prior_montecarlo

May 12, 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: 500
```

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: 500
```

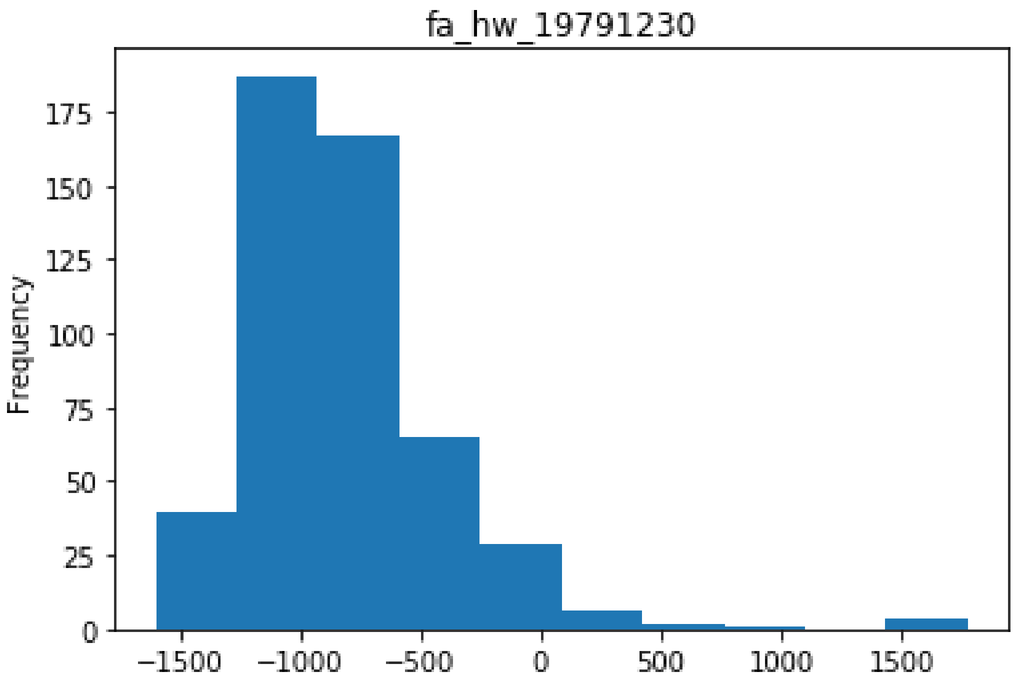
1.0.5 confirm which quantities were identified as forecasts

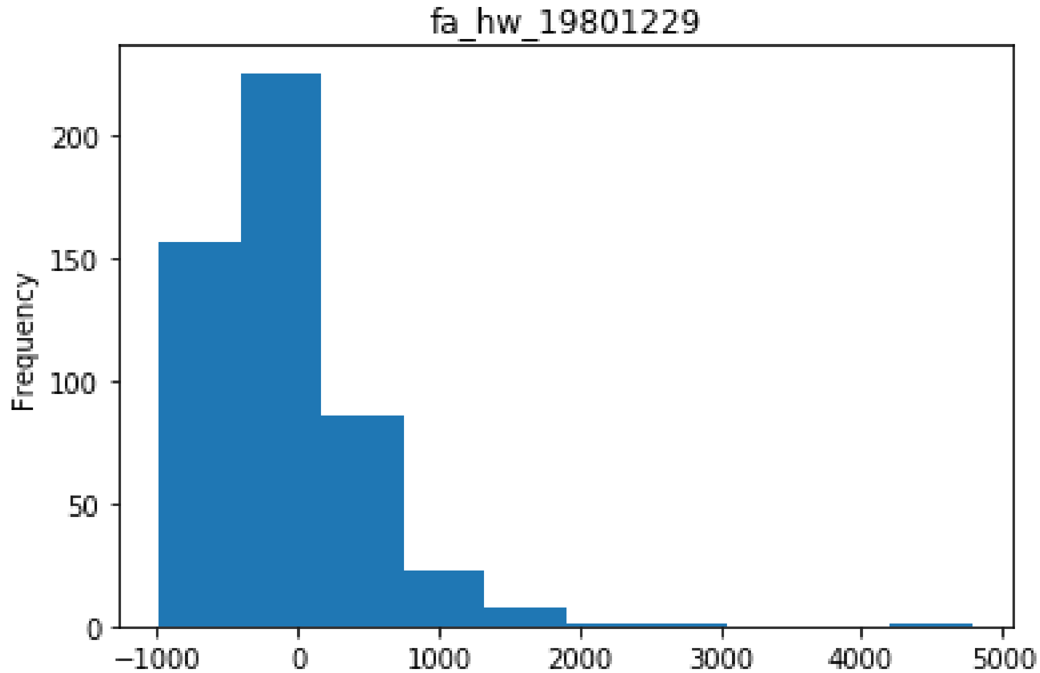
```
In [8]: fnames = pst.pestpp_options["forecasts"].split(',')
        fnames
```

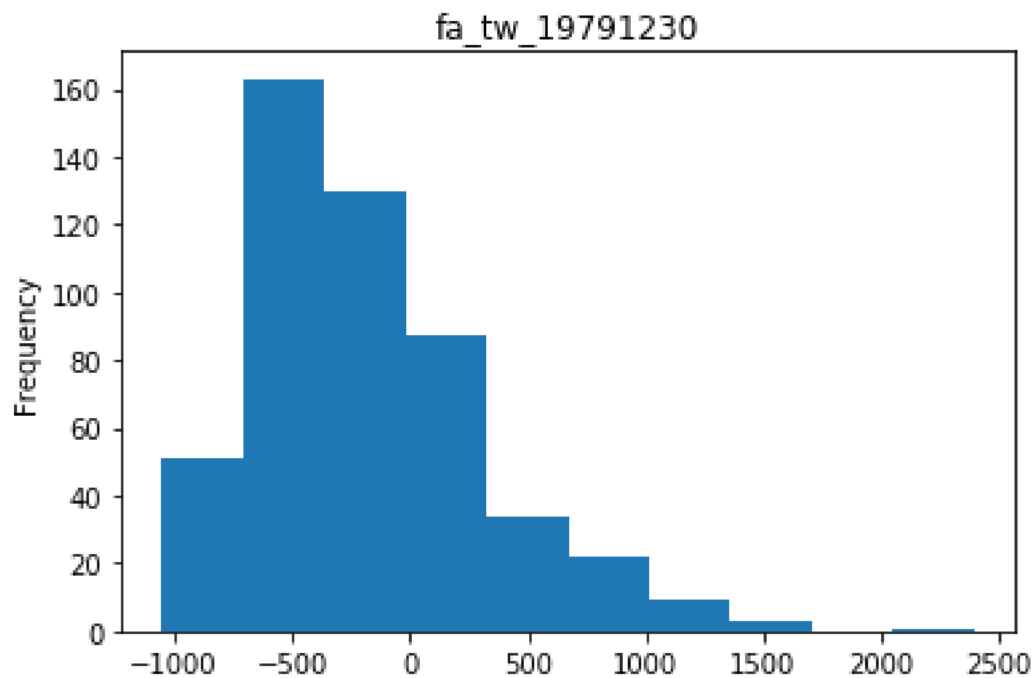
```
Out[8]: ['fa_hw_19791230',
         'fa_hw_19801229',
         'fa_tw_19791230',
         'fa_tw_19801229',
         'hds_00_013_002_000',
         'hds_00_013_002_001',
         'part_time',
         'part_status']
```

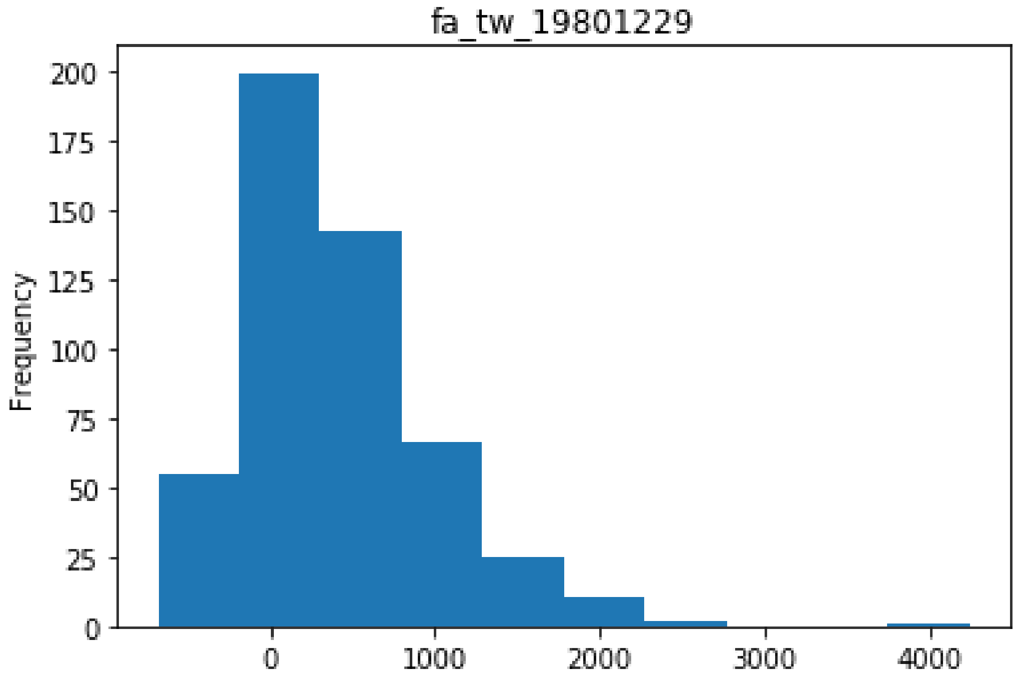
1.0.6 now we can plot the distributions of each forecast

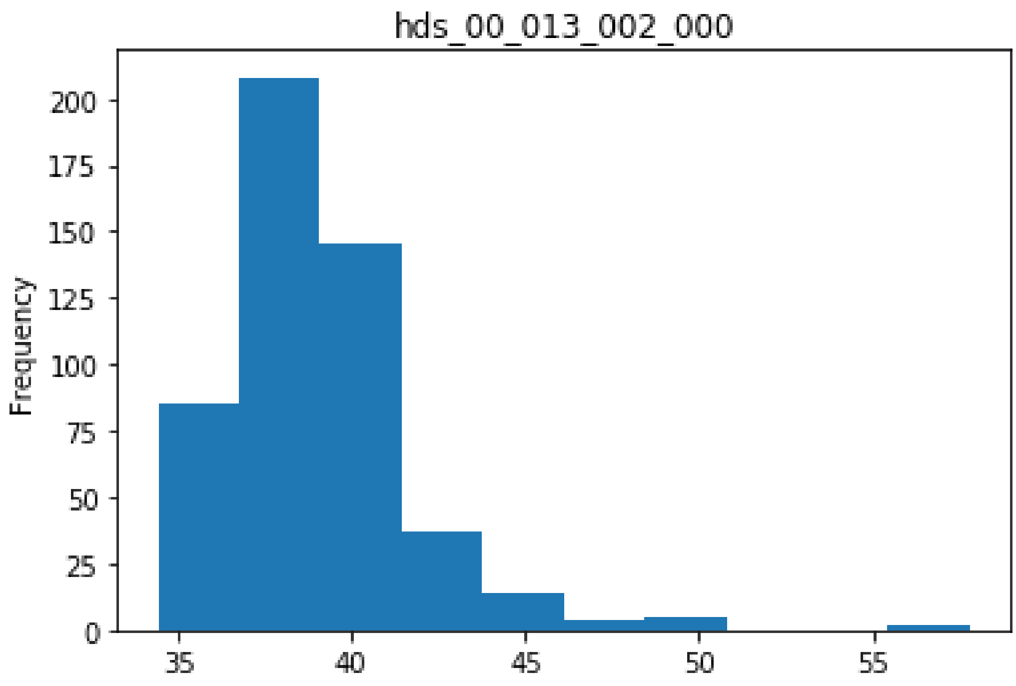
```
In [9]: for forecast in fnames:
            plt.figure()
            ax = obs_df.loc[:,forecast].plot(kind="hist")
            ax.set_title(forecast)
        plt.show()
```

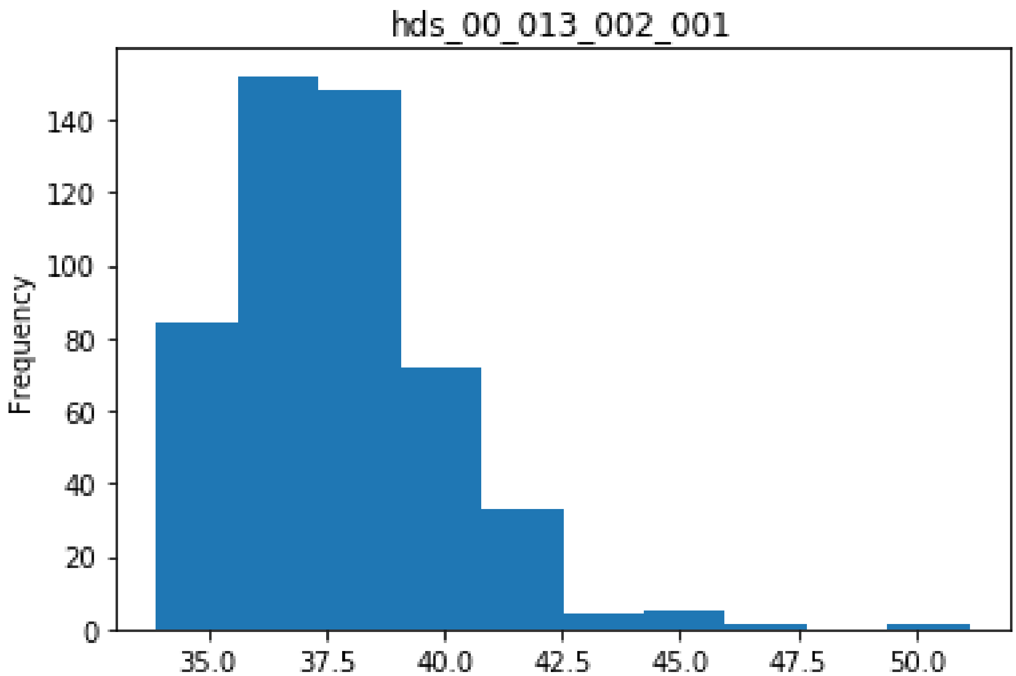


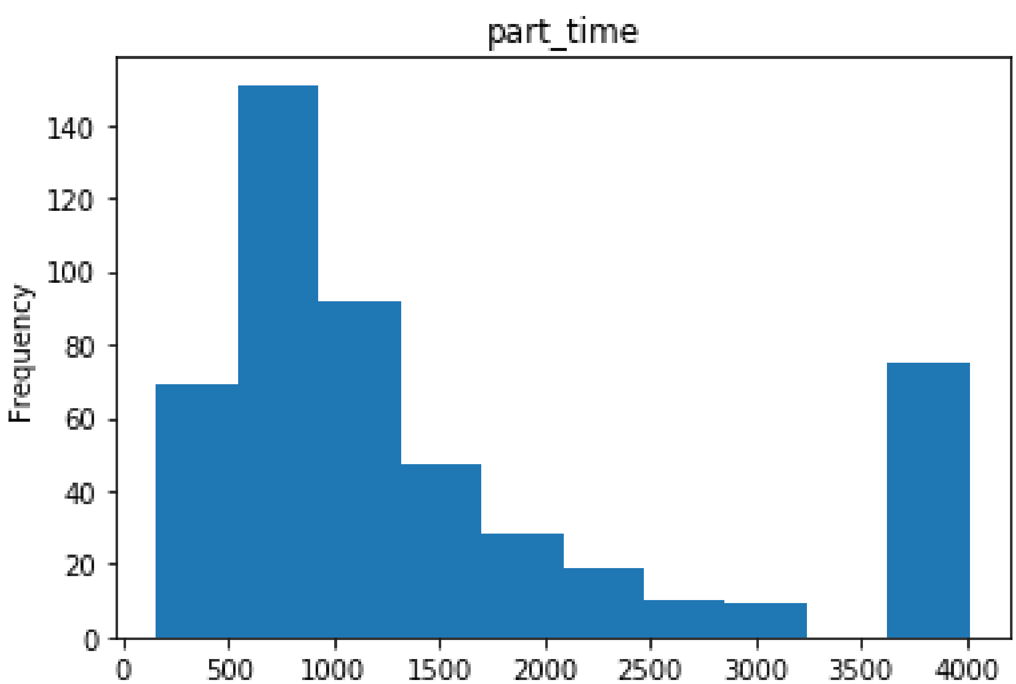

fa_hw_19801229
Frequency
200
150
100
50
0
−1000
0
1000
2000
3000
4000
5000

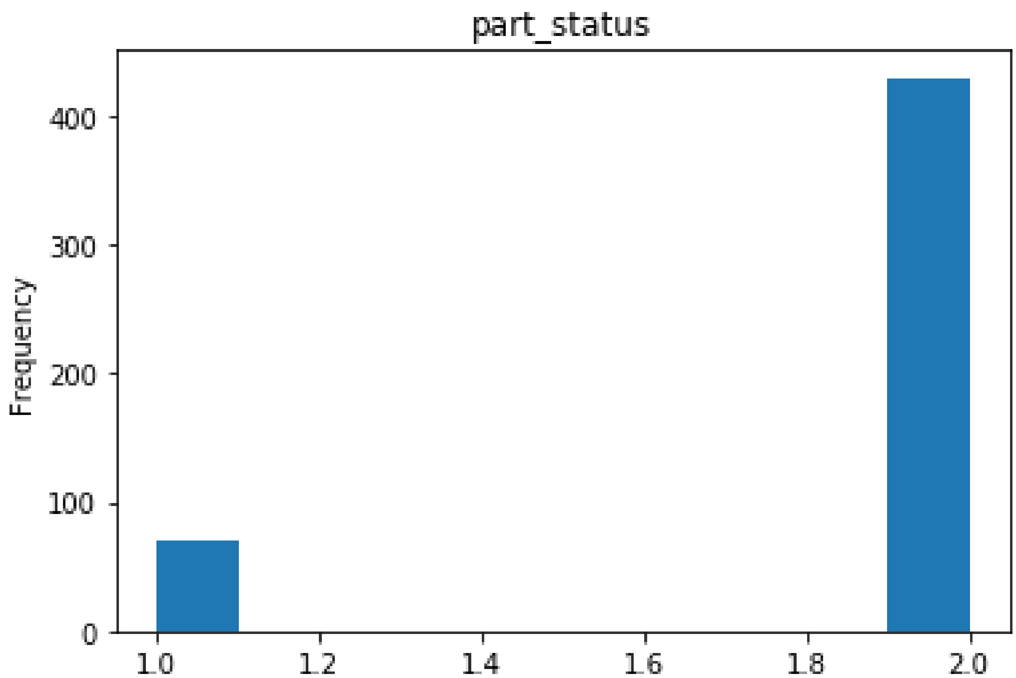

fa_tw_19791230
Frequency
160
140
120
100
80
60
40
20
0
−1000
−500
0
500
1000
1500
2000
2500


fa_tw_19801229
Frequency
200
175
150
125
100
75
50
25
0
0
1000
2000
3000
4000


hds_00_013_002_000
Frequency
200
175
150
125
100
75
50
25
0
35
40
45
50
55

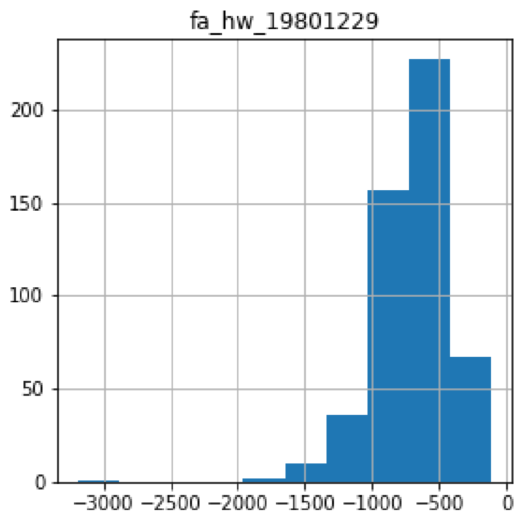

hds_00_013_002_001
Frequency
0
20
40
60
80
100
120
140
35.0
37.5
40.0
42.5
45.0
47.5
50.0

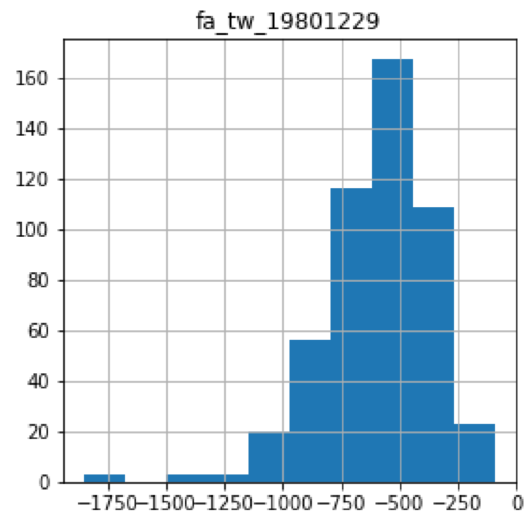

part_time
Frequency
0
20
40
60
80
100
120
140
0
500
1000
1500
2000
2500
3000
3500
4000

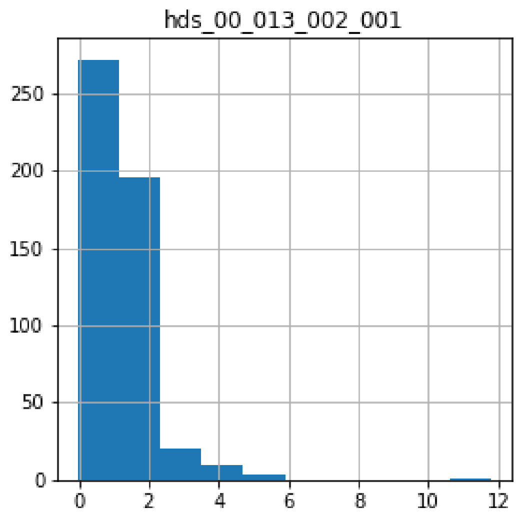


We see that under scenario conditions, many more realizations for the flow to the aquifer in the headwaters are postive (as expected). Lets difference these two:

```
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()
         hist_swgw = obs_df.loc[:,"part_time"].sort_values()
         idx = hist_swgw.index[20]
```

```
        idx
        hist_swgw

Out[11]: run_id
318      158.7090
262      217.6370
375      259.8265
77       267.3188
18       274.6335
244      281.7578
92       290.7109
155      294.9453
19       297.6992
101      303.7151
466      311.5639
197      316.1725
420      319.5435
394      327.7793
161      336.5960
97       349.8065
201      351.5172
65       356.3551
418      357.8107
258      358.4687
332      360.7590
136      368.0700
33       368.8190
36       372.1028
241      379.1664
38       381.2636
355      382.4316
465      384.3039
400      385.1978
60       387.0248
            ...
189     4015.0000
195     4015.0000
196     4015.0000
384     4015.0000
412     4015.0000
163     4015.0000
362     4015.0000
361     4015.0000
252     4015.0000
449     4015.0000
251     4015.0000
90      4015.0000
91      4015.0000
```

```
247    4015.0000
112    4015.0000
242    4015.0000
434    4015.0000
453    4015.0000
352    4015.0000
354    4015.0000
240    4015.0000
428    4015.0000
356    4015.0000
357    4015.0000
239    4015.0000
151    4015.0000
419    4015.0000
154    4015.0000
132    4015.0000
306    4015.0000
Name: part_time, Length: 500, dtype: float64
```

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

```
Out[12]: fo_39_19791230        9369.600000
hds_00_002_009_000      35.040295
hds_00_002_015_000      34.594372
hds_00_003_008_000      35.058357
hds_00_009_001_000      35.516644
hds_00_013_010_000      34.704311
hds_00_015_016_000      34.378880
hds_00_021_010_000      34.129391
hds_00_022_015_000      34.134243
hds_00_024_004_000      34.152634
hds_00_026_006_000      33.933083
hds_00_029_015_000      33.993011
hds_00_033_007_000      33.456963
hds_00_034_010_000      33.334564
Name: 332, 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        -620.622650
fa_hw_19801229         109.754350
fa_tw_19791230        1239.639600
fa_tw_19801229        1936.065600
hds_00_013_002_000      35.594288
hds_00_013_002_001      34.781490
part_time              360.759000
part_status              2.000000
Name: 332, 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)
```

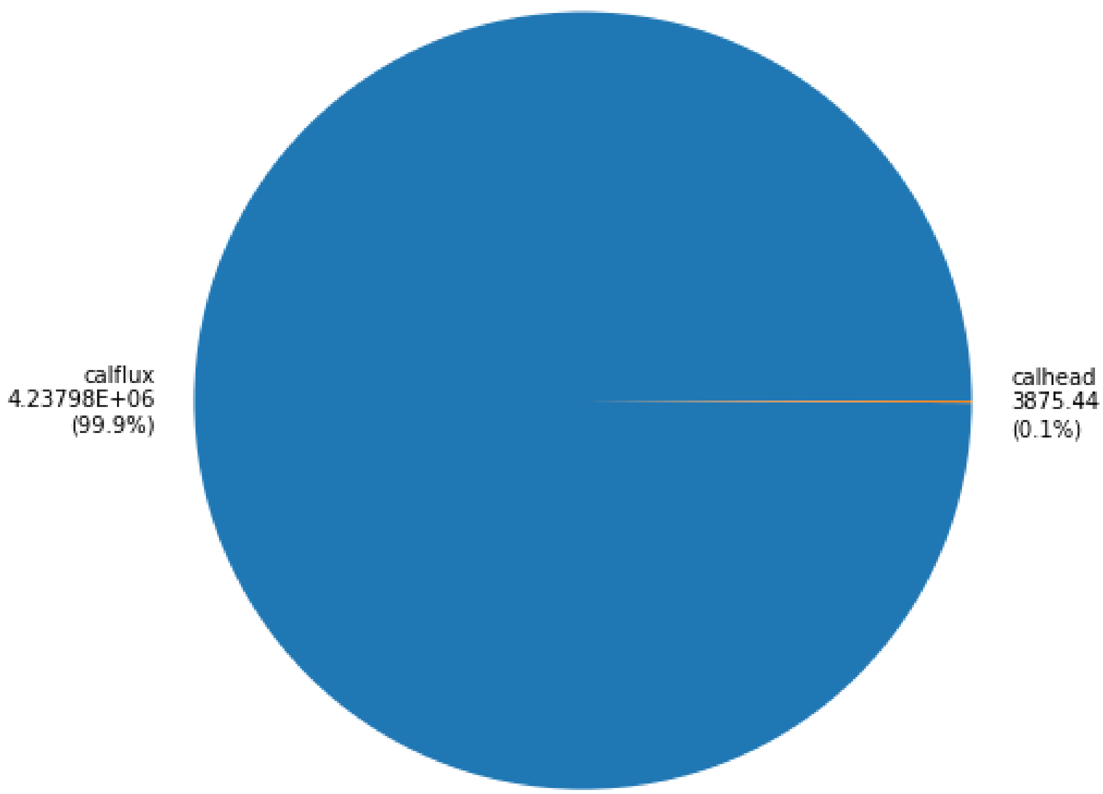
```
noptmax:0, npar_adj:14819, nnz_obs:14
```

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

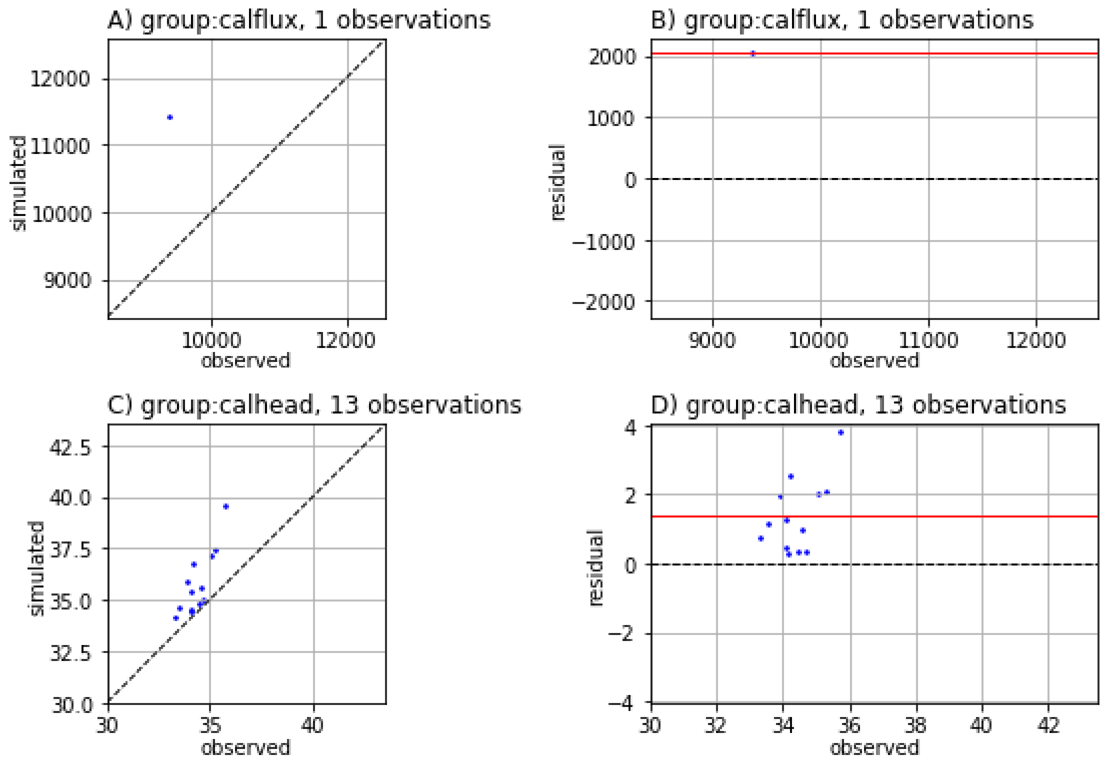
In [17]:
```
pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
print(pst.phi)
plt.figure()
pst.plot(kind='phi_pie')
print('Here are the non-zero weighted observation names')

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

```
4241857.402867099
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  9371.364052  11430.000000
hds_00_002_009_000  hds_00_002_009_000  calhead    35.080310     37.107498
hds_00_002_015_000  hds_00_002_015_000  calhead    34.692246     35.045185
hds_00_003_008_000  hds_00_003_008_000  calhead    35.282447     37.397289
hds_00_009_001_000  hds_00_009_001_000  calhead    35.703399     39.546417
hds_00_013_010_000  hds_00_013_010_000  calhead    34.606584     35.571774
hds_00_015_016_000  hds_00_015_016_000  calhead    34.473888     34.835716
hds_00_021_010_000  hds_00_021_010_000  calhead    34.114255     35.386250
hds_00_022_015_000  hds_00_022_015_000  calhead    34.123921     34.577492
hds_00_024_004_000  hds_00_024_004_000  calhead    34.193694     36.760464
hds_00_026_006_000  hds_00_026_006_000  calhead    33.947487     35.896149
hds_00_029_015_000  hds_00_029_015_000  calhead    34.138439     34.453842
hds_00_033_007_000  hds_00_033_007_000  calhead    33.533066     34.678810
hds_00_034_010_000  hds_00_034_010_000  calhead    33.346732     34.118073

                       residual  weight
name
fo_39_19791230     -2058.635948     1.0
hds_00_002_009_000    -2.027188    10.0
hds_00_002_015_000    -0.352939    10.0
hds_00_003_008_000    -2.114843    10.0
hds_00_009_001_000    -3.843018    10.0
hds_00_013_010_000    -0.965190    10.0
hds_00_015_016_000    -0.361828    10.0
hds_00_021_010_000    -1.271995    10.0
hds_00_022_015_000    -0.453571    10.0
hds_00_024_004_000    -2.566770    10.0
hds_00_026_006_000    -1.948662    10.0
hds_00_029_015_000    -0.315403    10.0
hds_00_033_007_000    -1.145744    10.0
hds_00_034_010_000    -0.771341    10.0
```

Publication ready figs - oh snap!

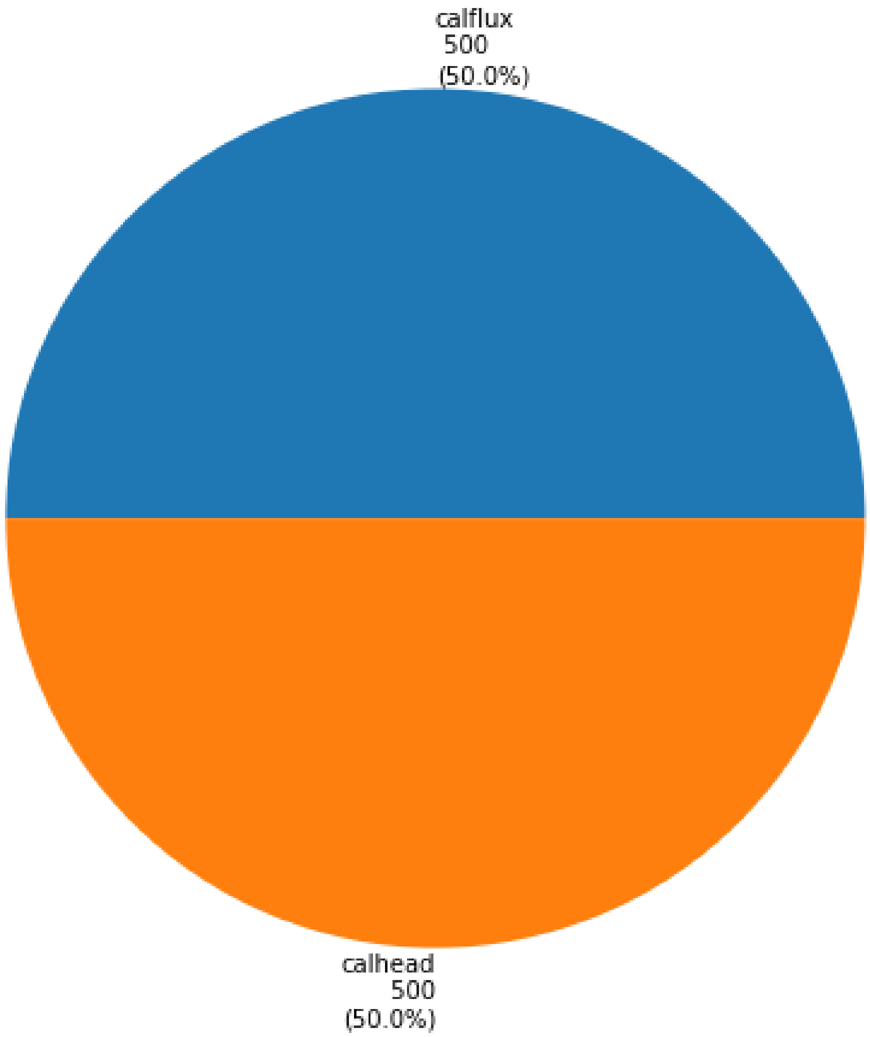
Depending on the truth you chose, we may have a problem - we set the weights for both the heads and the flux to reasonable values based on what we expect for measurement noise. But the contributions to total phi might be out of balance - if contribution of the flux measurement to total phi is too low, 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 0x181fd71828>
```



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          0.010862
         hds_00_002_009_000      3.591903
         hds_00_002_015_000      3.591903
         hds_00_003_008_000      3.591903
         hds_00_009_001_000      3.591903
         hds_00_013_010_000      3.591903
         hds_00_015_016_000      3.591903
         hds_00_021_010_000      3.591903
         hds_00_022_015_000      3.591903
```

```
hds_00_024_004_000    3.591903
hds_00_026_006_000    3.591903
hds_00_029_015_000    3.591903
hds_00_033_007_000    3.591903
hds_00_034_010_000    3.591903
Name: weight, dtype: float64
```

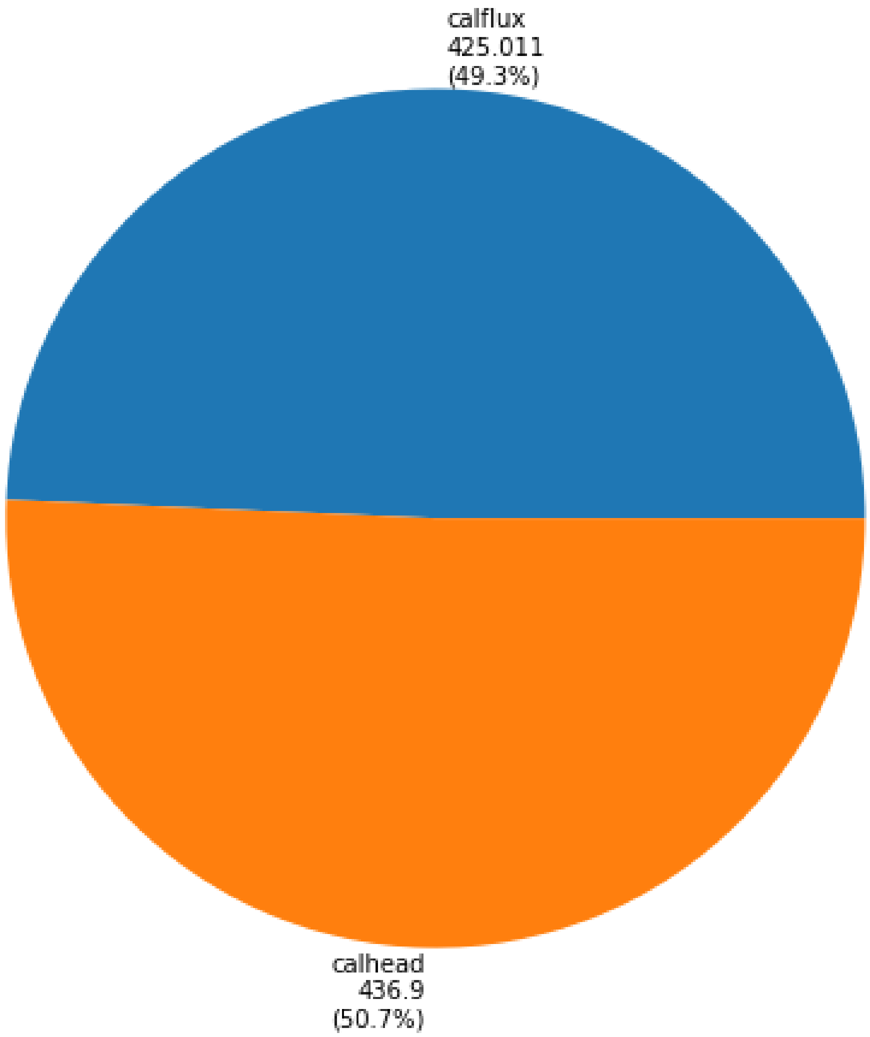
Now, for some super trickery: since we changed the weight, we need to generate the observation noise using these new weights for the error model (so meta!)

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        162.407476
hds_00_002_009_000      0.111405
hds_00_002_015_000      0.272485
hds_00_003_008_000      0.623873
hds_00_009_001_000      0.519936
hds_00_013_010_000     -0.272078
hds_00_015_016_000      0.264508
hds_00_021_010_000     -0.042138
hds_00_022_015_000     -0.028737
hds_00_024_004_000      0.114312
hds_00_026_006_000      0.040102
hds_00_029_015_000      0.404875
hds_00_033_007_000      0.211876
hds_00_034_010_000      0.033875
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
861.9109104046662
```



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,:]
```

```
17.528847233972552
```

```
Out[23]:                                  name    group     measured     modelled  \
         name
         fo_39_19791230          fo_39_19791230  calflux  9532.007476  9369.600000
         hds_00_002_009_000  hds_00_002_009_000  calhead    35.151700    35.040295
         hds_00_002_015_000  hds_00_002_015_000  calhead    34.866856    34.594372
         hds_00_003_008_000  hds_00_003_008_000  calhead    35.682231    35.058357
         hds_00_009_001_000  hds_00_009_001_000  calhead    36.036579    35.516644
         hds_00_013_010_000  hds_00_013_010_000  calhead    34.432233    34.704311
         hds_00_015_016_000  hds_00_015_016_000  calhead    34.643388    34.378880
         hds_00_021_010_000  hds_00_021_010_000  calhead    34.087252    34.129391
         hds_00_022_015_000  hds_00_022_015_000  calhead    34.105506    34.134243
         hds_00_024_004_000  hds_00_024_004_000  calhead    34.266946    34.152634
         hds_00_026_006_000  hds_00_026_006_000  calhead    33.973185    33.933083
         hds_00_029_015_000  hds_00_029_015_000  calhead    34.397887    33.993011
         hds_00_033_007_000  hds_00_033_007_000  calhead    33.668838    33.456963
         hds_00_034_010_000  hds_00_034_010_000  calhead    33.368439    33.334564

                               residual    weight
         name
         fo_39_19791230      162.407476  0.010862
         hds_00_002_009_000    0.111405  3.591903
         hds_00_002_015_000    0.272485  3.591903
         hds_00_003_008_000    0.623873  3.591903
         hds_00_009_001_000    0.519936  3.591903
         hds_00_013_010_000   -0.272078  3.591903
         hds_00_015_016_000    0.264508  3.591903
         hds_00_021_010_000   -0.042138  3.591903
         hds_00_022_015_000   -0.028737  3.591903
         hds_00_024_004_000    0.114312  3.591903
         hds_00_026_006_000    0.040102  3.591903
         hds_00_029_015_000    0.404875  3.591903
         hds_00_033_007_000    0.211876  3.591903
         hds_00_034_010_000    0.033875  3.591903
```

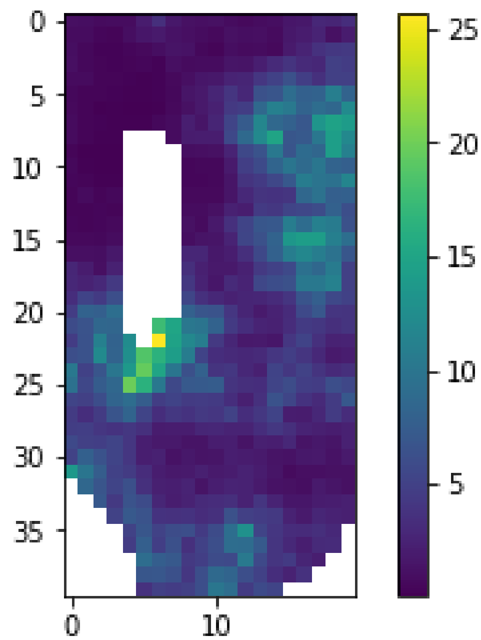
The residual should be exactly the noise values from above.Lets load the model (that was just run using the true pars) and check some things

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

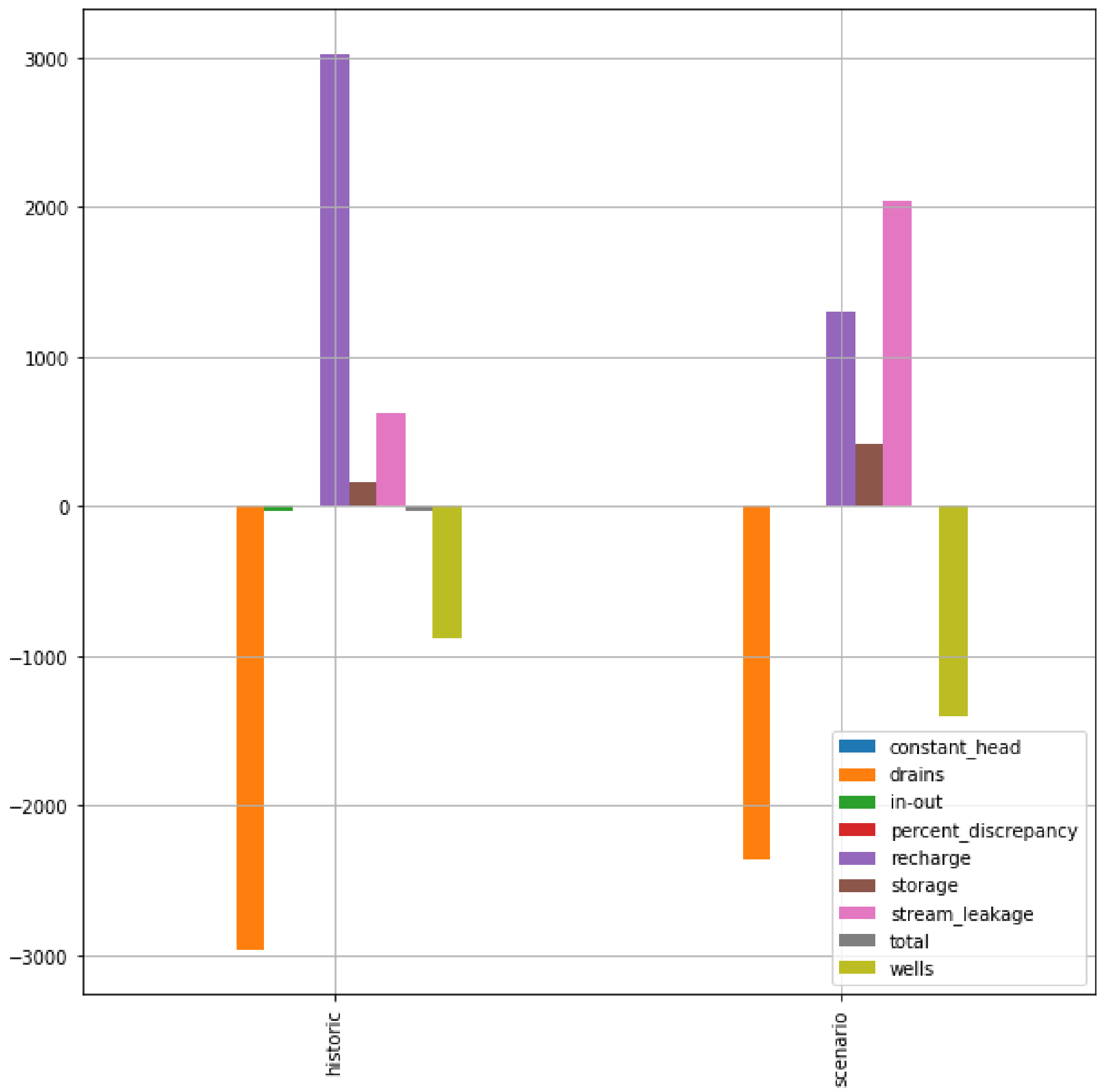
```
In [25]: a = m.upw.hk[0].array
         #a = m.rch.rech[0].array
```

```
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()
```

```
0.09253527 25.65187
```



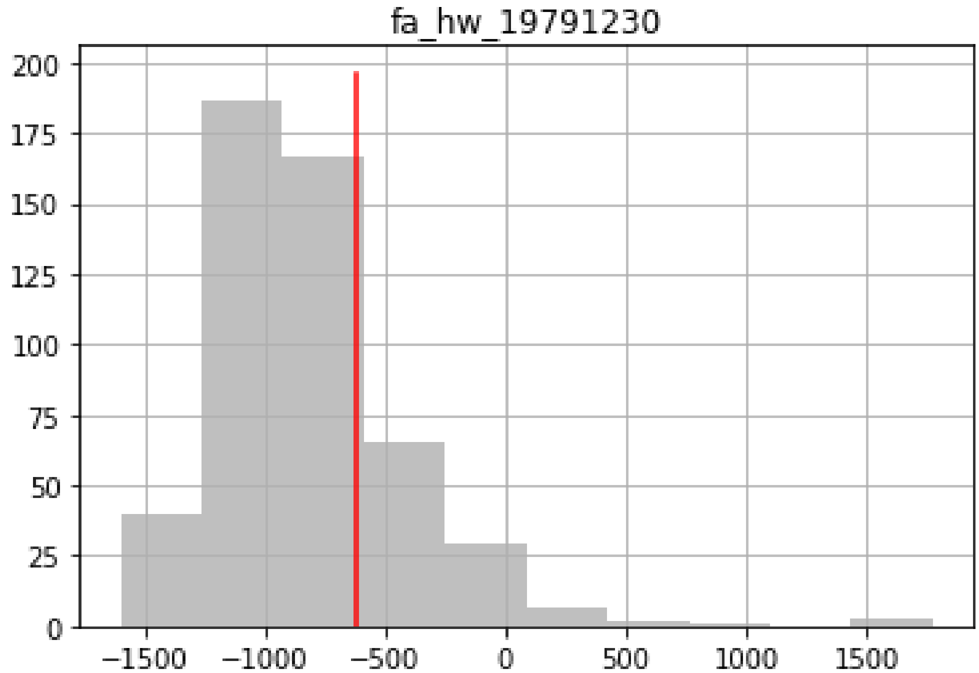

```
In [26]: lst = flopy.utils.MfListBudget(os.path.join(m_d,"freyberg.list"))
         df = lst.get_dataframes(diff=True)[0]
         ax = df.plot(kind="bar",figsize=(10,10), grid=True)
         a = ax.set_xticklabels(["historic","scenario"],rotation=90)
         plt.show(ax)
```

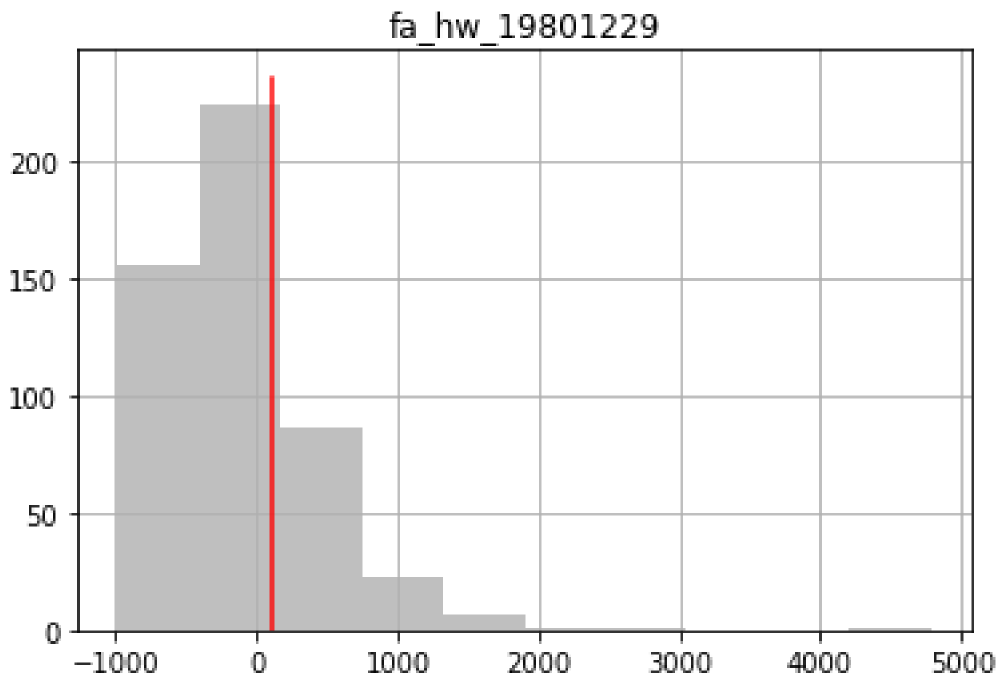


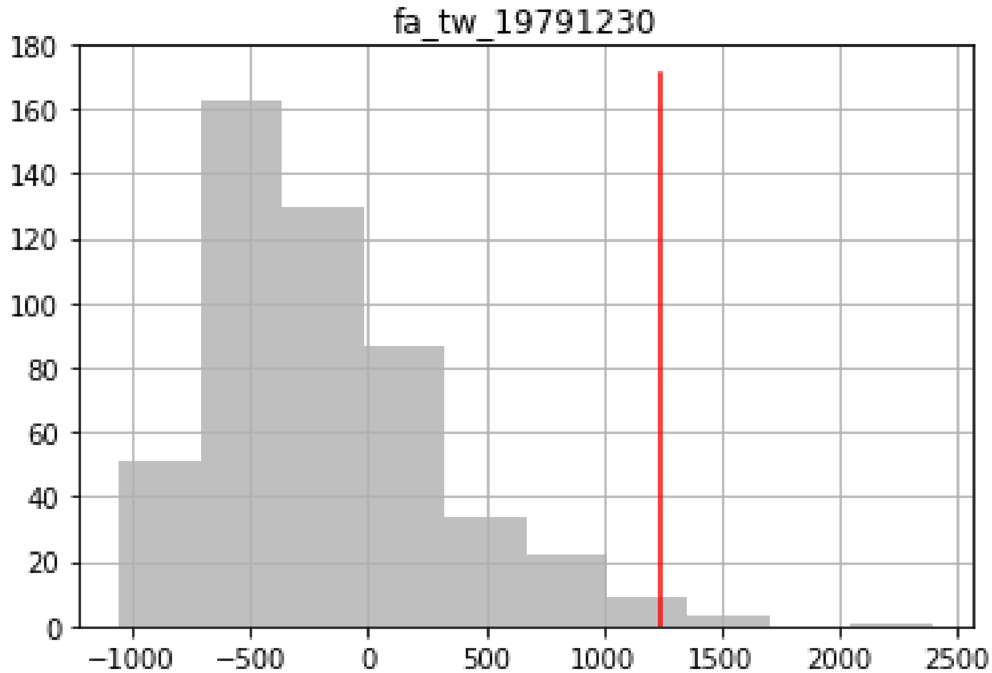
1.0.8 see how our existing observation ensemble compares to the truth

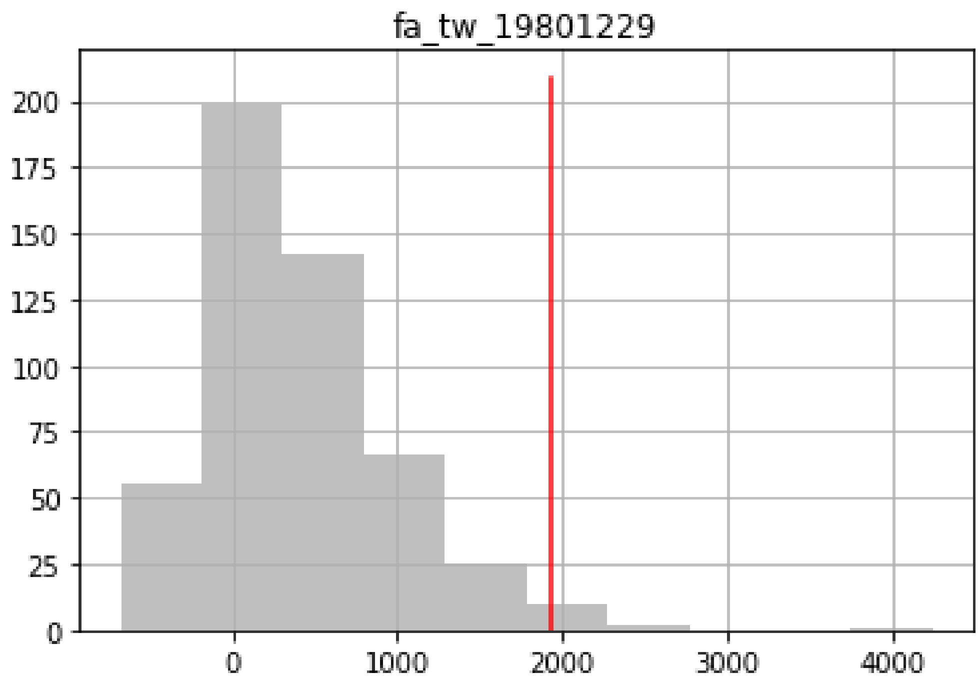
forecasts:

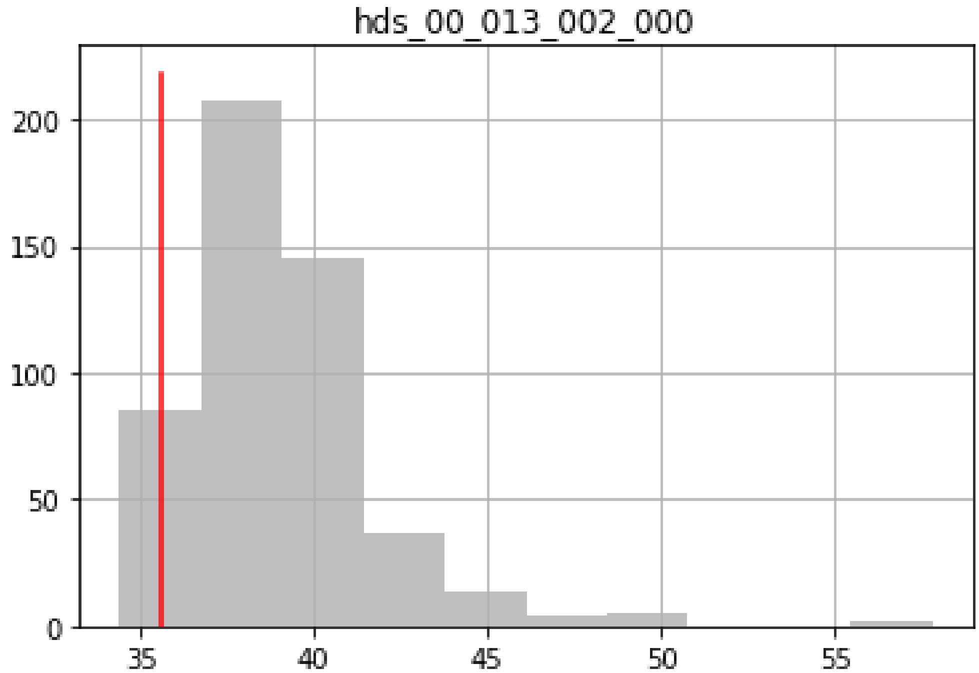
```
In [27]: obs = pst.observation_data
         plt.figure()
         for forecast in fnames:
             ax = plt.subplot(111)
             obs_df.loc[:,forecast].hist(ax=ax,color="0.5",alpha=0.5)
             ax.plot([obs.loc[forecast,"obsval"],obs.loc[forecast,"obsval"]],ax.get_ylim(),"r")
    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

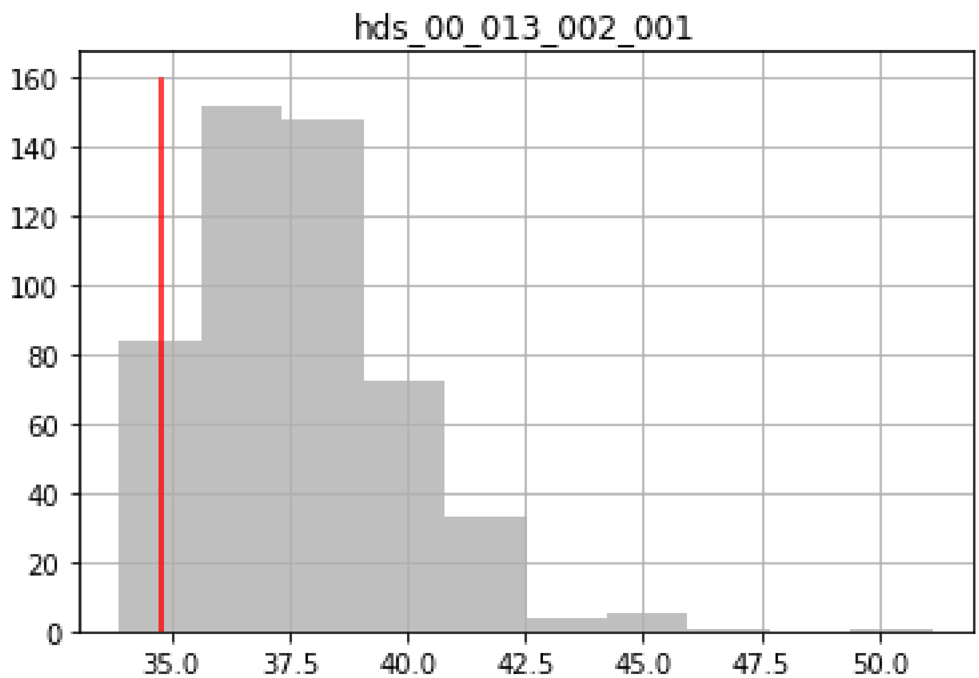

fa_hw_19791230
200
175
150
125
100
75
50
25
0
−1500
−1000
−500
0
500
1000
1500

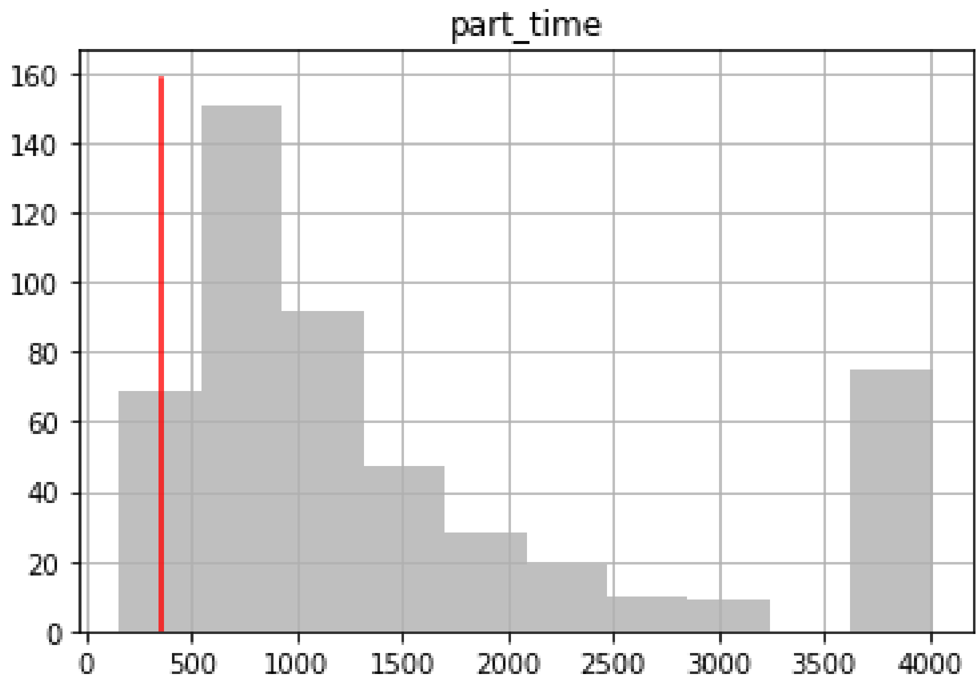

fa_hw_19801229
200
150
100
50
0
−1000
0
1000
2000
3000
4000
5000

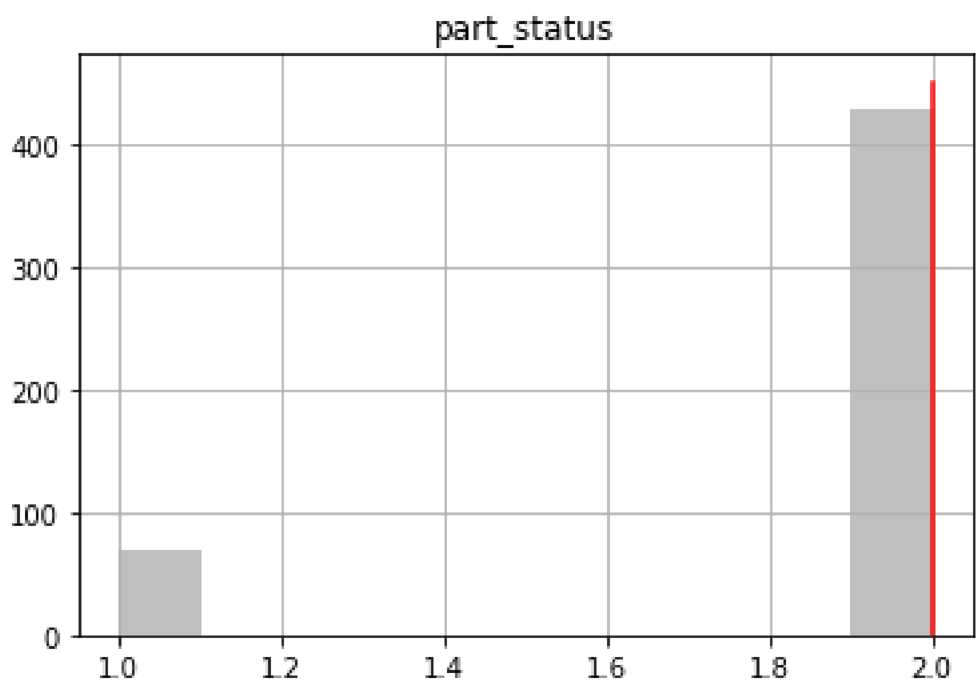

fa_tw_19791230
180
160
140
120
100
80
60
40
20
0
−1000
−500
0
500
1000
1500
2000
2500


fa_tw_19801229
200
175
150
125
100
75
50
25
0
0
1000
2000
3000
4000


hds_00_013_002_000
200
150
100
50
0
35
40
45
50
55

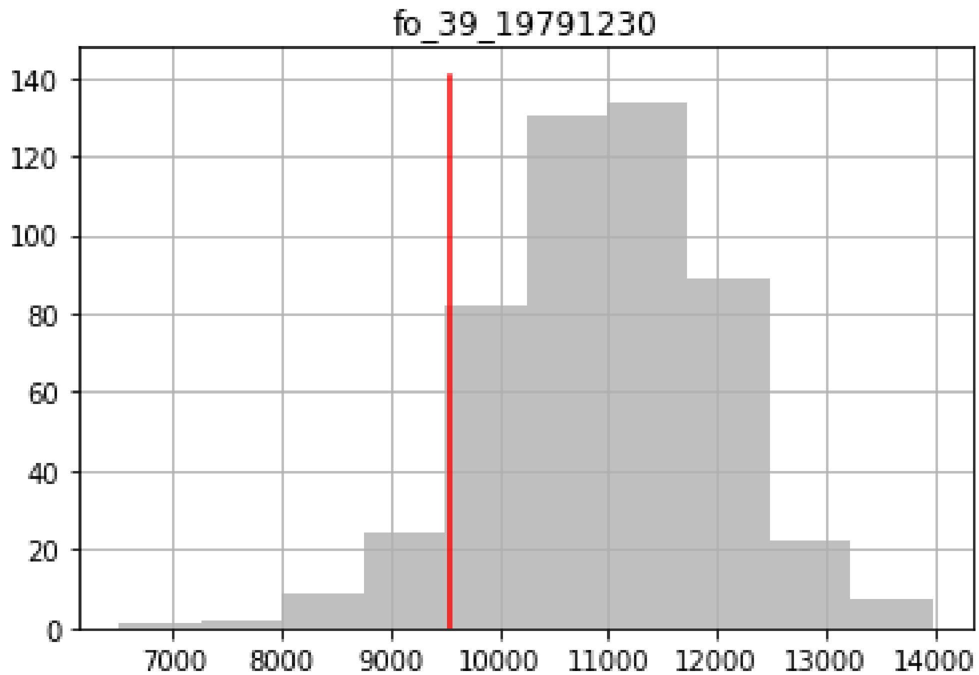

hds_00_013_002_001
160
140
120
100
80
60
40
20
0
35.0
37.5
40.0
42.5
45.0
47.5
50.0

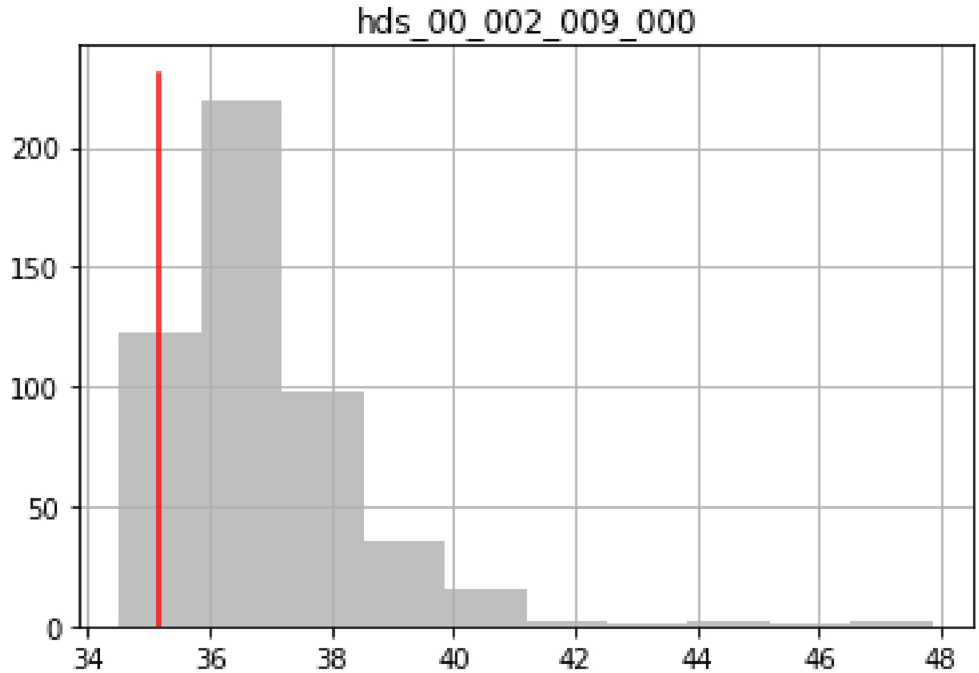


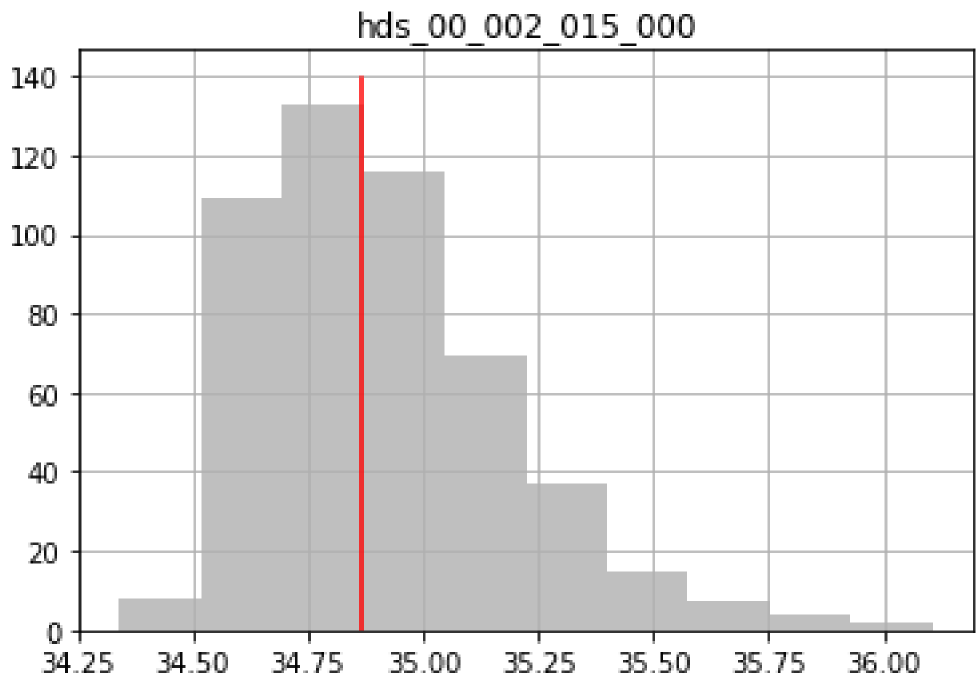


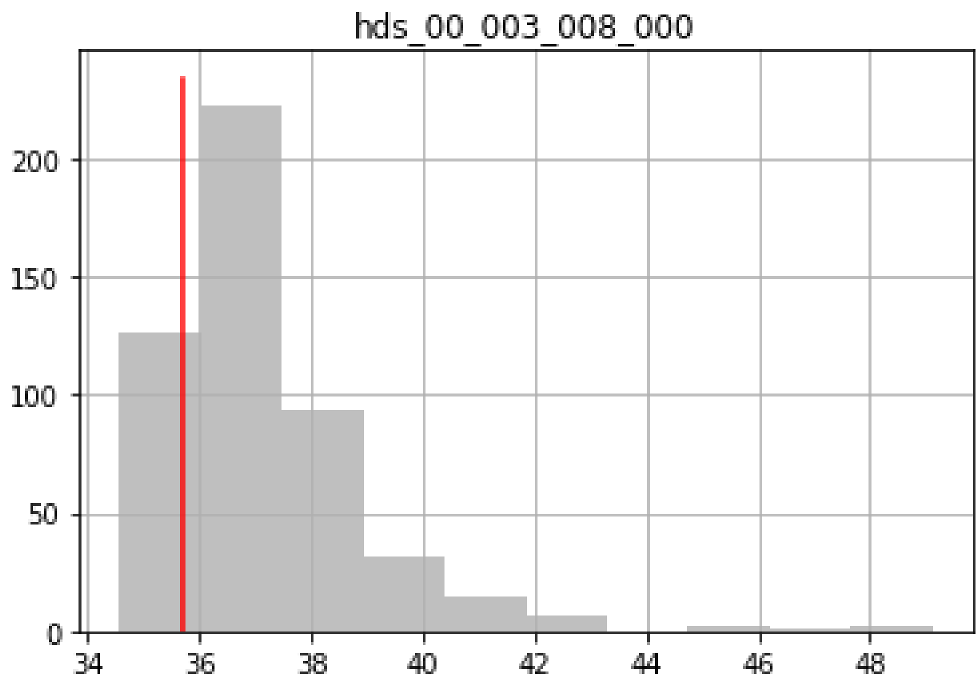
observations:

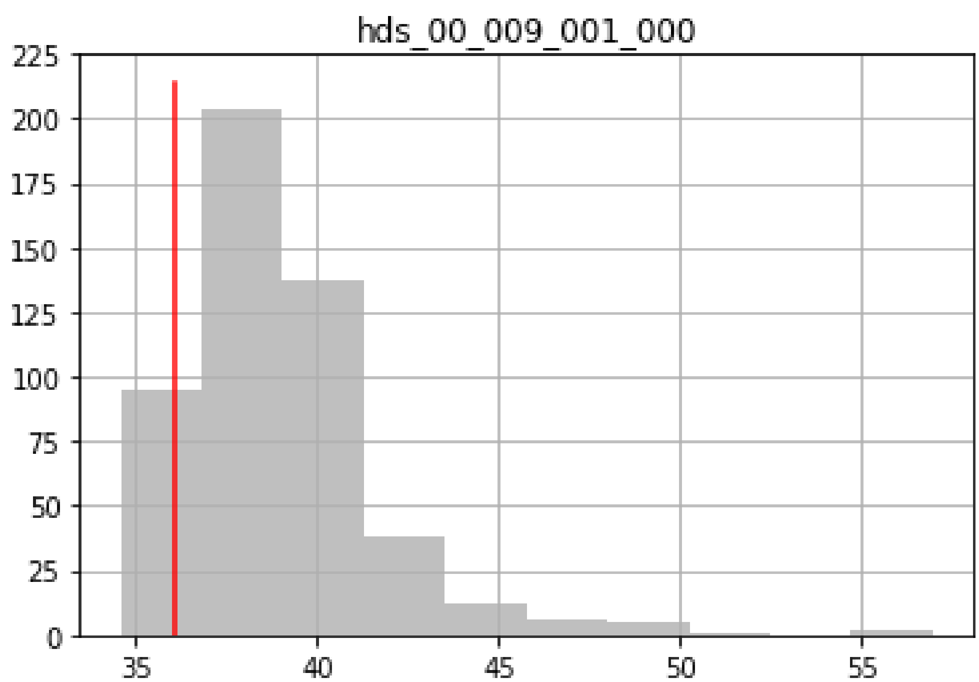
In [28]:
```
for oname in pst.nnz_obs_names:
    ax = plt.subplot(111)
    obs_df.loc[:,oname].hist(ax=ax,color="0.5",alpha=0.5)
    ax.plot([obs.loc[oname,"obsval"],obs.loc[oname,"obsval"]],ax.get_ylim(),"r")
    ax.set_title(oname)
    plt.show()
```

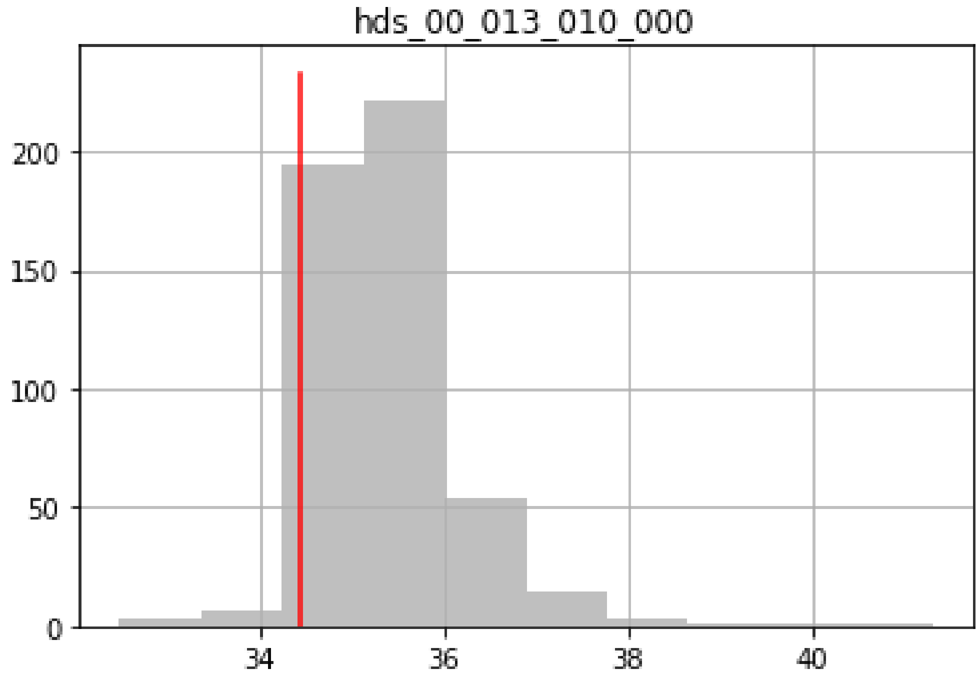


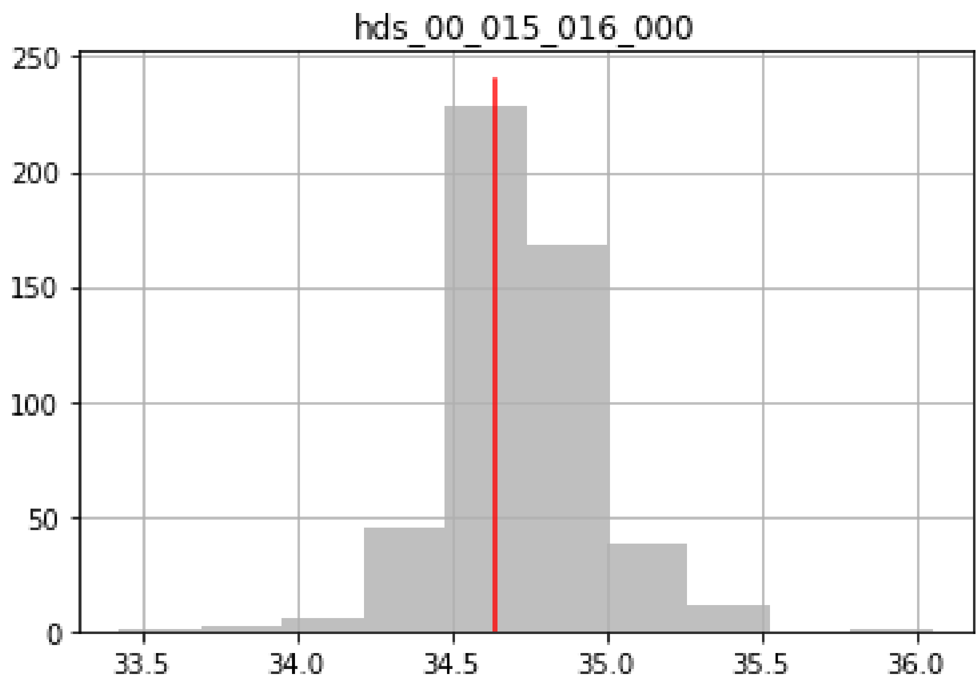

hds_00_002_009_000
0
50
100
150
200
34
36
38
40
42
44
46
48

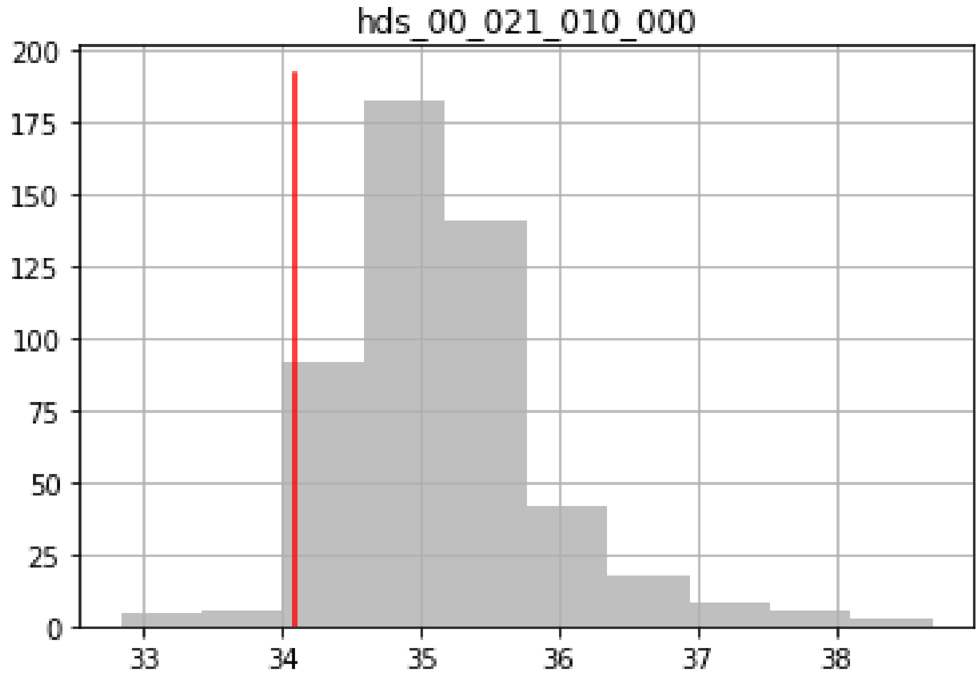

hds_00_002_015_000
0
20
40
60
80
100
120
140
34.25
34.50
34.75
35.00
35.25
35.50
35.75
36.00

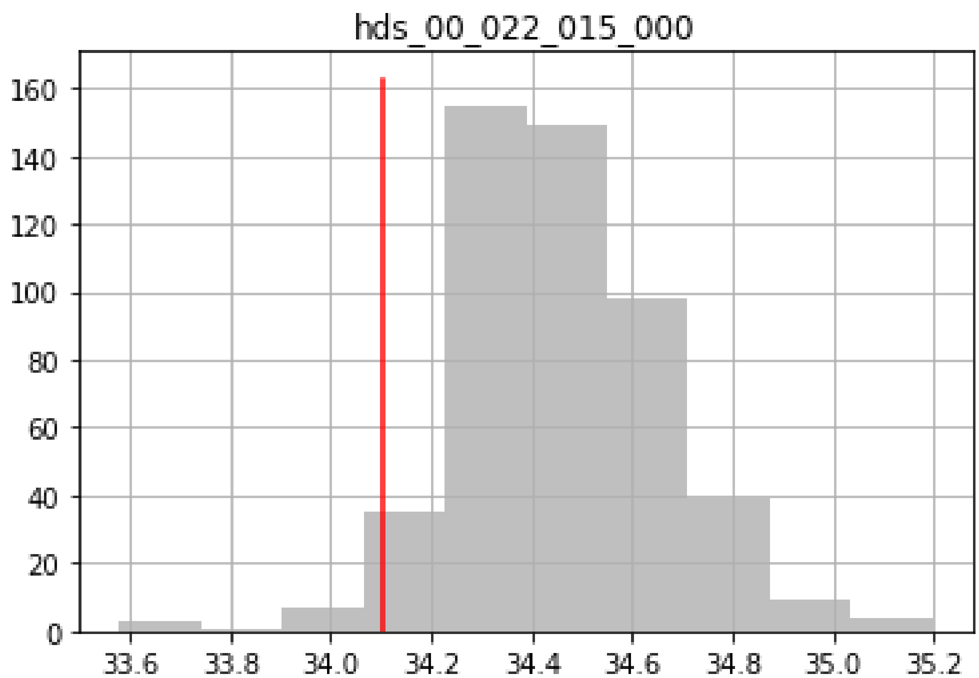

hds_00_003_008_000
200
150
100
50
0
34
36
38
40
42
44
46
48

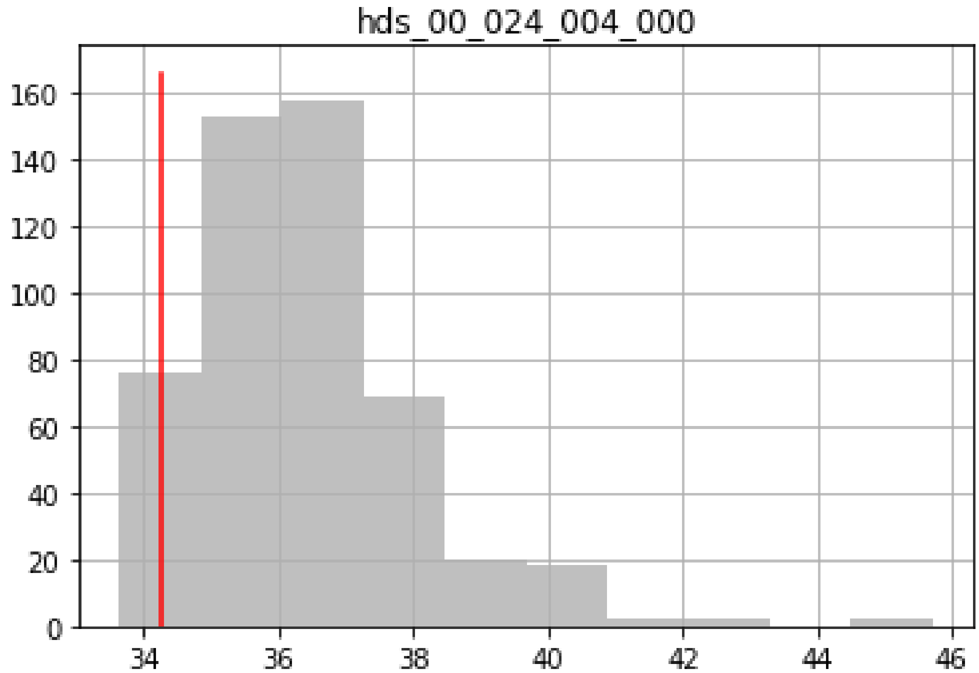

hds_00_009_001_000
225
200
175
150
125
100
75
50
25
0
35
40
45
50
55

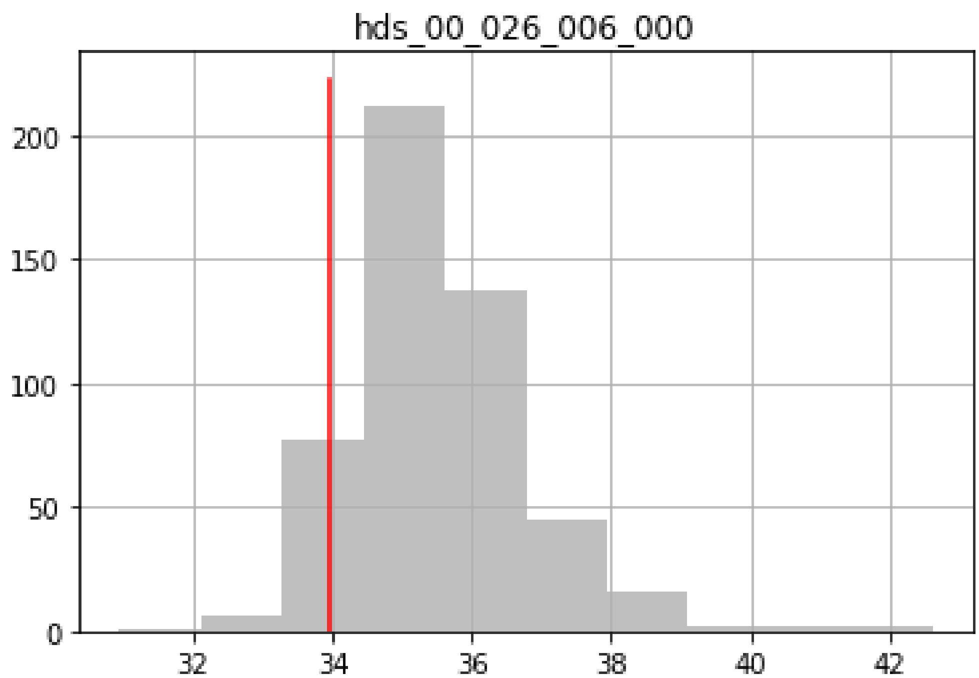

hds_00_013_010_000
0
50
100
150
200
34
36
38
40

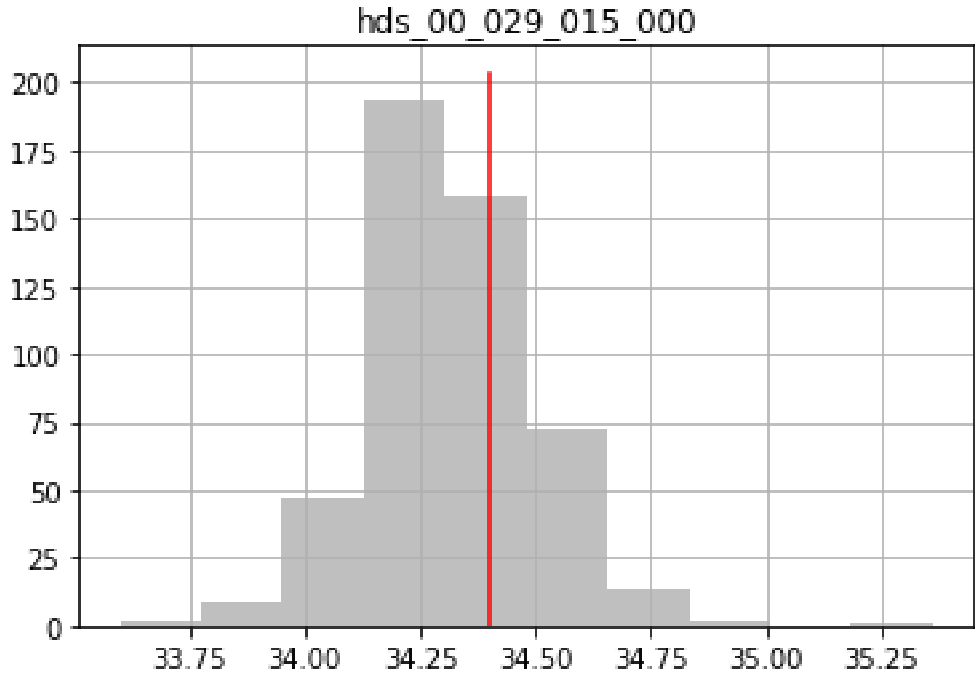

hds_00_015_016_000
0
50
100
150
200
250
33.5
34.0
34.5
35.0
35.5
36.0

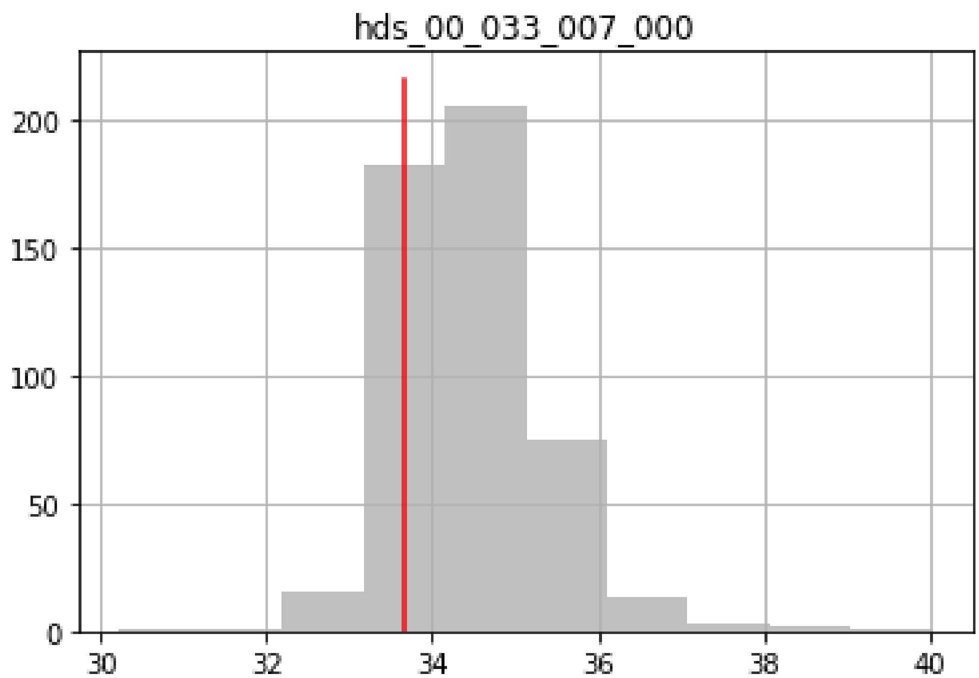

hds_00_021_010_000
0
25
50
75
100
125
150
175
200
33
34
35
36
37
38

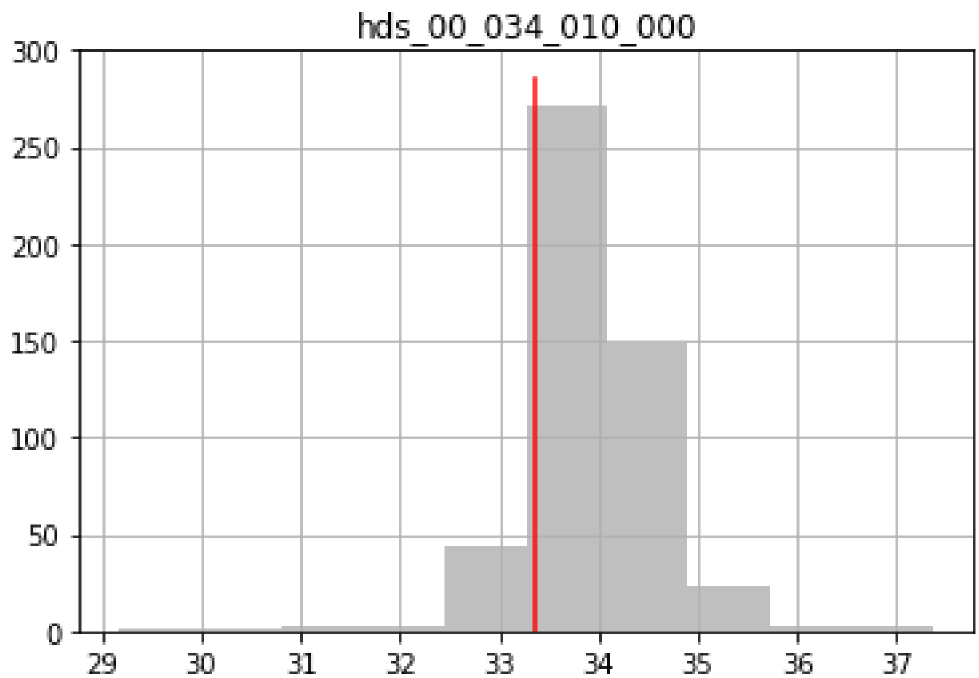

hds_00_022_015_000
0
20
40
60
80
100
120
140
160
33.6
33.8
34.0
34.2
34.4
34.6
34.8
35.0
35.2


hds_00_024_004_000
0
20
40
60
80
100
120
140
160
34
36
38
40
42
44
46


hds_00_026_006_000
0
50
100
150
200
32
34
36
38
40
42


hds_00_029_015_000
200
175
150
125
100
75
50
25
0
33.75
34.00
34.25
34.50
34.75
35.00
35.25


hds_00_033_007_000
200
150
100
50
0
30
32
34
36
38
40


hds_00_034_010_000
300
250
200
150
100
50
0
29
30
31
32
33
34
35
36
37