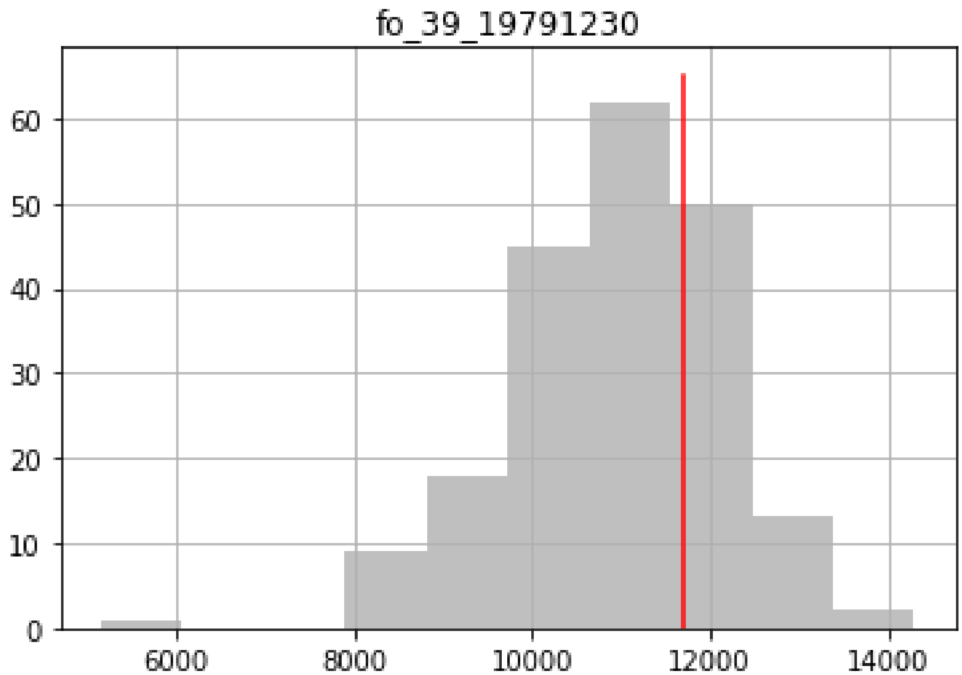

hds_00_013_002_001
0
10
20
30
40
34
36
38
40
42
44
46

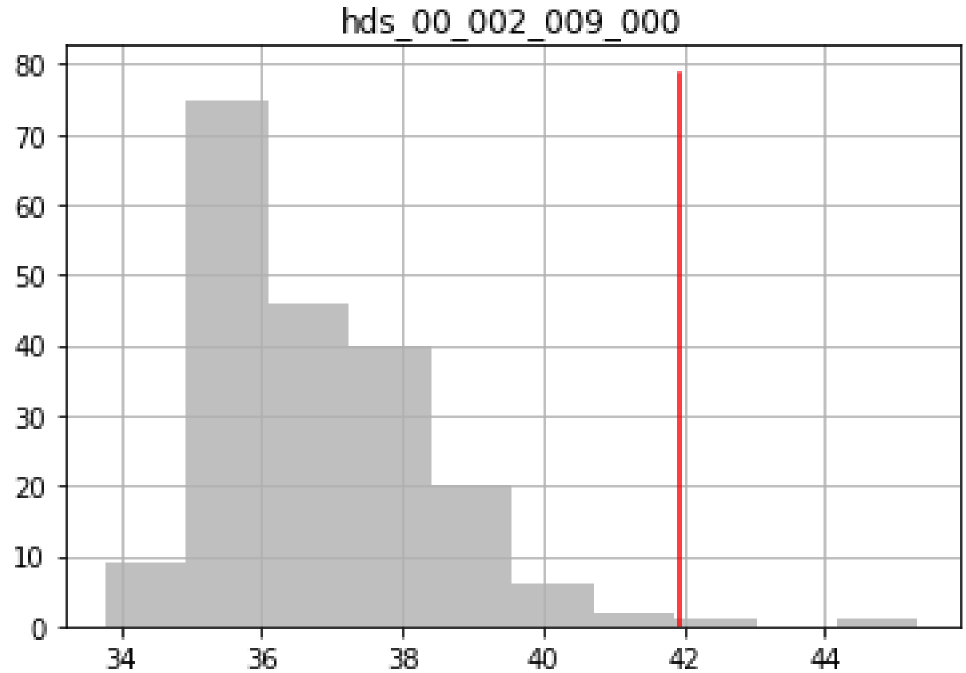


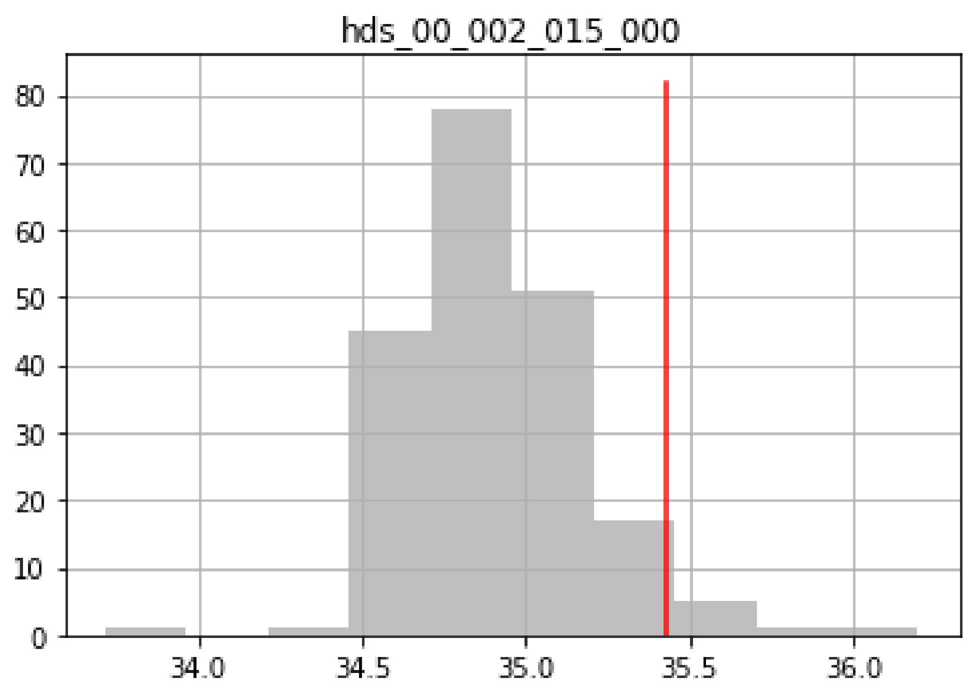


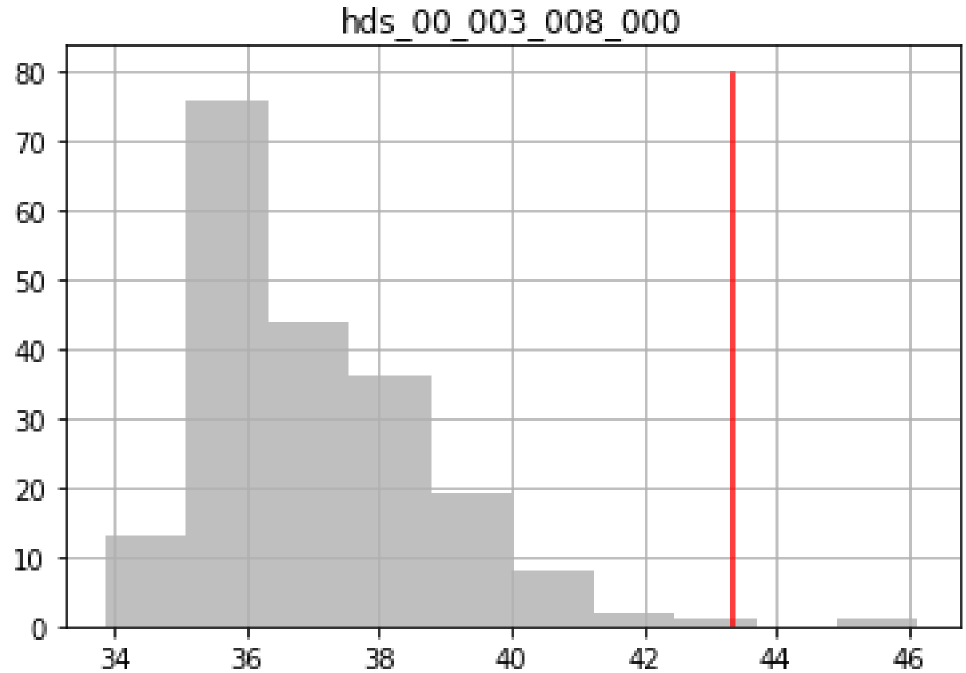
observations:

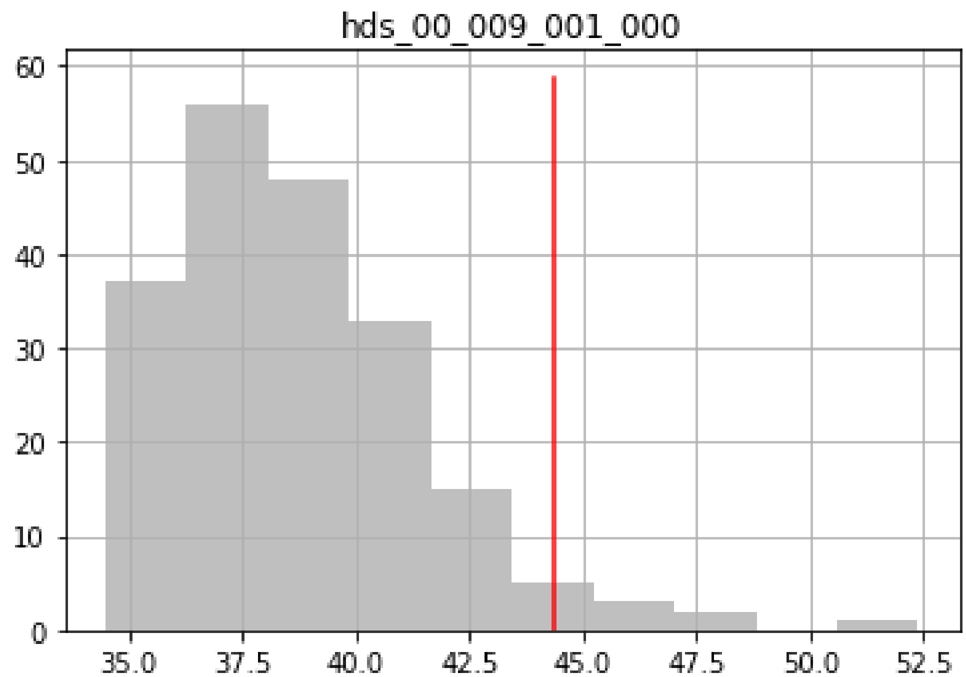
In [28]:
```
for oname in pst.nnz_obs_names:
    ax = plt.subplot(111)
    obs_df.loc[:,oname].hist(ax=ax,color="0.5",alpha=0.5)
    ax.plot([obs.loc[oname,"obsval"],obs.loc[oname,"obsval"]],ax.get_ylim(),"r")
    ax.set_title(oname)
    plt.show()
```

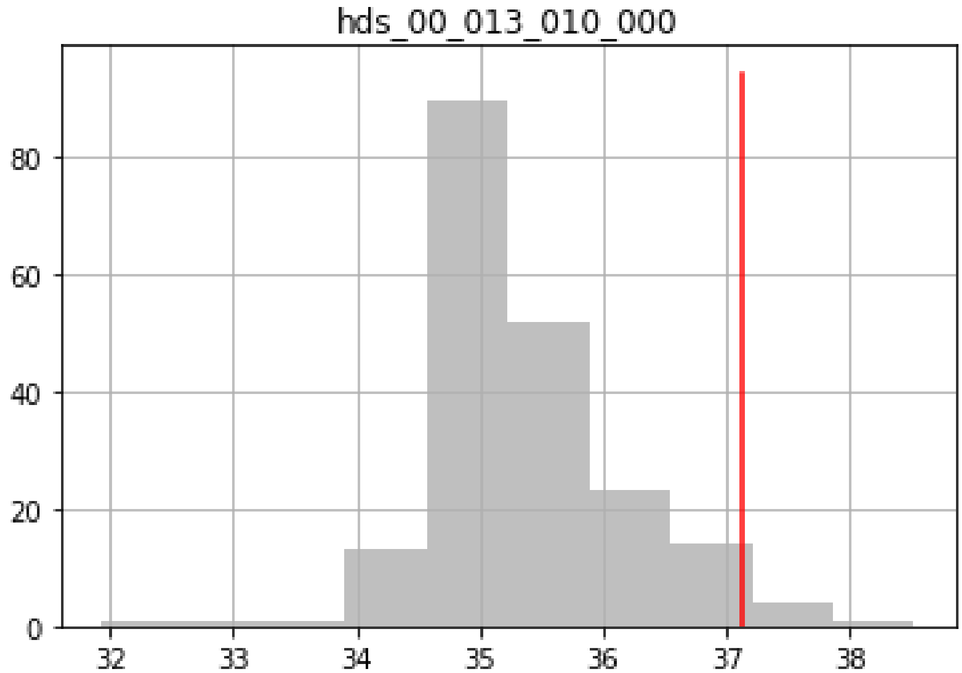


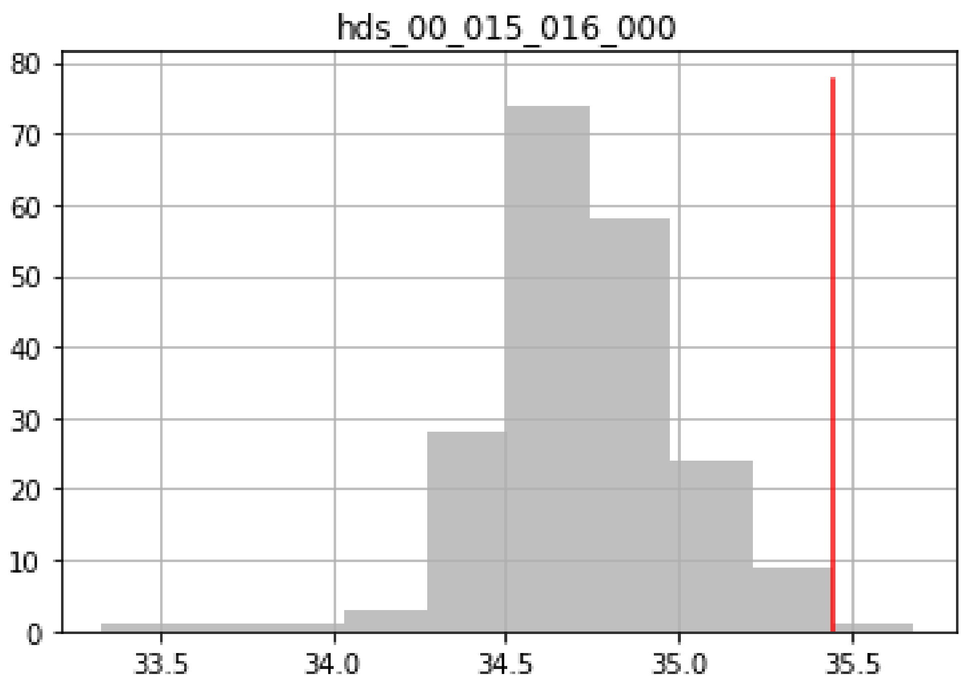

hds_00_002_009_000
80
70
60
50
40
30
20
10
0
34
36
38
40
42
44

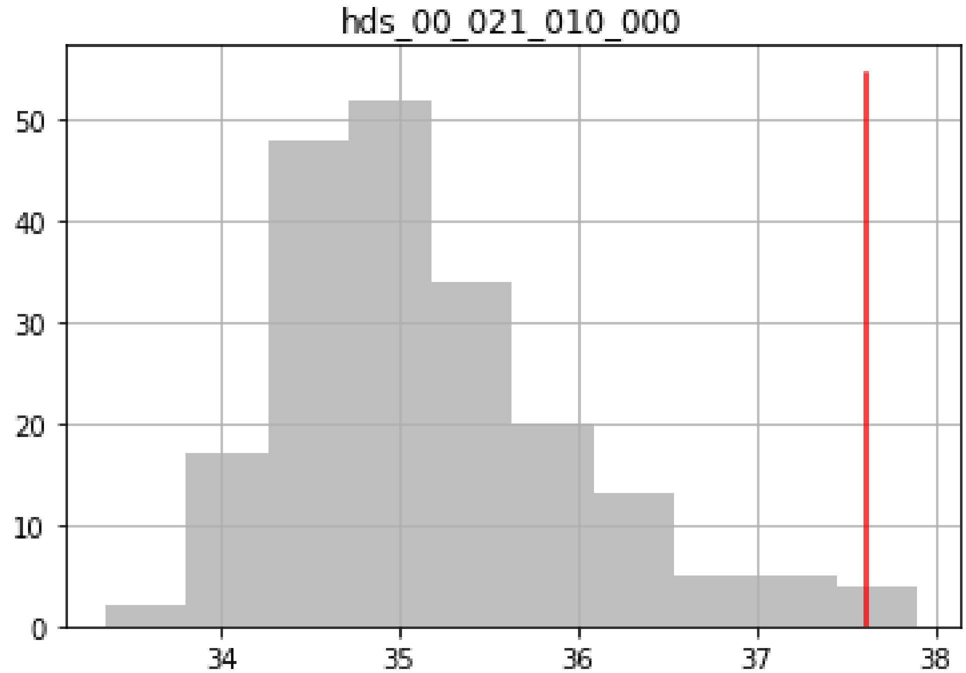

hds_00_002_015_000
80
70
60
50
40
30
20
10
0
34.0
34.5
35.0
35.5
36.0

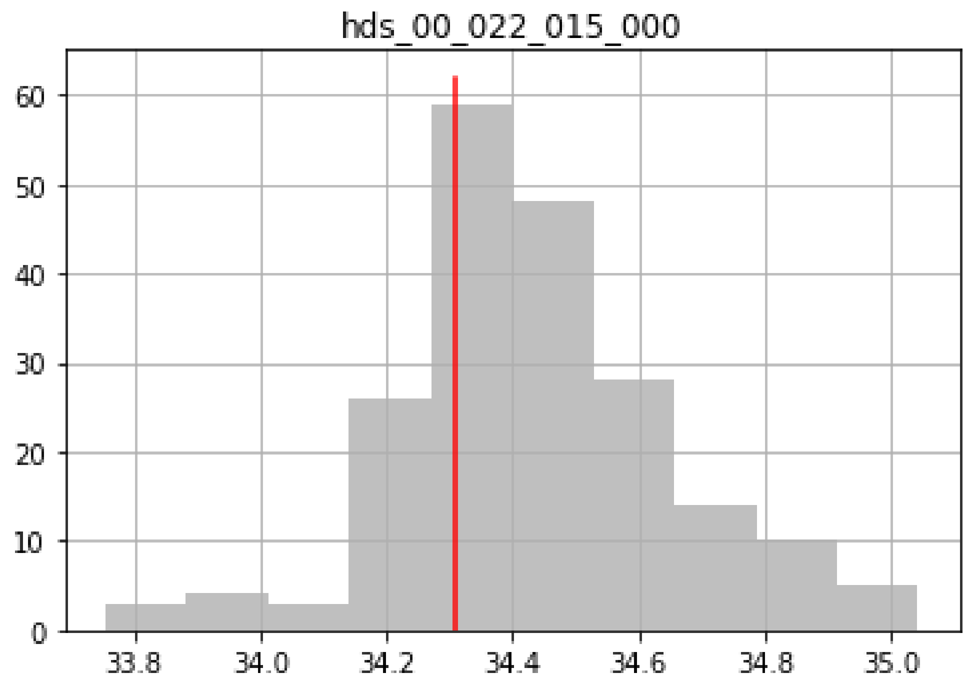

hds_00_003_008_000
80
70
60
50
40
30
20
10
0
34
36
38
40
42
44
46

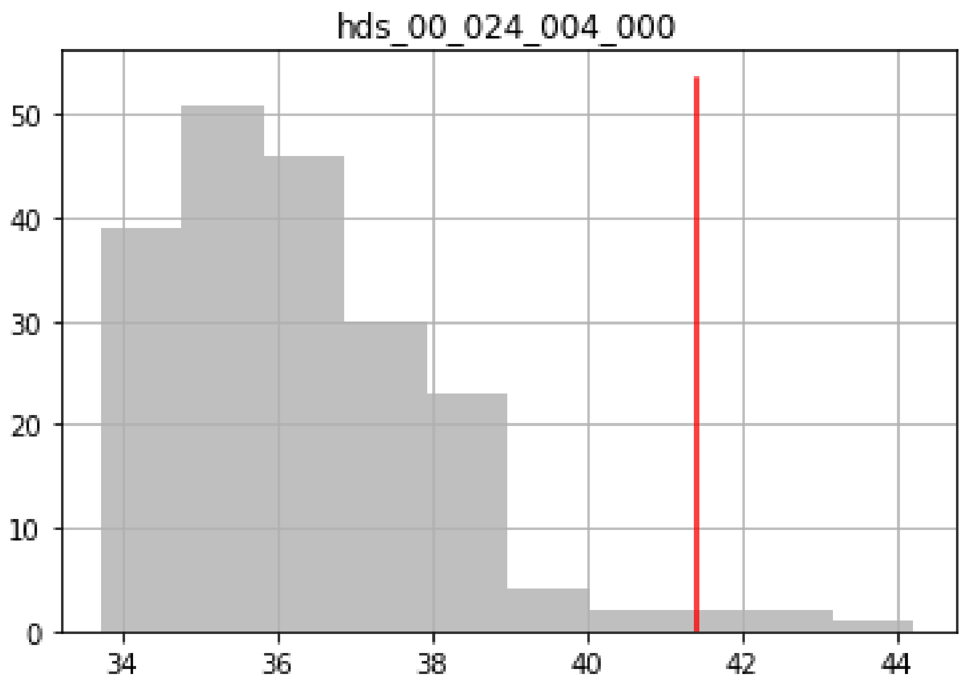

hds_00_009_001_000
60
50
40
30
20
10
0
35.0
37.5
40.0
42.5
45.0
47.5
50.0
52.5

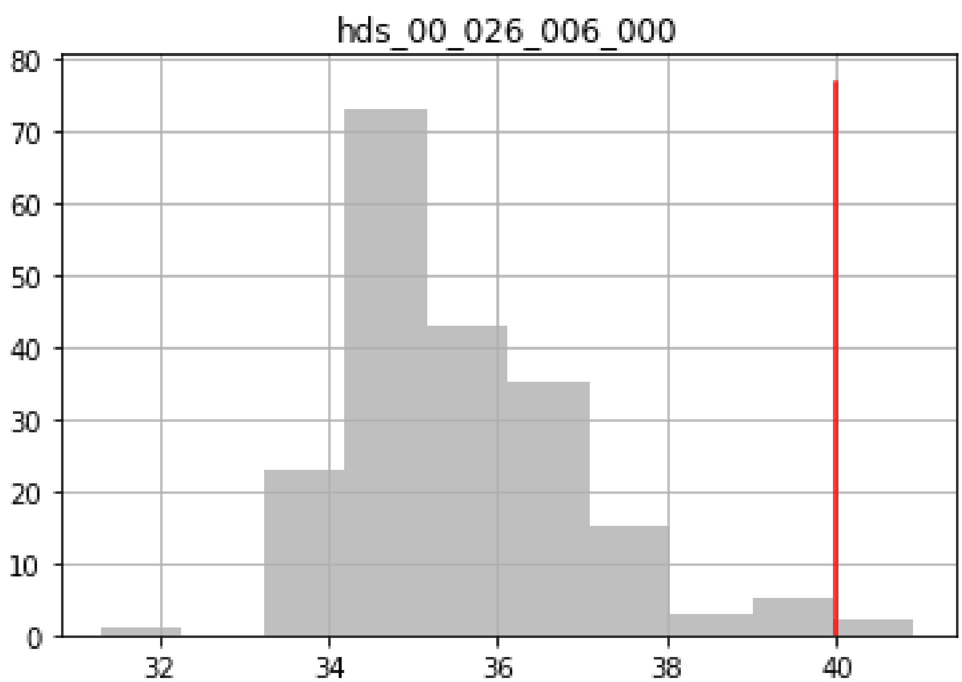

hds_00_013_010_000
0
20
40
60
80
32
33
34
35
36
37
38

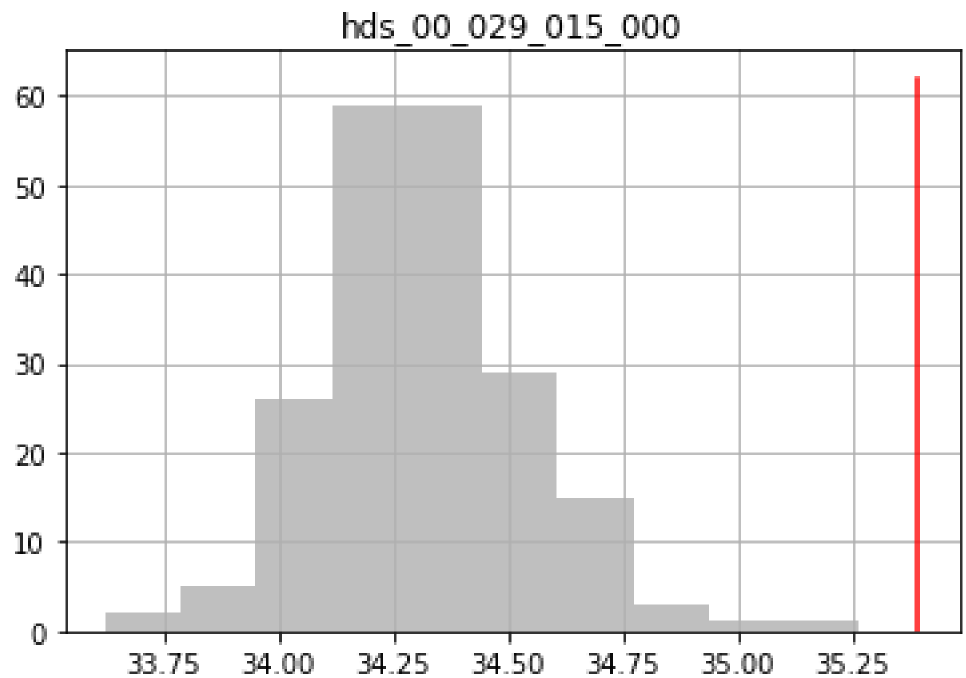

hds_00_015_016_000
0
10
20
30
40
50
60
70
80
33.5
34.0
34.5
35.0
35.5

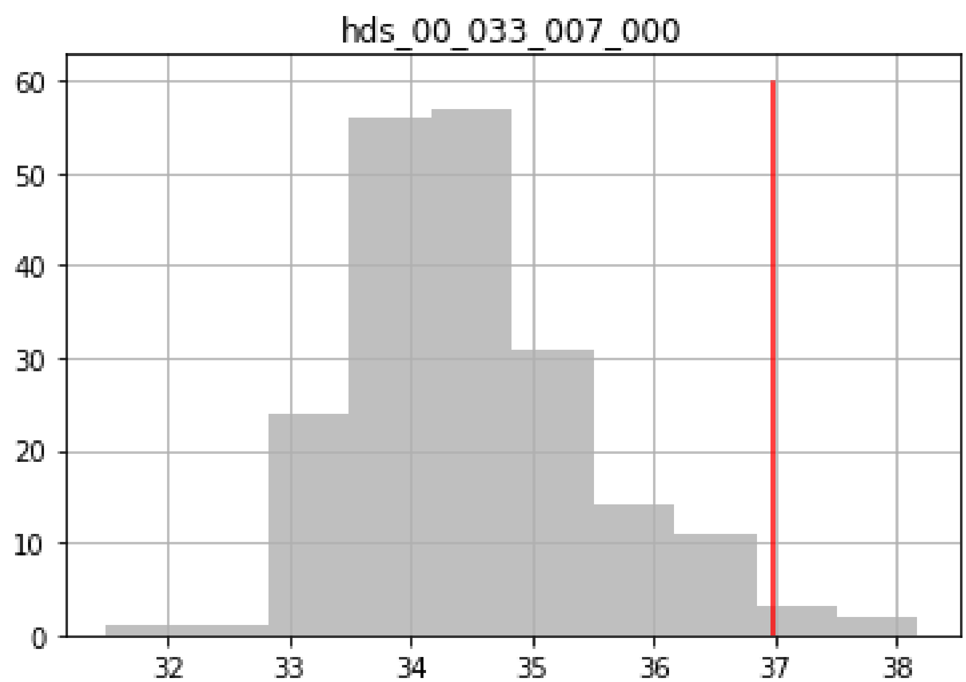

hds_00_021_010_000
50
40
30
20
10
0
34
35
36
37
38

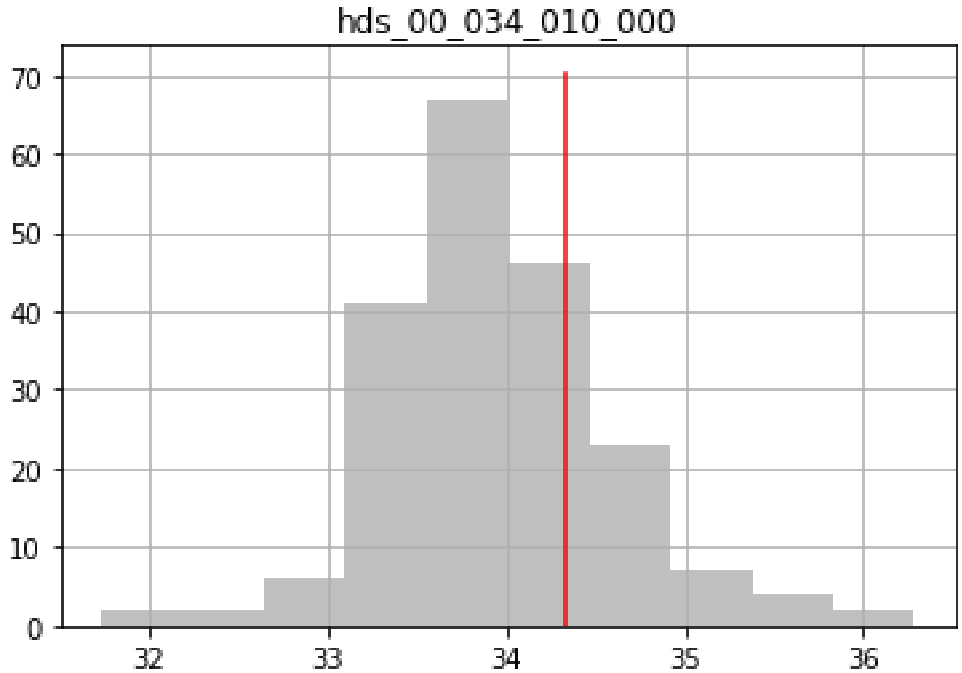

hds_00_022_015_000
60
50
40
30
20
10
0
33.8
34.0
34.2
34.4
34.6
34.8
35.0


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


hds_00_026_006_000
0
10
20
30
40
50
60
70
80
32
34
36
38
40


hds_00_029_015_000
60
50
40
30
20
10
0
33.75
34.00
34.25
34.50
34.75
35.00
35.25


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


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