prior_montecarlo

May 8, 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
```

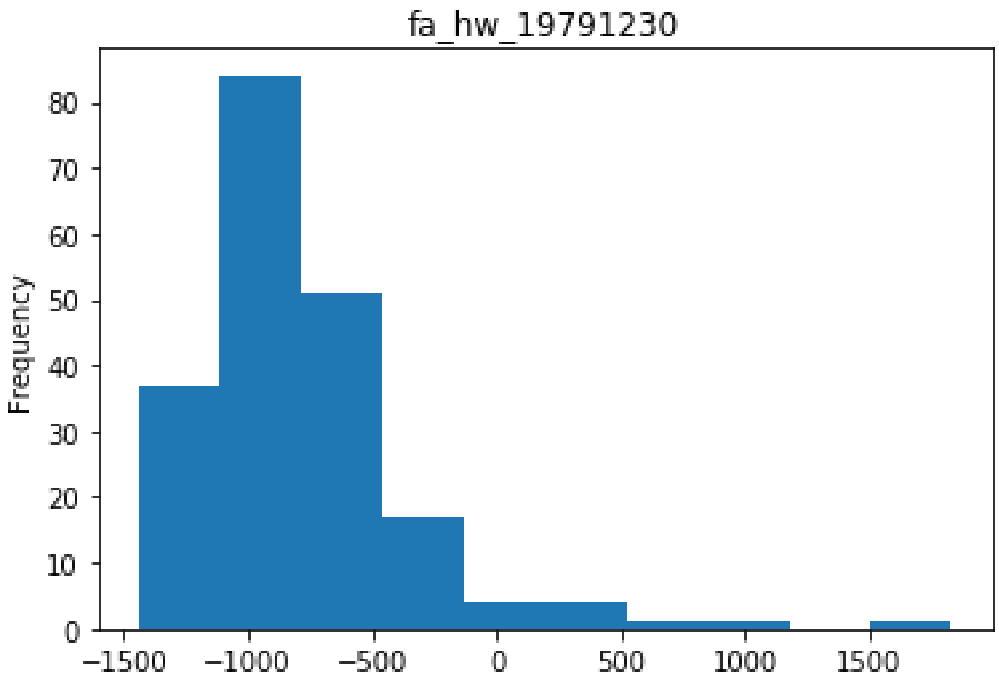
1.0.5 confirm which quantities were identified as forecasts

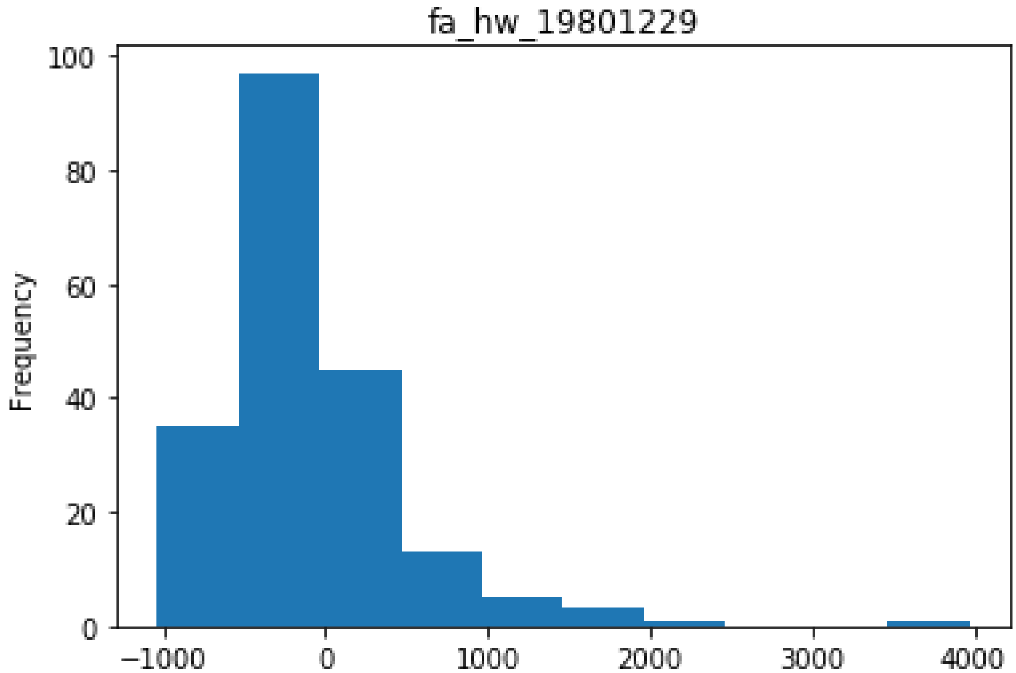
```
In [8]: fnames = pst.pestpp_options["forecasts"].split(',')
        fnames
```

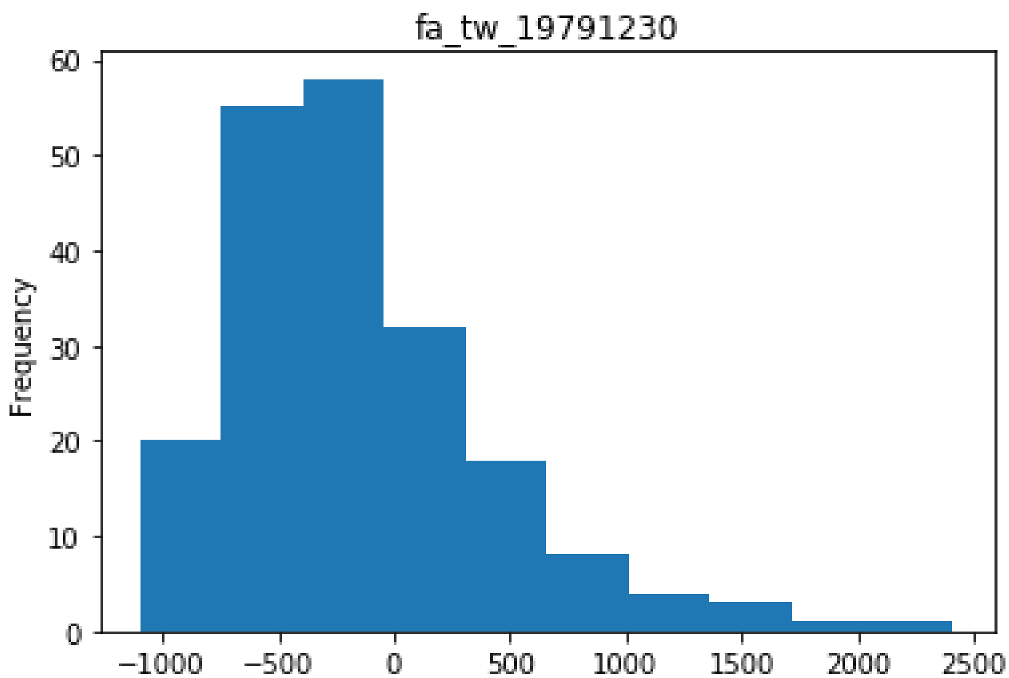
```
Out[8]: ['fa_hw_19791230',
         'fa_hw_19801229',
         'fa_tw_19791230',
         'fa_tw_19801229',
         'hds_00_013_002_000',
         'hds_00_013_002_001',
         'part_time',
         'part_status']
```

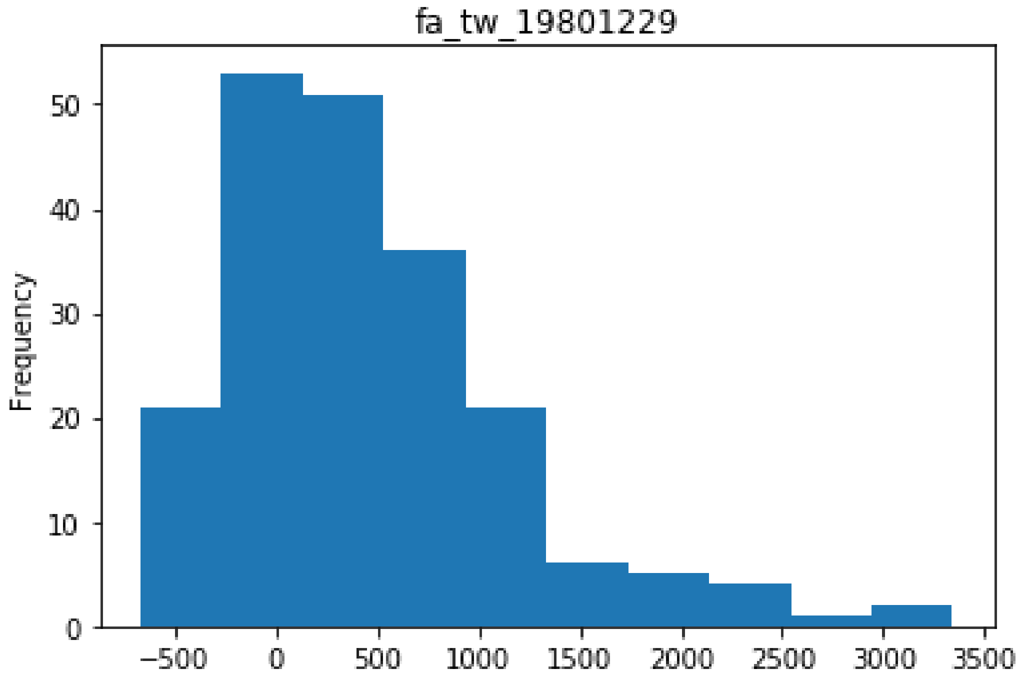
1.0.6 now we can plot the distributions of each forecast

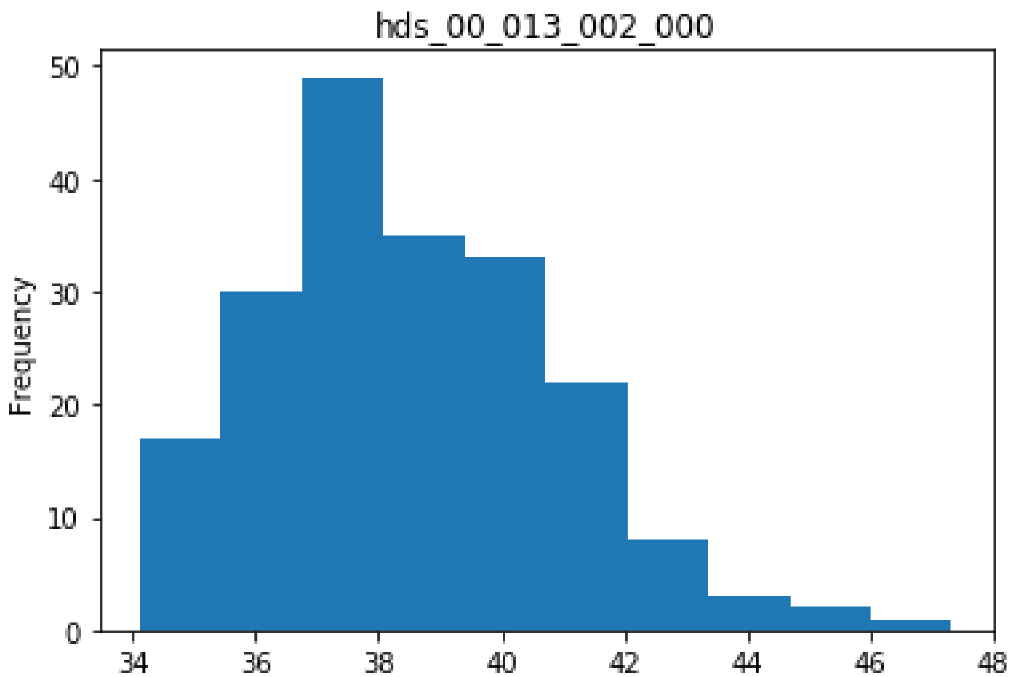
```
In [9]: for forecast in fnames:
            plt.figure()
            ax = obs_df.loc[:,forecast].plot(kind="hist")
            ax.set_title(forecast)
        plt.show()
```

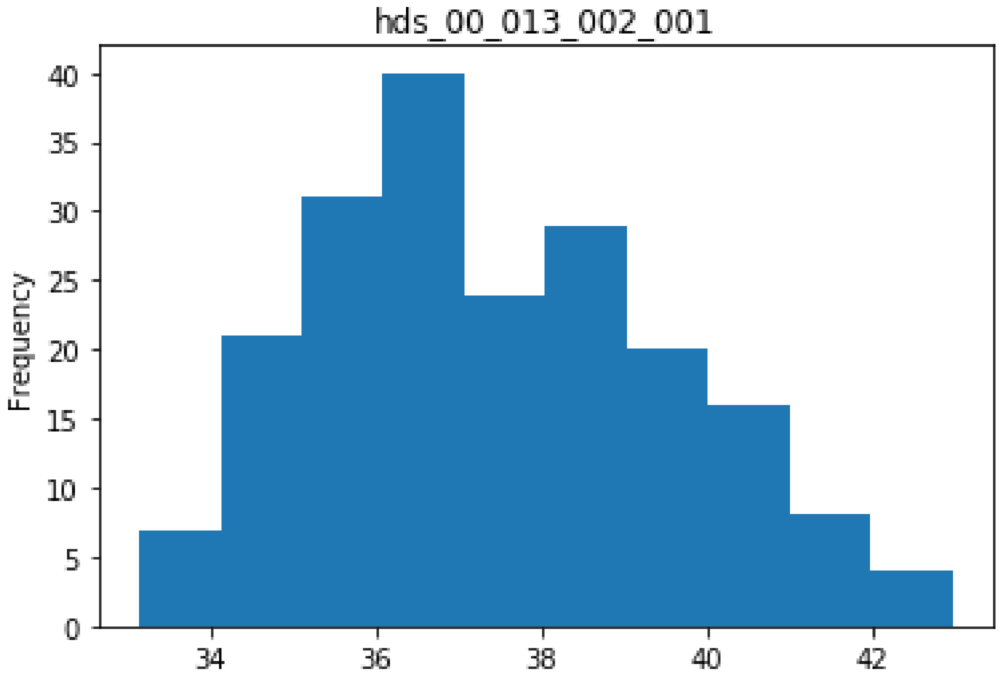


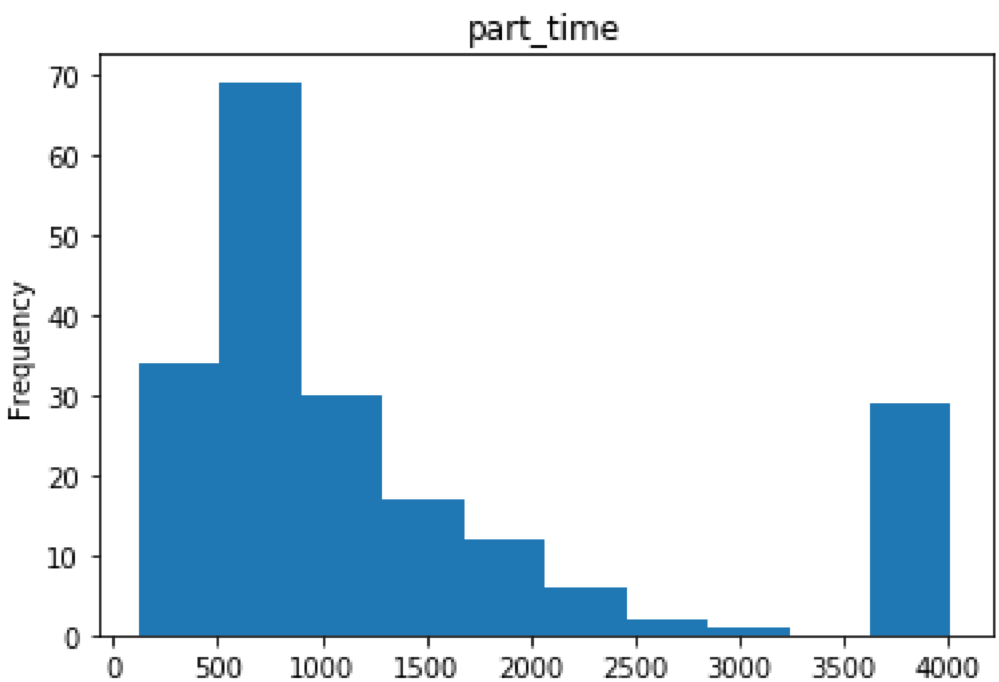

fa_hw_19801229
Frequency
100
80
60
40
20
0
−1000
0
1000
2000
3000
4000

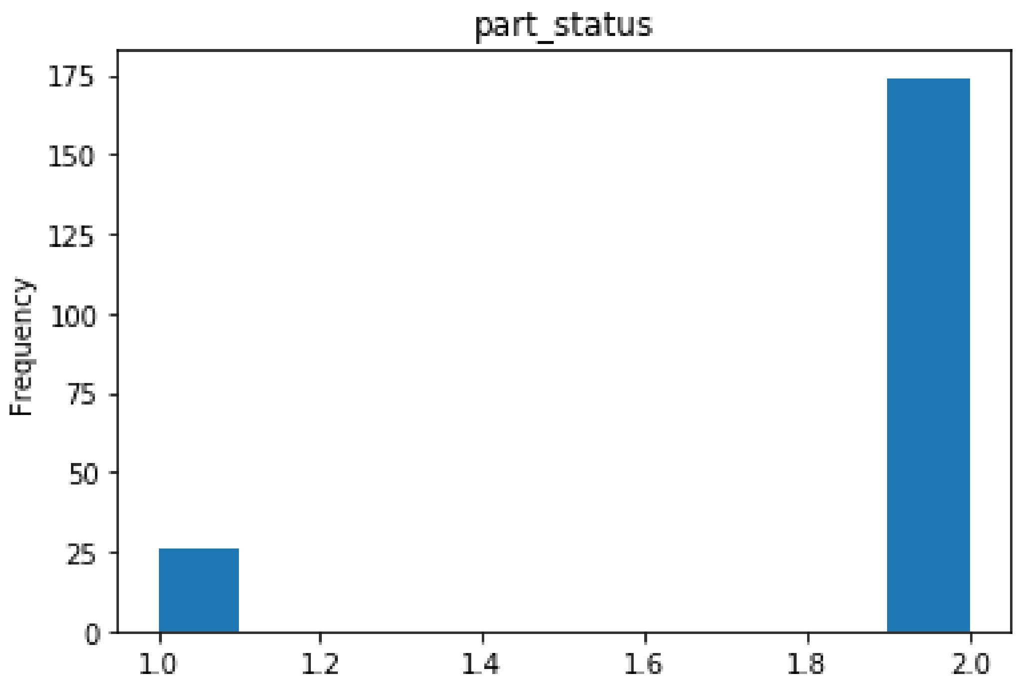

fa_tw_19791230
Frequency
60
50
40
30
20
10
0
−1000
−500
0
500
1000
1500
2000
2500


fa_tw_19801229
Frequency
0
10
20
30
40
50
−500
0
500
1000
1500
2000
2500
3000
3500


hds_00_013_002_000
Frequency
0
10
20
30
40
50
34
36
38
40
42
44
46
48

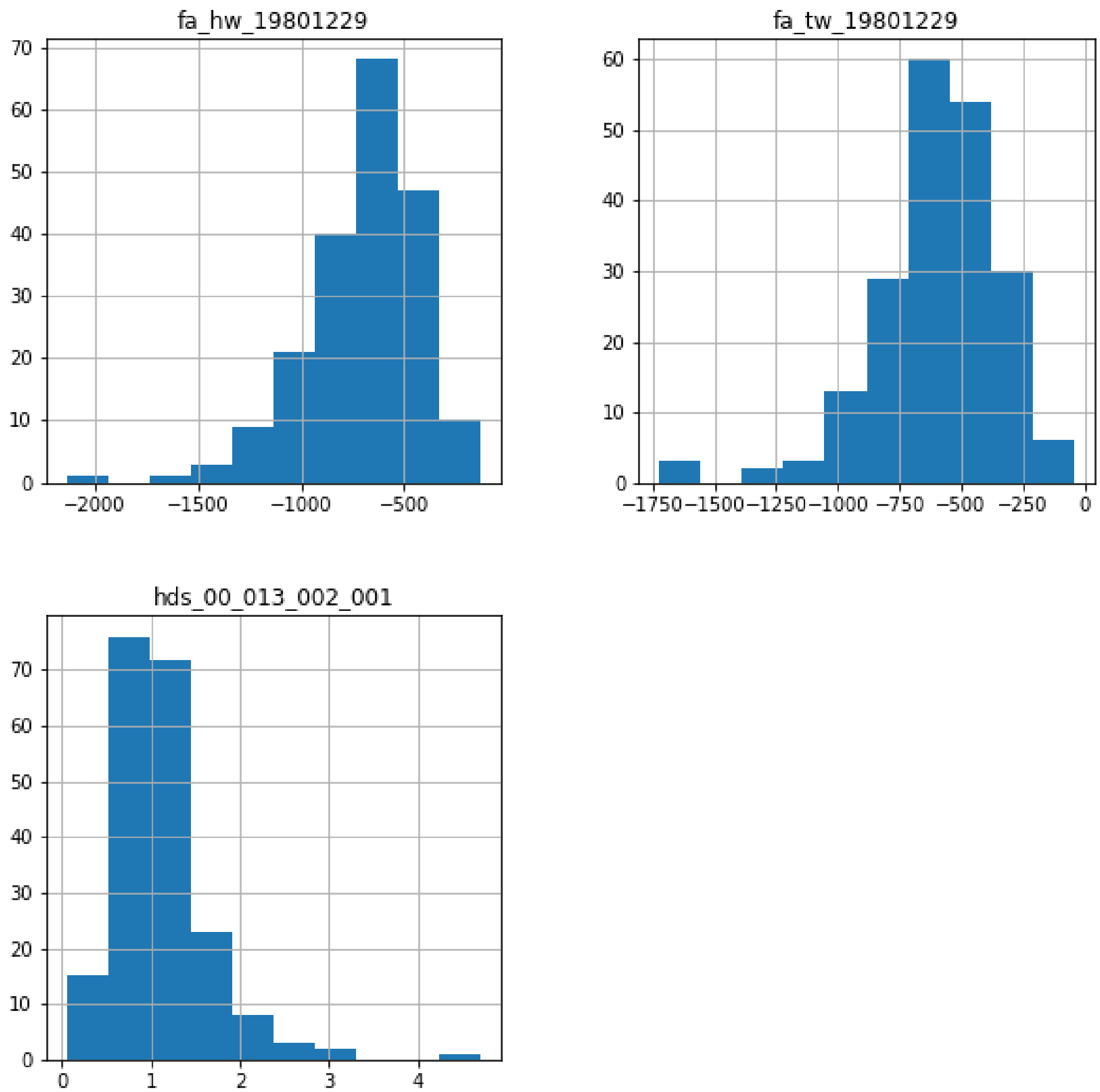

hds_00_013_002_001
Frequency
0
5
10
15
20
25
30
35
40
34
36
38
40
42


part_time
Frequency
0
10
20
30
40
50
60
70
0
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()
         idx = hist_swgw.index[100]
         idx
         hist_swgw
```

```
Out[11]: run_id
72     -1439.898100
114    -1407.885900
135    -1400.619100
57     -1371.581000
100    -1330.061800
21     -1322.180700
145    -1310.682900
40     -1299.322400
93     -1297.668900
45     -1289.105600
137    -1282.997600
131    -1273.692600
33     -1270.382600
159    -1263.388500
69     -1234.232800
49     -1229.869700
23     -1222.724680
168    -1212.237700
119    -1202.302000
36     -1195.689330
151    -1183.319000
155    -1182.481800
43     -1176.382700
136    -1165.006500
133    -1161.195400
7      -1149.238900
17     -1141.957200
25     -1141.522700
105    -1138.499000
31     -1137.319600
                ...
63      -467.193200
75      -459.782621
89      -443.632510
59      -440.333500
27      -403.127710
38      -395.609500
61      -363.008265
161     -357.548200
116     -265.722100
103     -264.996300
76      -255.625800
180     -252.541616
52      -251.051210
173     -237.581829
92      -222.417300
108     -221.172100
```

```
190    -191.298431
30     -154.094510
111    -146.408040
71      -55.583140
98      -36.865800
56       49.053710
12      131.248200
51      251.162480
0       278.989000
165     387.888840
47      513.326680
113     781.076500
129     849.195100
120    1830.209000
Name: fa_hw_19791230, Length: 200, dtype: float64
```

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

Out[12]:
```
fo_39_19791230         12061.000000
hds_00_002_009_000        36.519390
hds_00_002_015_000        34.630413
hds_00_003_008_000        36.641518
hds_00_009_001_000        37.685490
hds_00_013_010_000        34.970249
hds_00_015_016_000        34.697254
hds_00_021_010_000        34.946480
hds_00_022_015_000        34.708725
hds_00_024_004_000        35.704624
hds_00_026_006_000        35.169151
hds_00_029_015_000        34.468609
hds_00_033_007_000        34.716385
hds_00_034_010_000        34.254543
Name: 8, 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        -870.595760
fa_hw_19801229        -221.448077
fa_tw_19791230        -666.416960
fa_tw_19801229        -102.134800
hds_00_013_002_000      37.737610
hds_00_013_002_001      36.841492
part_time             1597.562000
part_status              2.000000
Name: 8, 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"] = 0.005
```

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        352.810469
         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
```

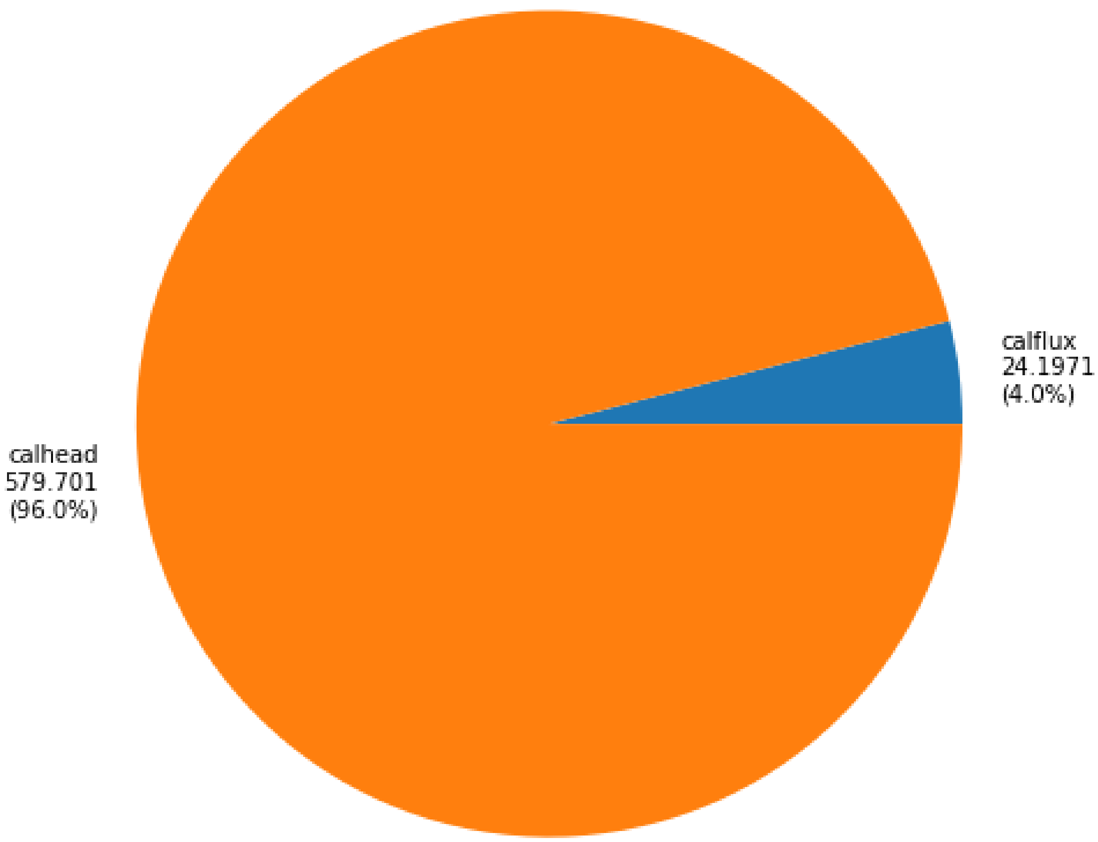
Now we can read in the results and make some figures showing residuals and the balance of the objective function

```
In [17]: pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
         print(pst.phi)
         plt.figure()
         pst.plot(kind='phi_pie')
```

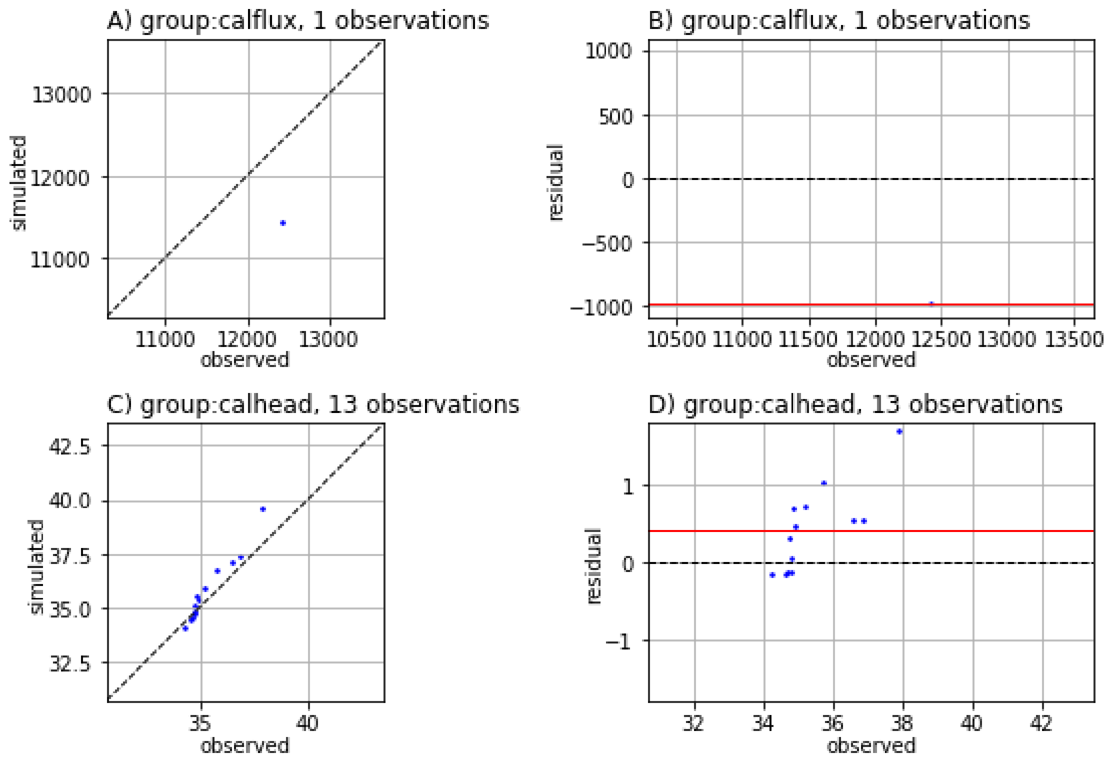
```
print('Here are the non-zero weighted observation names')

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

```
603.8982892954039
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  12413.810469  11430.000000
```

```
hds_00_002_009_000  hds_00_002_009_000  calhead     36.559406     37.107498
hds_00_002_015_000  hds_00_002_015_000  calhead     34.728287     35.045185
hds_00_003_008_000  hds_00_003_008_000  calhead     36.865607     37.397289
hds_00_009_001_000  hds_00_009_001_000  calhead     37.872245     39.546417
hds_00_013_010_000  hds_00_013_010_000  calhead     34.872521     35.571774
hds_00_015_016_000  hds_00_015_016_000  calhead     34.792263     34.835716
hds_00_021_010_000  hds_00_021_010_000  calhead     34.931344     35.386250
hds_00_022_015_000  hds_00_022_015_000  calhead     34.698403     34.577492
hds_00_024_004_000  hds_00_024_004_000  calhead     35.745684     36.760464
hds_00_026_006_000  hds_00_026_006_000  calhead     35.183556     35.896149
hds_00_029_015_000  hds_00_029_015_000  calhead     34.614036     34.453842
hds_00_033_007_000  hds_00_033_007_000  calhead     34.792489     34.678810
hds_00_034_010_000  hds_00_034_010_000  calhead     34.266711     34.118073

                      residual  weight
name
fo_39_19791230      983.810469   0.005
hds_00_002_009_000   -0.548092  10.000
hds_00_002_015_000   -0.316898  10.000
hds_00_003_008_000   -0.531682  10.000
hds_00_009_001_000   -1.674172  10.000
hds_00_013_010_000   -0.699252  10.000
hds_00_015_016_000   -0.043453  10.000
hds_00_021_010_000   -0.454905  10.000
hds_00_022_015_000    0.120911  10.000
hds_00_024_004_000   -1.014780  10.000
hds_00_026_006_000   -0.712593  10.000
hds_00_029_015_000    0.160194  10.000
hds_00_033_007_000    0.113679  10.000
hds_00_034_010_000    0.148638  10.000
```

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"]}
#pst.adjust_weights(obsgrp_dict=target)
#pst.plot(kind='phi_pie')
```

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 [19]:
```
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 [20]:
```python
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,:]
```

```
17.52884723870795
```

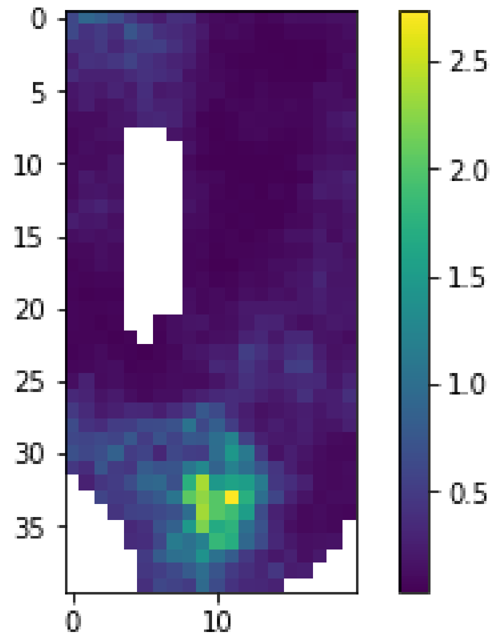
Out[20]:
```
                                 name    group     measured     modelled  \
name
fo_39_19791230         fo_39_19791230  calflux  12413.810469  12061.000000
hds_00_002_009_000  hds_00_002_009_000  calhead     36.559406     36.519390
hds_00_002_015_000  hds_00_002_015_000  calhead     34.728287     34.630413
hds_00_003_008_000  hds_00_003_008_000  calhead     36.865607     36.641518
hds_00_009_001_000  hds_00_009_001_000  calhead     37.872245     37.685490
hds_00_013_010_000  hds_00_013_010_000  calhead     34.872521     34.970249
hds_00_015_016_000  hds_00_015_016_000  calhead     34.792263     34.697254
hds_00_021_010_000  hds_00_021_010_000  calhead     34.931344     34.946480
hds_00_022_015_000  hds_00_022_015_000  calhead     34.698403     34.708725
hds_00_024_004_000  hds_00_024_004_000  calhead     35.745684     35.704624
hds_00_026_006_000  hds_00_026_006_000  calhead     35.183556     35.169151
hds_00_029_015_000  hds_00_029_015_000  calhead     34.614036     34.468609
hds_00_033_007_000  hds_00_033_007_000  calhead     34.792489     34.716385
hds_00_034_010_000  hds_00_034_010_000  calhead     34.266711     34.254543

                      residual  weight
name
fo_39_19791230      352.810469   0.005
hds_00_002_009_000    0.040016  10.000
hds_00_002_015_000    0.097874  10.000
hds_00_003_008_000    0.224089  10.000
hds_00_009_001_000    0.186756  10.000
hds_00_013_010_000   -0.097728  10.000
hds_00_015_016_000    0.095009  10.000
hds_00_021_010_000   -0.015136  10.000
hds_00_022_015_000   -0.010322  10.000
hds_00_024_004_000    0.041060  10.000
hds_00_026_006_000    0.014404  10.000
hds_00_029_015_000    0.145427  10.000
hds_00_033_007_000    0.076104  10.000
hds_00_034_010_000    0.012168  10.000
```

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

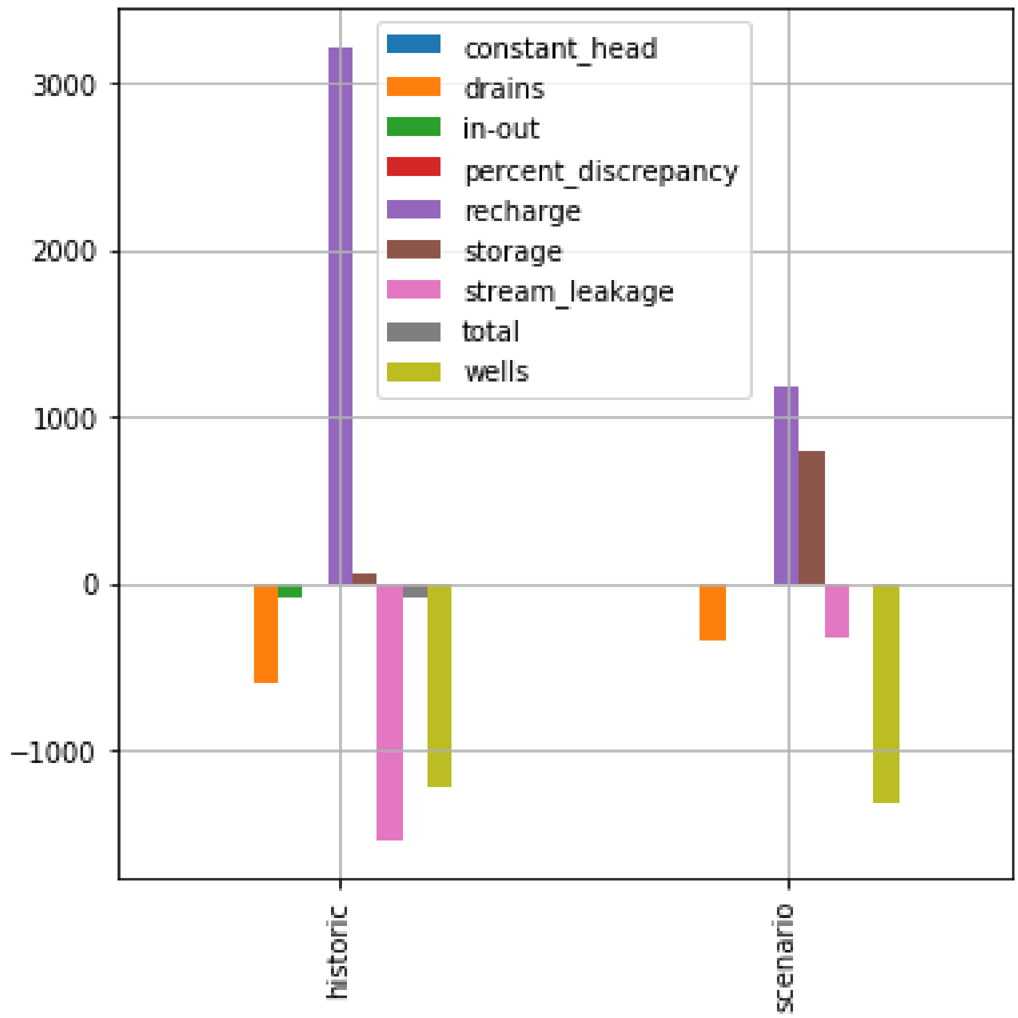
```
In [21]: m = flopy.modflow.Modflow.load("freyberg.nam",model_ws=m_d)
```

```
In [22]: a = m.upw.vka[1].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()
```

```
0.03417815 2.730791
```



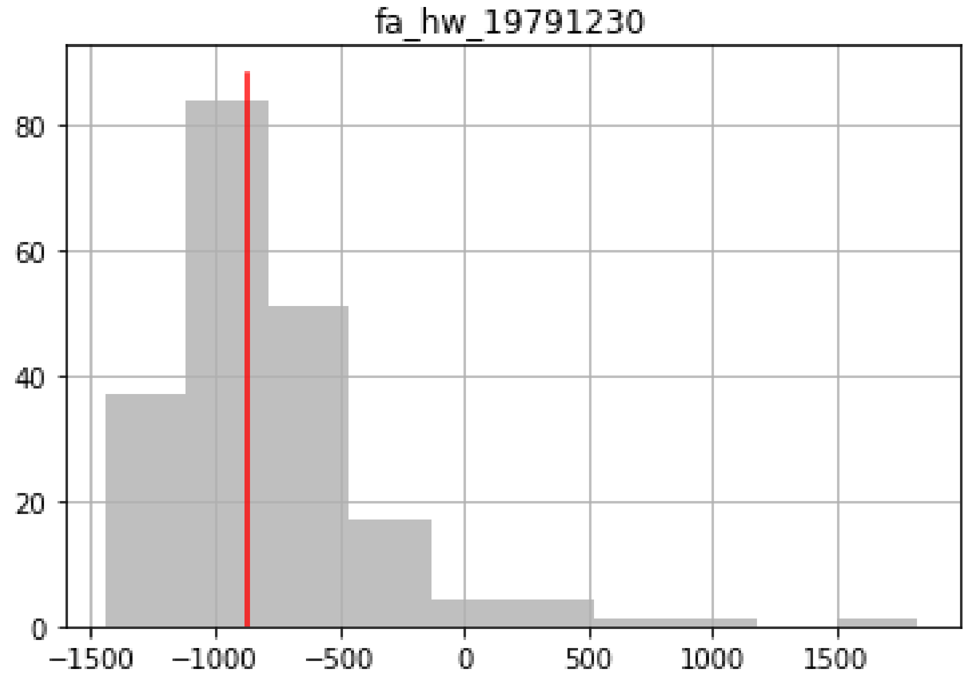

```
In [23]: lst = flopy.utils.MfListBudget(os.path.join(m_d,"freyberg.list"))
         df = lst.get_dataframes(diff=True)[0]
         ax = df.plot(kind="bar",figsize=(6,6), grid=True)
         a = ax.set_xticklabels(["historic","scenario"],rotation=90)
         plt.show(ax)
```

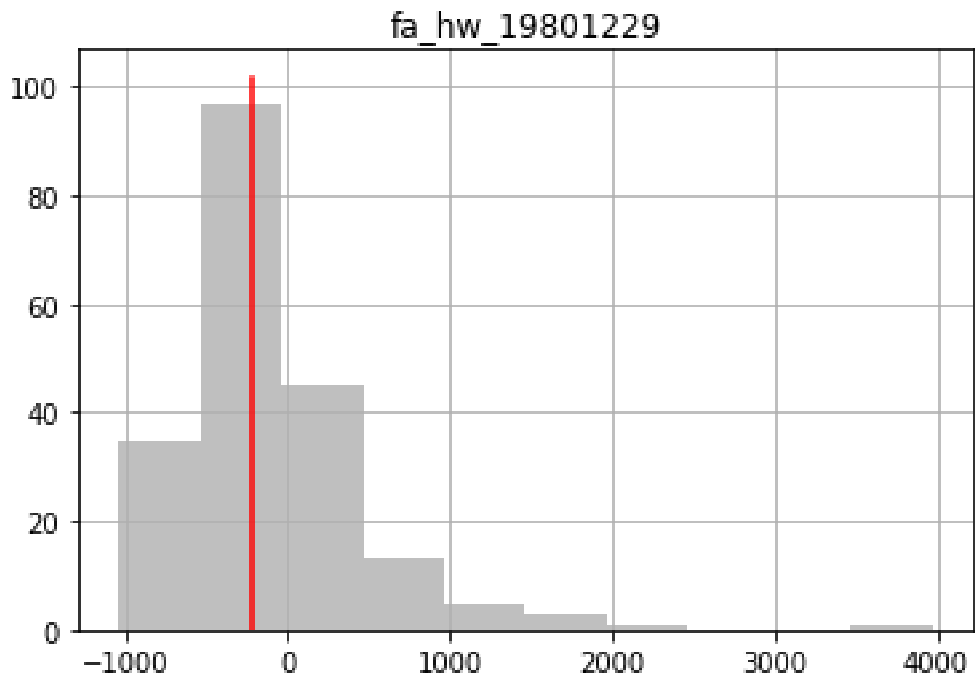


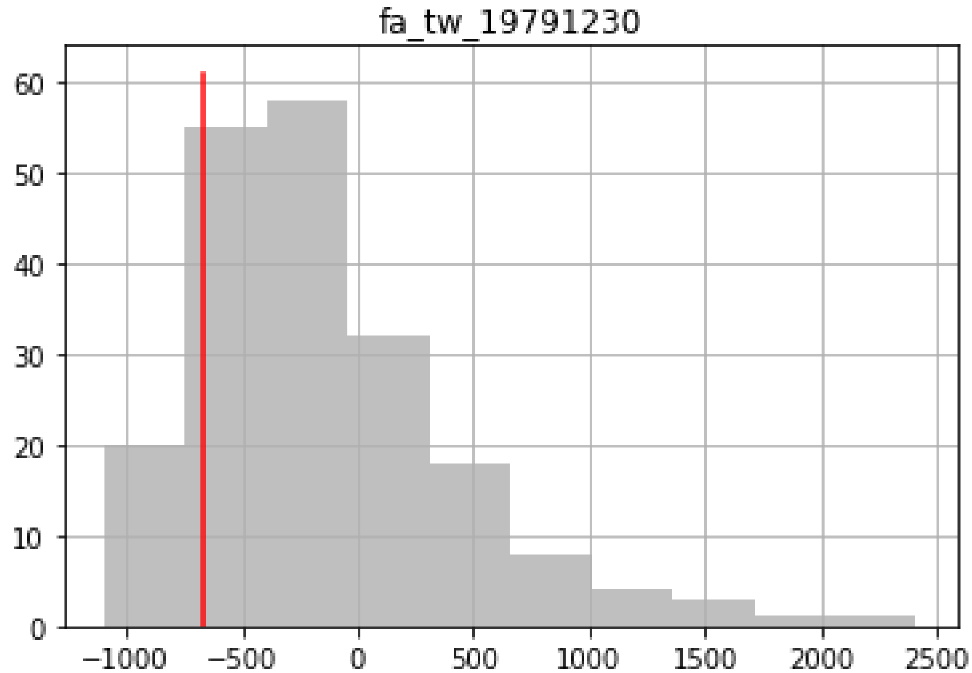
1.0.8 see how our existing observation ensemble compares to the truth

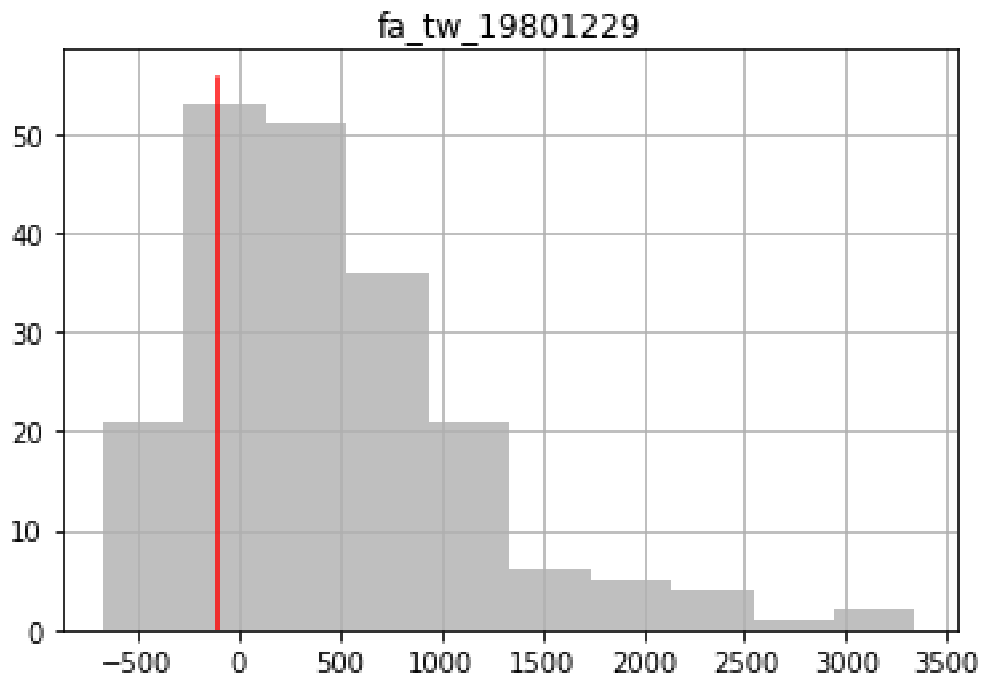
forecasts:

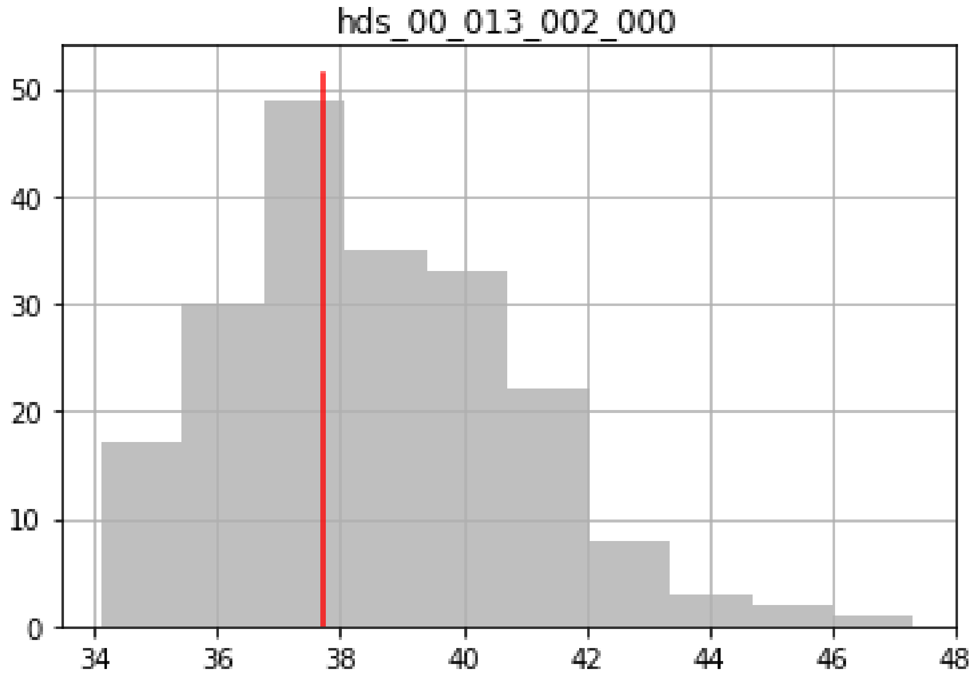
```
In [24]: obs = pst.observation_data
         plt.figure()
         for forecast in fnames:
             ax = plt.subplot(111)
             obs_df.loc[:,forecast].hist(ax=ax,color="0.5",alpha=0.5)
             ax.plot([obs.loc[forecast,"obsval"],obs.loc[forecast,"obsval"]],ax.get_ylim(),"r"
             ax.set_title(forecast)
             plt.show()
```

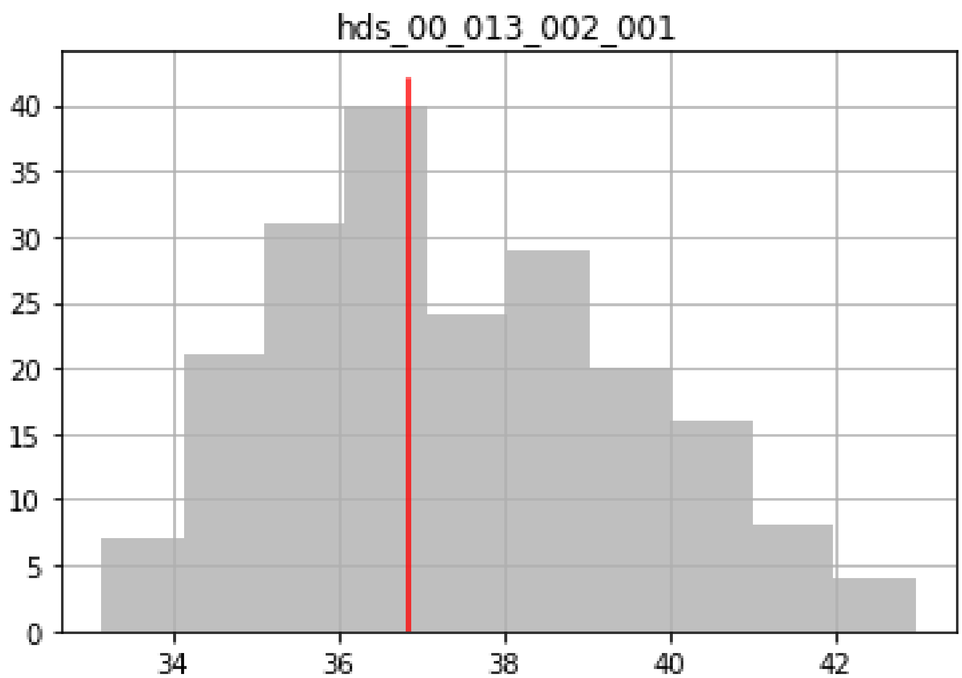

fa_hw_19791230
80
60
40
20
0
−1500
−1000
−500
0
500
1000
1500

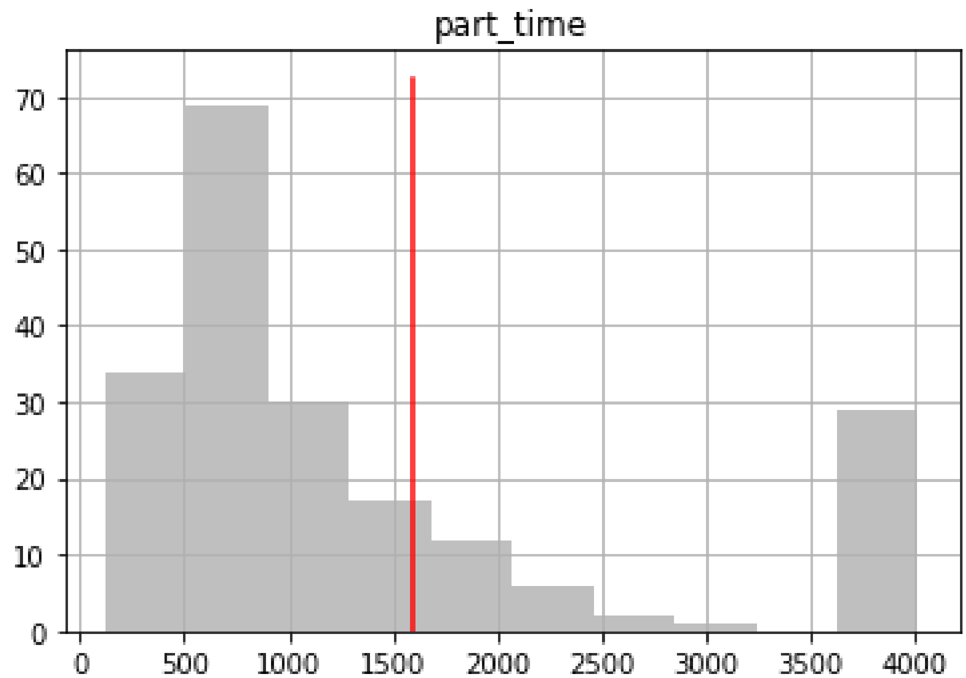

fa_hw_19801229
100
80
60
40
20
0
−1000
0
1000
2000
3000
4000

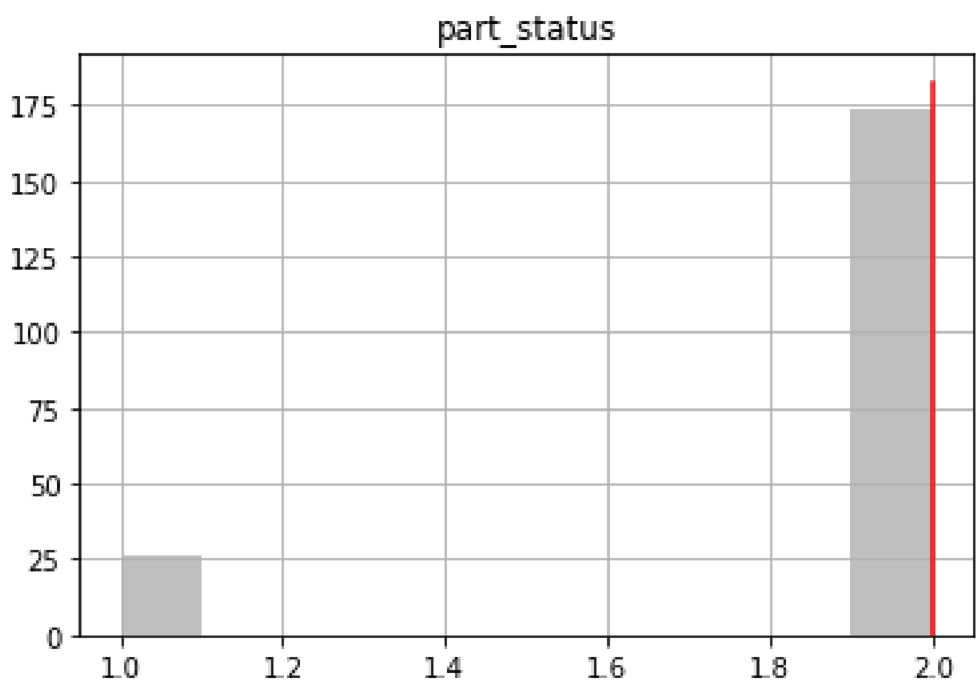

fa_tw_19791230
60
50
40
30
20
10
0
−1000
−500
0
500
1000
1500
2000
2500


fa_tw_19801229
50
40
30
20
10
0
−500
0
500
1000
1500
2000
2500
3000
3500


hds_00_013_002_000
0
10
20
30
40
50
34
36
38
40
42
44
46
48

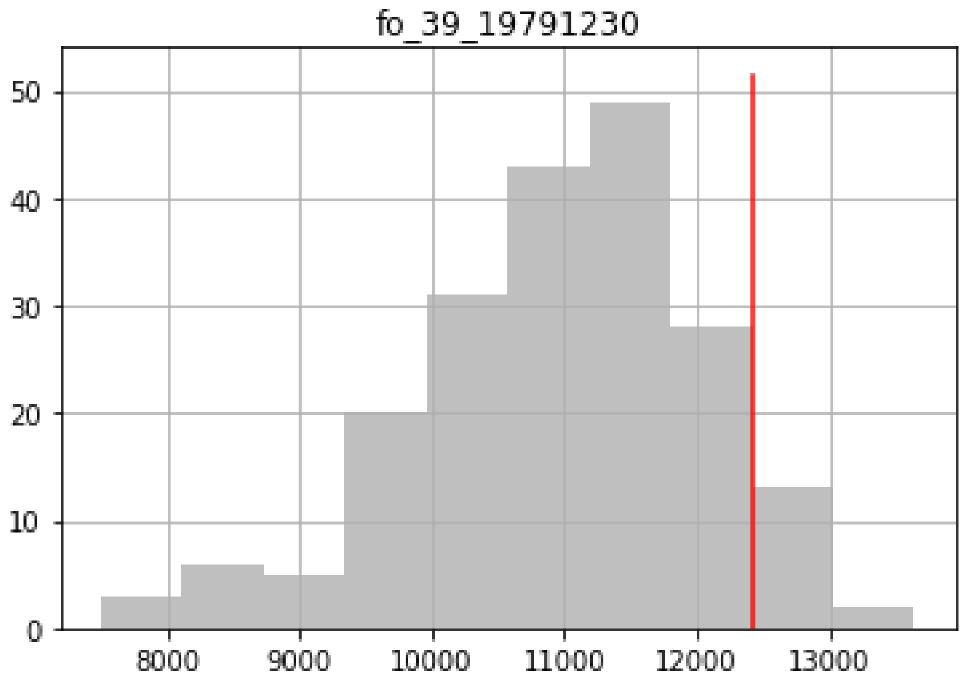

hds_00_013_002_001
0
5
10
15
20
25
30
35
40
34
36
38
40
42

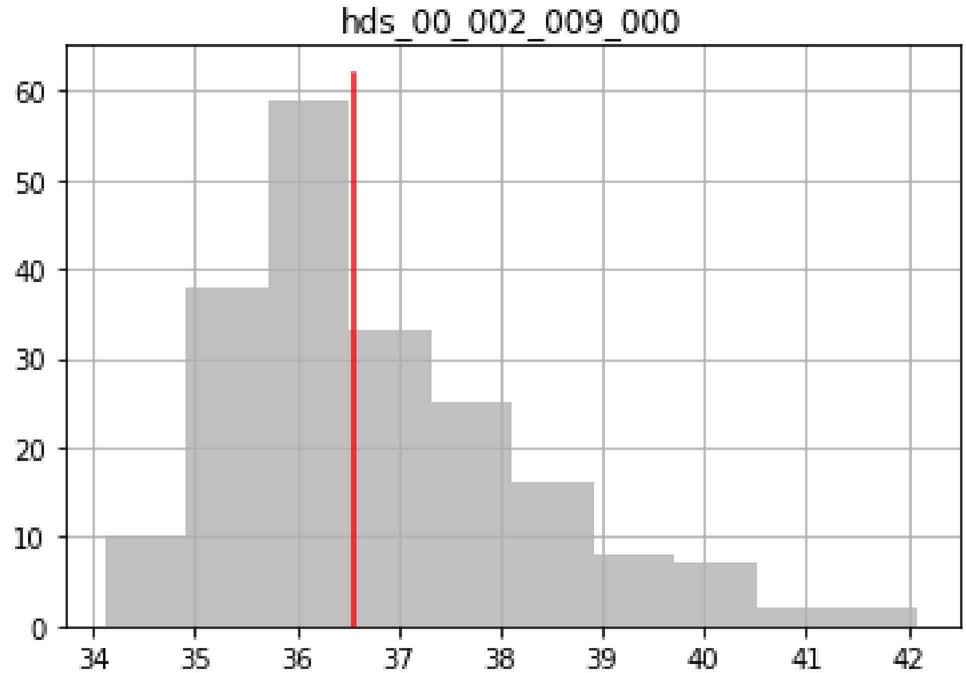


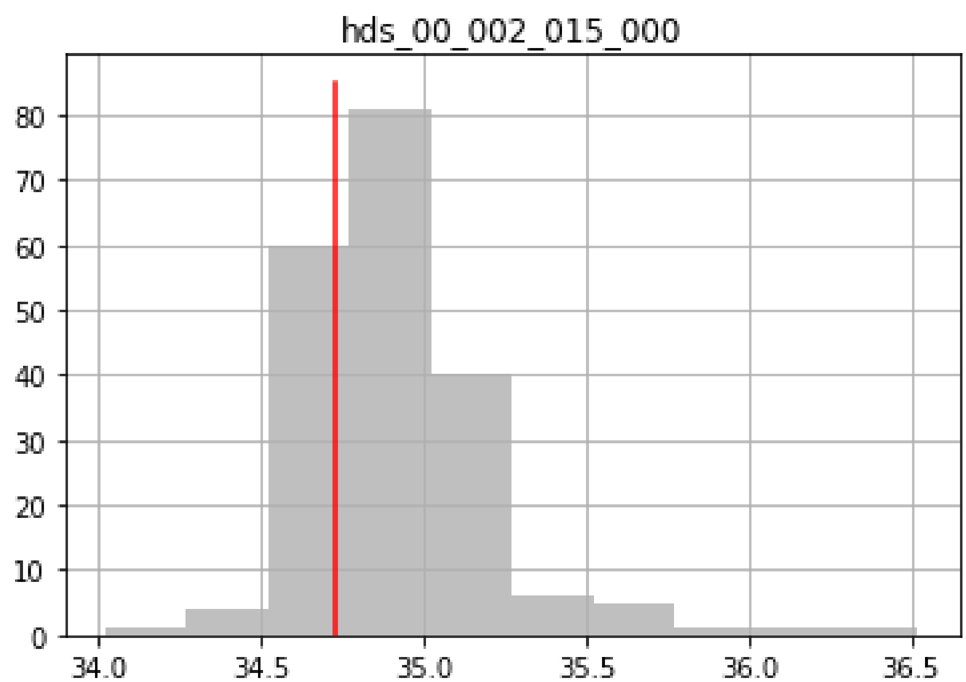


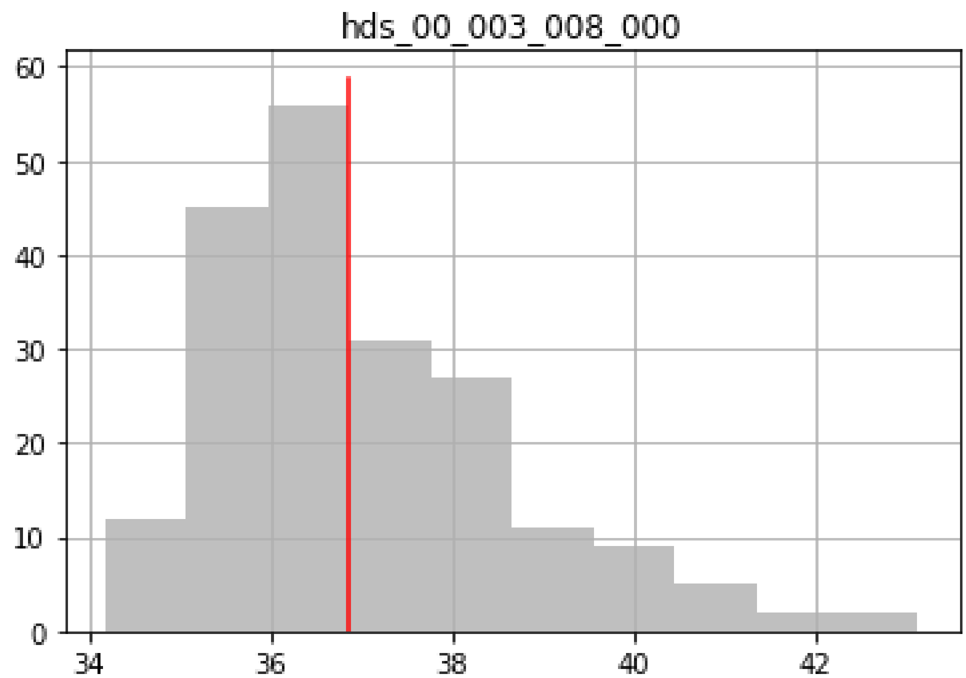
observations:

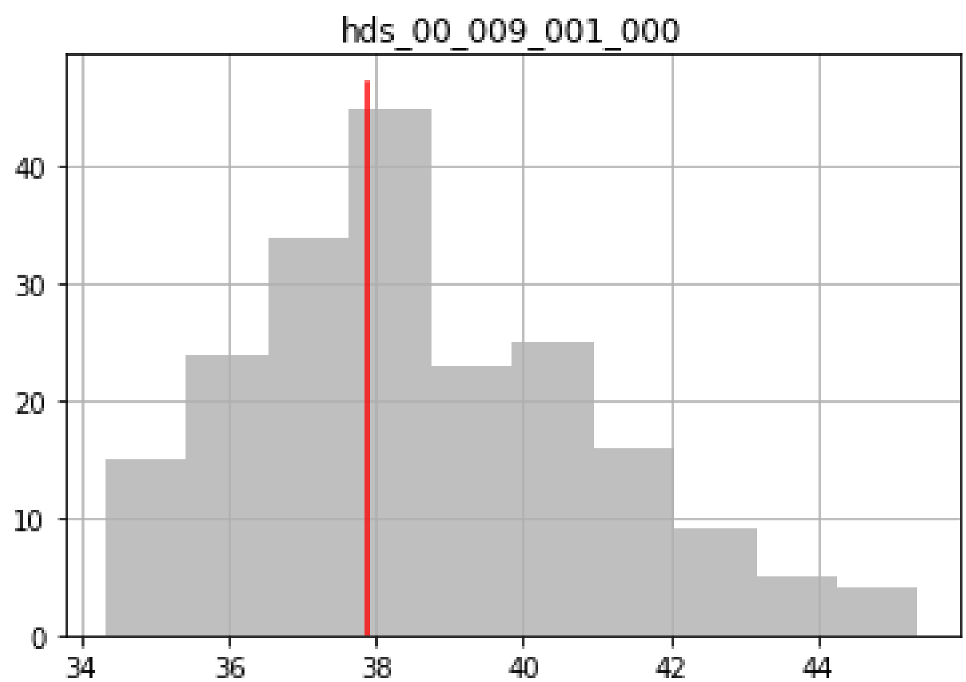
In [25]:
```
for oname in pst.nnz_obs_names:
    ax = plt.subplot(111)
    obs_df.loc[:,oname].hist(ax=ax,color="0.5",alpha=0.5)
    ax.plot([obs.loc[oname,"obsval"],obs.loc[oname,"obsval"]],ax.get_ylim(),"r")
    ax.set_title(oname)
    plt.show()
```

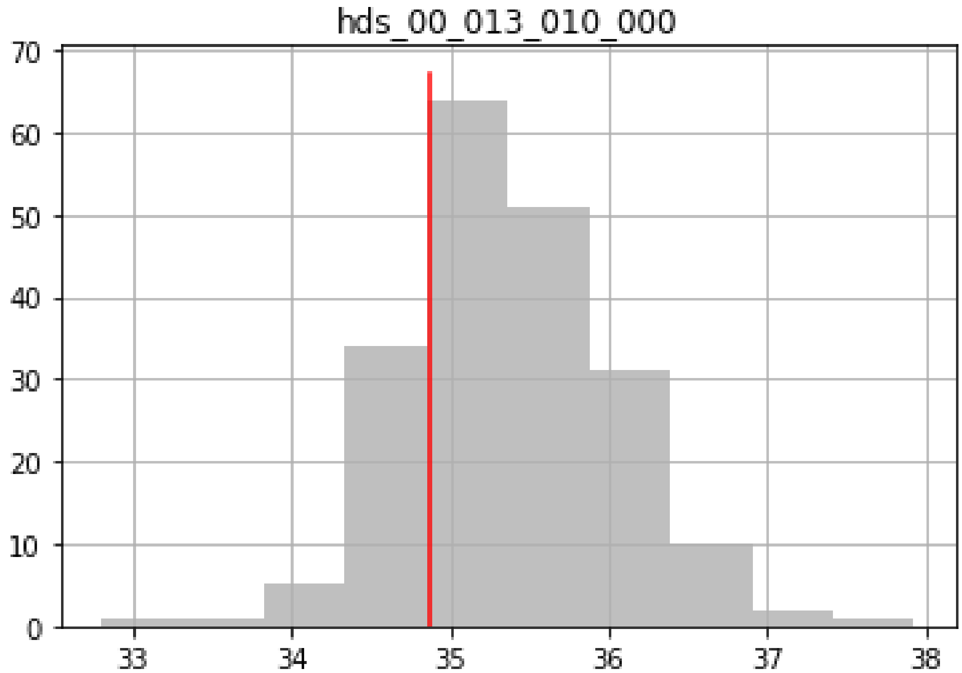


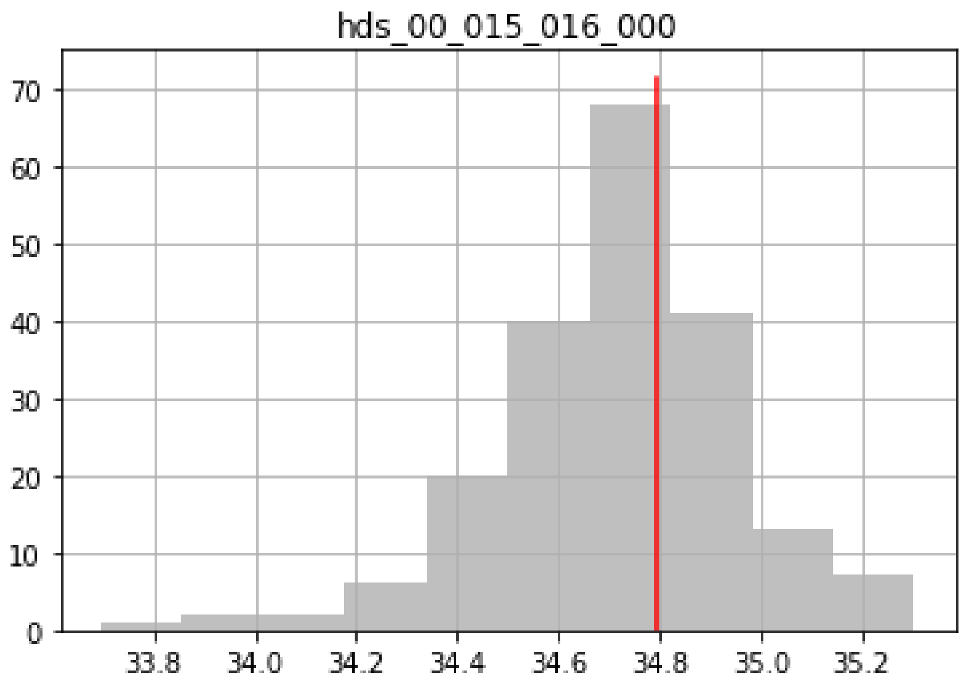

hds_00_002_009_000
60
50
40
30
20
10
0
34
35
36
37
38
39
40
41
42

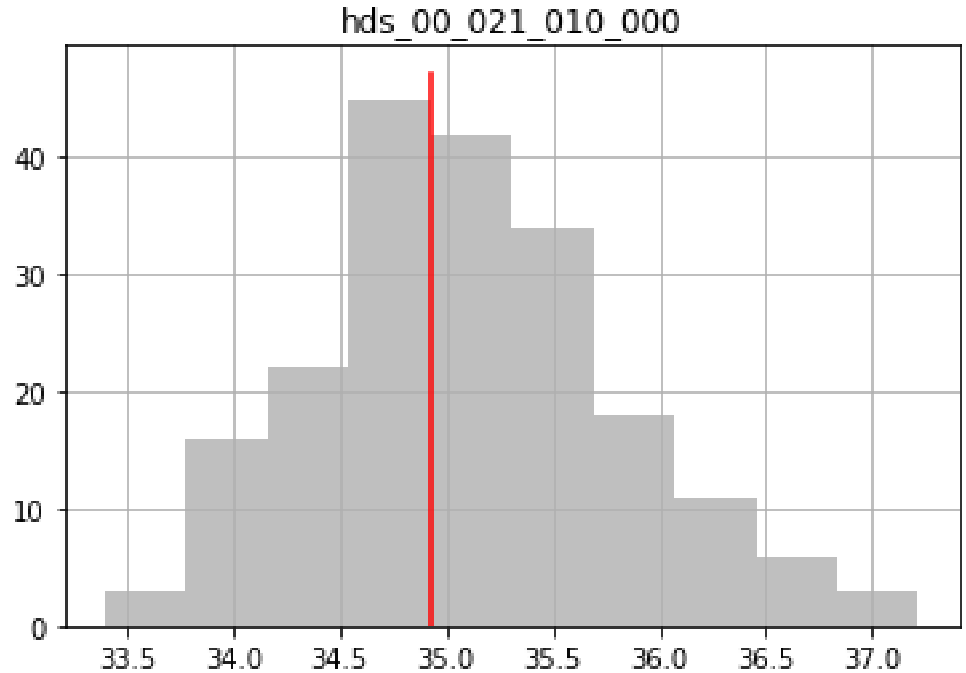

hds_00_002_015_000
80
70
60
50
40
30
20
10
0
34.0
34.5
35.0
35.5
36.0
36.5

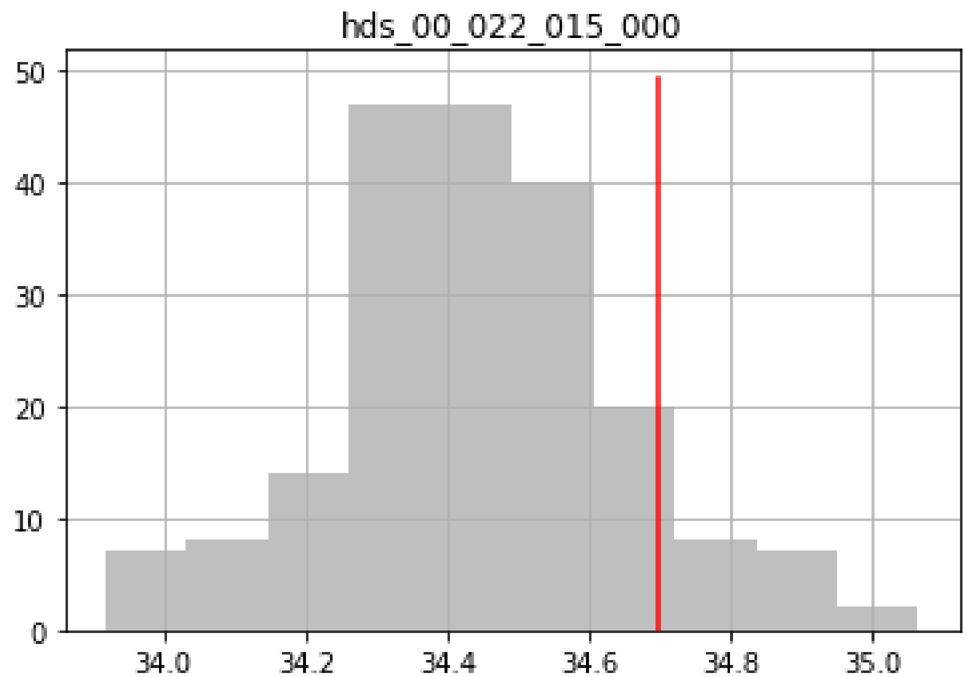

hds_00_003_008_000
0
10
20
30
40
50
60
34
36
38
40
42

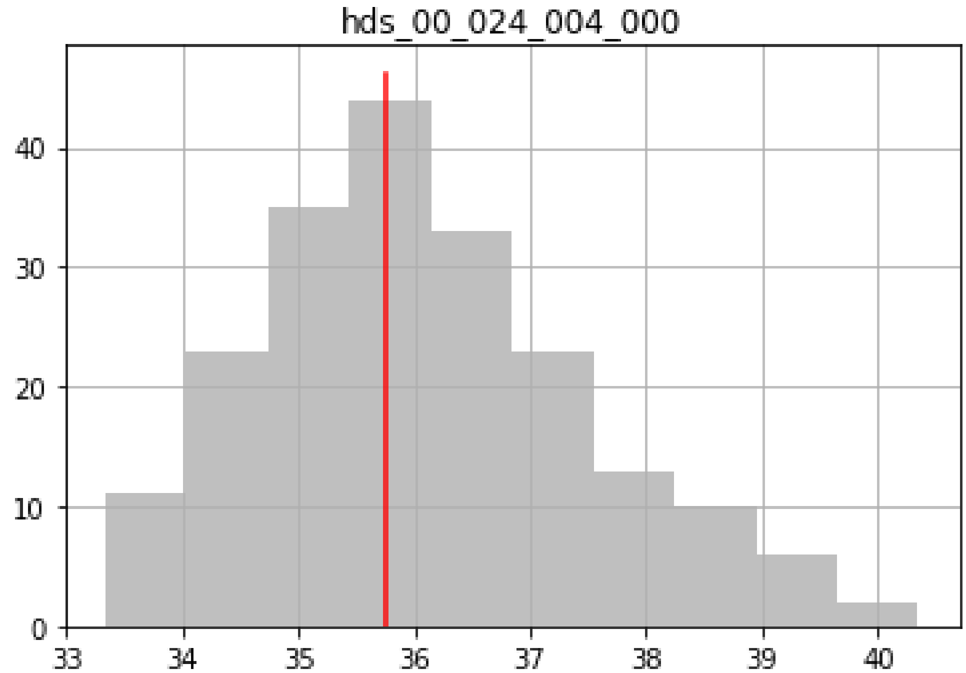

hds_00_009_001_000
0
10
20
30
40
34
36
38
40
42
44

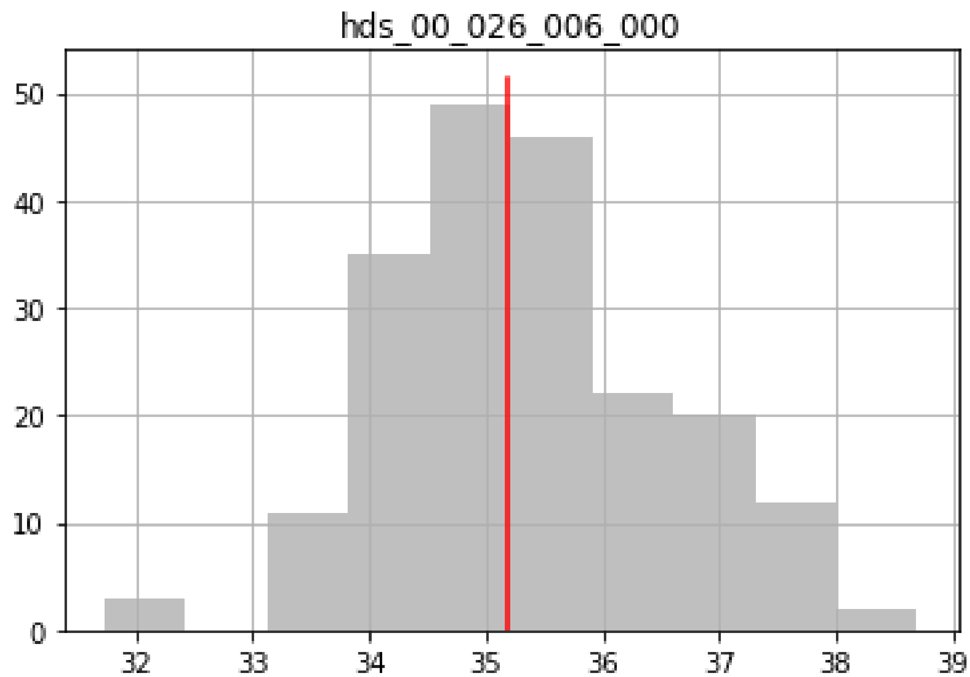

hds_00_013_010_000
0
10
20
30
40
50
60
70
33
34
35
36
37
38

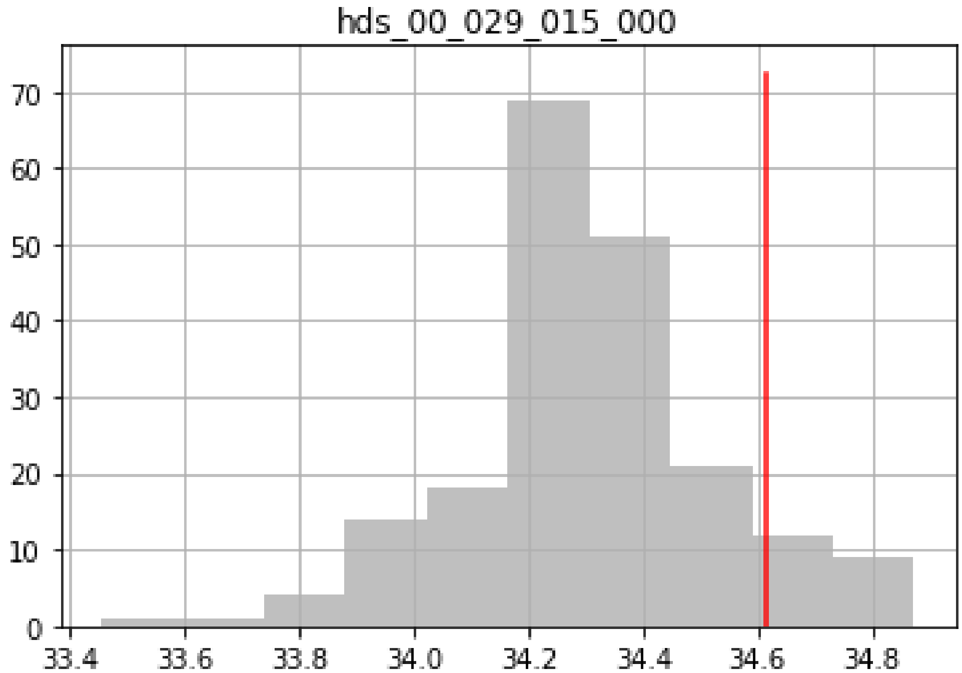

hds_00_015_016_000
0
10
20
30
40
50
60
70
33.8
34.0
34.2
34.4
34.6
34.8
35.0
35.2

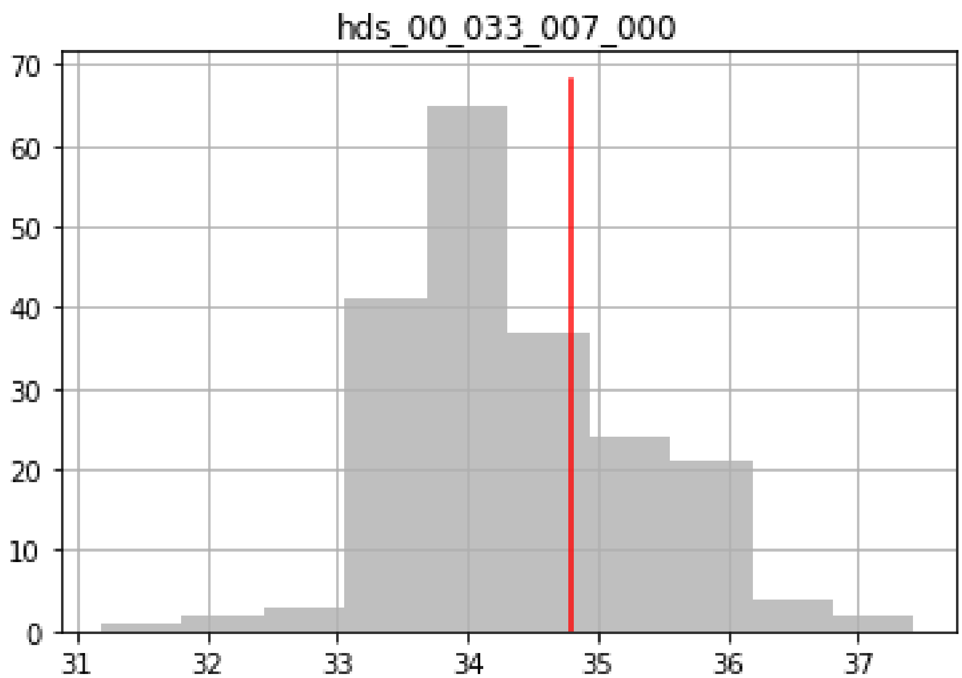

hds_00_021_010_000
0
10
20
30
40
33.5
34.0
34.5
35.0
35.5
36.0
36.5
37.0

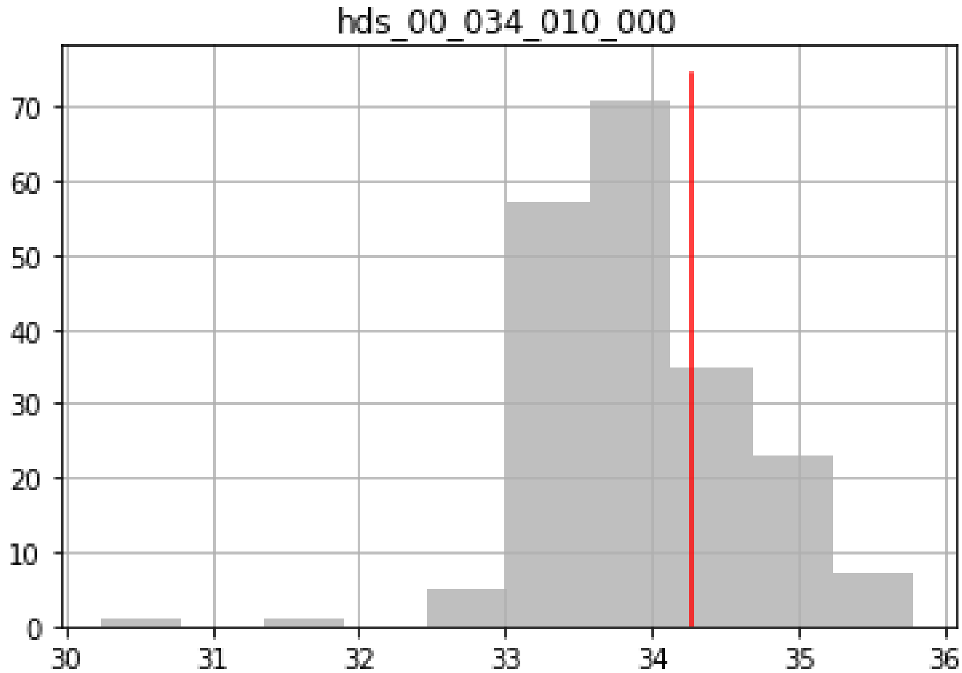

hds_00_022_015_000
0
10
20
30
40
50
34.0
34.2
34.4
34.6
34.8
35.0


hds_00_024_004_000
0
10
20
30
40
33
34
35
36
37
38
39
40


hds_00_026_006_000
0
10
20
30
40
50
32
33
34
35
36
37
38
39


hds_00_029_015_000
70
60
50
40
30
20
10
0
33.4
33.6
33.8
34.0
34.2
34.4
34.6
34.8


hds_00_033_007_000
70
60
50
40
30
20
10
0
31
32
33
34
35
36
37


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