prior_montecarlo

July 1, 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
        %matplotlib inline
```

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

1.1 SUPER IMPORTANT: SET HOW MANY PARALLEL WORKERS TO USE

```
In [2]: num_workers = 10
```

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

```
In [3]: t_d = "template"
        pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
```

Load the previously generated parameter ensemble and inspect...

```
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"))
        pe.shape
```

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

Out[4]: (500, 14819)

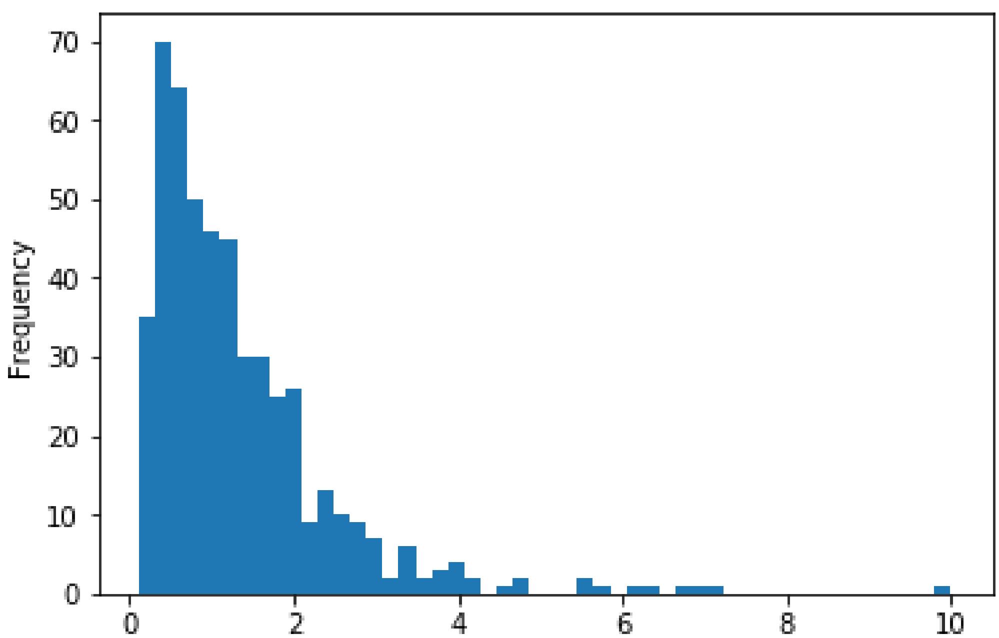
In [5]: `pe.loc[:,"hk031"]`

```
Out[5]: 0      1.825601
        1      0.492395
        2      1.732541
        3      0.705128
        4      2.297188
        5      1.252259
        6      0.516871
        7      2.804304
        8      0.901501
        9      0.687264
        10     0.560744
        11     1.109859
        12     3.769390
        13     5.603115
        14     0.498500
        15     1.641592
        16     1.671183
        17     0.789444
        18     2.699144
        19     0.597935
        20     0.509815
        21     0.599028
        22     1.237666
        23     1.282956
        24     0.626140
        25     1.804318
        26     0.800346
        27     3.591938
        28     1.315121
        29     0.407926
                 ...
        470    2.182434
        471    0.428269
        472    0.315921
        473    1.160333
        474    2.349660
        475    4.092183
        476    0.965726
        477    0.541814
        478    0.353042
        479    0.988625
```

```
480    0.292822
481    0.369399
482    2.130098
483    0.641830
484    1.972197
485    0.783833
486    2.121167
487    4.772154
488    1.429077
489    2.661986
490    0.995772
491    1.386897
492    1.510872
493    1.115474
494    2.634503
495    1.353126
496    1.003978
497    0.668727
498    0.492995
499    1.109220
Name: hk031, Length: 500, dtype: float64
```

In [6]:
```
pe.loc[:,"hk031"].plot.hist(bins=50)
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1817cef1d0>
```



look! hk is log-normal-ish
Lets run the first realization through the pest interface for a test:

In [7]:
```
# replace the par vals witht the first row in the par ensemble
pst.parameter_data.loc[pe.columns,"parval1"] = pe.iloc[0,:]
pst.control_data.noptmax = 0
pst.write(os.path.join(t_d,"test.pst"))
pyemu.os_utils.run("pestpp-ies test.pst",cwd=t_d)
res = pyemu.pst_utils.read_resfile(os.path.join(t_d,"test.base.rei"))
res.loc[pst.nnz_obs_names,:]
```

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

Out[7]:
```
                                   name    group     measured     modelled  \
name
fo_39_19791230           fo_39_19791230  calflux  11430.000000  9598.300000
hds_00_002_009_000   hds_00_002_009_000  calhead     37.107498    37.257839
hds_00_002_015_000   hds_00_002_015_000  calhead     35.045185    34.803825
hds_00_003_008_000   hds_00_003_008_000  calhead     37.397289    37.389198
hds_00_009_001_000   hds_00_009_001_000  calhead     39.546417    38.189617
hds_00_013_010_000   hds_00_013_010_000  calhead     35.571774    35.907837
hds_00_015_016_000   hds_00_015_016_000  calhead     34.835716    35.016644
hds_00_021_010_000   hds_00_021_010_000  calhead     35.386250    35.802383
hds_00_022_015_000   hds_00_022_015_000  calhead     34.577492    34.437019
hds_00_024_004_000   hds_00_024_004_000  calhead     36.760464    36.456657
hds_00_026_006_000   hds_00_026_006_000  calhead     35.896149    35.731762
hds_00_029_015_000   hds_00_029_015_000  calhead     34.453842    34.589817
hds_00_033_007_000   hds_00_033_007_000  calhead     34.678810    34.782738
hds_00_034_010_000   hds_00_034_010_000  calhead     34.118073    34.351669

                       residual  weight
name
fo_39_19791230      1831.700000     1.0
hds_00_002_009_000    -0.150341     1.0
hds_00_002_015_000     0.241360     1.0
hds_00_003_008_000     0.008091     1.0
hds_00_009_001_000     1.356800     1.0
hds_00_013_010_000    -0.336063     1.0
hds_00_015_016_000    -0.180927     1.0
hds_00_021_010_000    -0.416134     1.0
hds_00_022_015_000     0.140472     1.0
hds_00_024_004_000     0.303806     1.0
hds_00_026_006_000     0.164387     1.0
hds_00_029_015_000    -0.135975     1.0
hds_00_033_007_000    -0.103928     1.0
hds_00_034_010_000    -0.233597     1.0
```

1.1.2 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 [8]: m_d = "master_prior_sweep"
        pyemu.os_utils.start_slaves(t_d,"pestpp-swp","freyberg.pst",num_slaves=num_workers,sla
```

1.1.3 Load the output ensemble and plot a few things

```
In [9]: 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
```

1.1.4 drop any failed runs

```
In [10]: 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: 499
```

```
In [11]: obs_df.iloc[0,:]
```

```
Out[11]: input_run_id           0.000000e+00
         failed_flag            0.000000e+00
         phi                    3.355127e+06
         meas_phi               3.355127e+06
         regul_phi              0.000000e+00
         flx_recharg            0.000000e+00
         flx_in-out             0.000000e+00
         vol_total              0.000000e+00
         flx_wells              0.000000e+00
         flx_constan            0.000000e+00
         obgnme                 0.000000e+00
         vol_recharg            0.000000e+00
         flout                  0.000000e+00
         calflux                3.355125e+06
         vol_wells              0.000000e+00
         flx_percent            0.000000e+00
         vol_percent            0.000000e+00
         vol_in-out             0.000000e+00
         calhead                2.463581e+00
         flaqx                  0.000000e+00
         flx_drains             0.000000e+00
         vol_storage            0.000000e+00
         vol_constan            0.000000e+00
         flx_total              0.000000e+00
         hds                    0.000000e+00
         vol_stream_            0.000000e+00
```

```
flx_storage              0.000000e+00
flx_stream_              0.000000e+00
vol_drains               0.000000e+00
fa_0_19791230           -8.355700e+01
                             ...
hds_02_039_010_000       3.258115e+01
hds_02_039_010_001       3.256272e+01
hds_02_039_011_000       3.256712e+01
hds_02_039_011_001       3.255236e+01
hds_02_039_012_000       3.256037e+01
hds_02_039_012_001       3.254858e+01
hds_02_039_013_000       3.255714e+01
hds_02_039_013_001       3.254855e+01
hds_02_039_014_000       3.255387e+01
hds_02_039_014_001       3.254865e+01
vol_constan_19791230     0.000000e+00
vol_constan_19801229     0.000000e+00
vol_drains_19791230     -1.930785e+06
vol_drains_19801229     -2.079174e+06
vol_in-out_19791230     -6.532200e+04
vol_in-out_19801229     -6.518400e+04
vol_percent_19791230    -5.500000e-01
vol_percent_19801229    -5.100000e-01
vol_recharg_19791230     1.116655e+07
vol_recharg_19801229     1.195581e+07
vol_storage_19791230     6.156623e+05
vol_storage_19801229     7.822023e+05
vol_stream__19791230    -5.239726e+06
vol_stream__19801229    -5.598338e+06
vol_total_19791230      -6.532200e+04
vol_total_19801229      -6.518400e+04
vol_wells_19791230      -4.677026e+06
vol_wells_19801229      -5.125686e+06
part_status              2.000000e+00
part_time                7.860861e+02
Name: 0, Length: 4465, dtype: float64
```

1.1.5 confirm which quantities were identified as forecasts

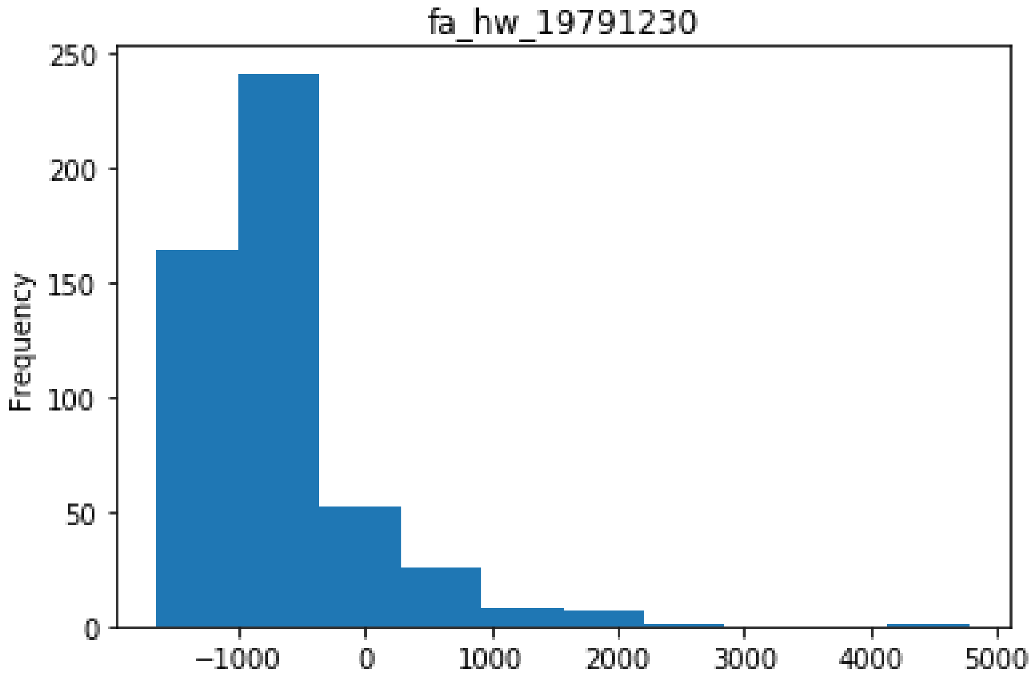
In [12]:
```
fnames = pst.pestpp_options["forecasts"].split(',')
fnames
```

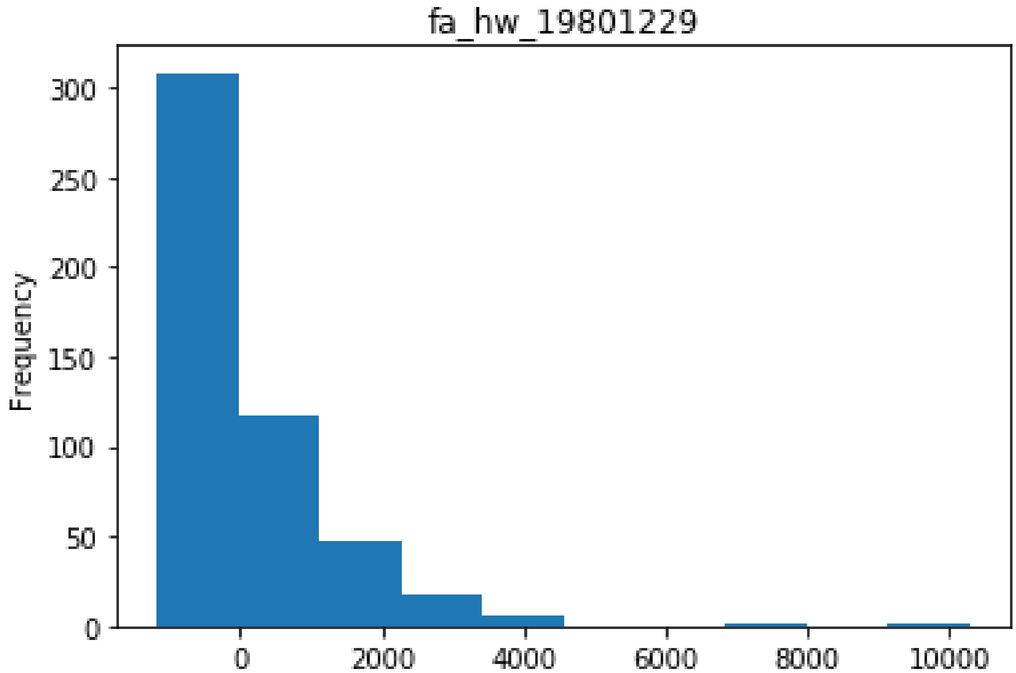
Out[12]:
```
['fa_hw_19791230',
 'fa_hw_19801229',
 'fa_tw_19791230',
 'fa_tw_19801229',
 'hds_00_013_002_000',
 'hds_00_013_002_001',
```

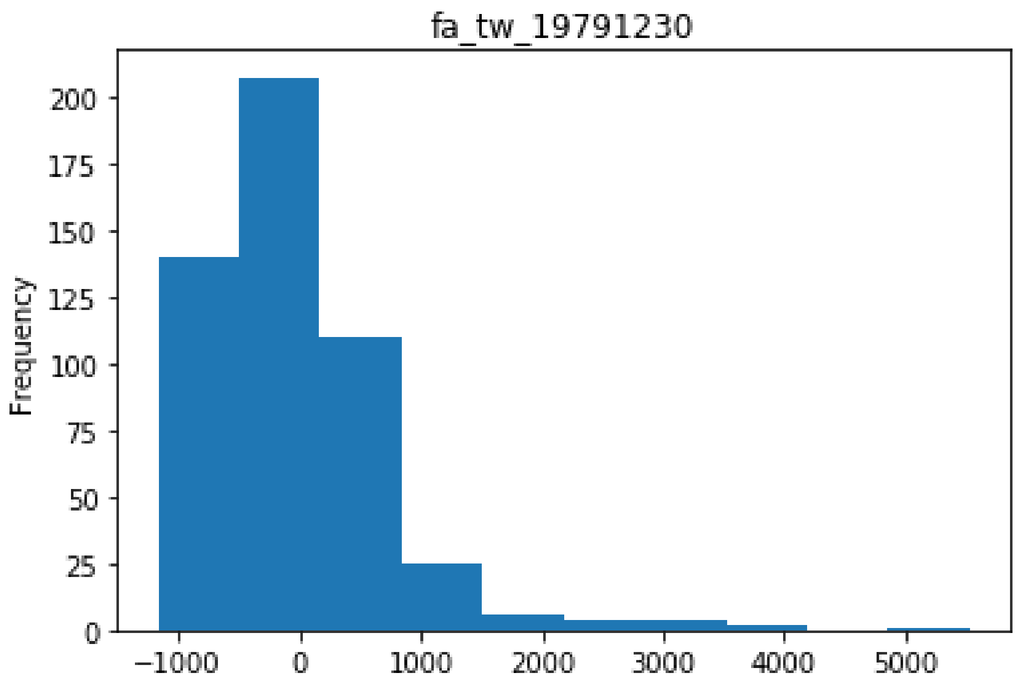
```
'part_time',
'part_status']
```

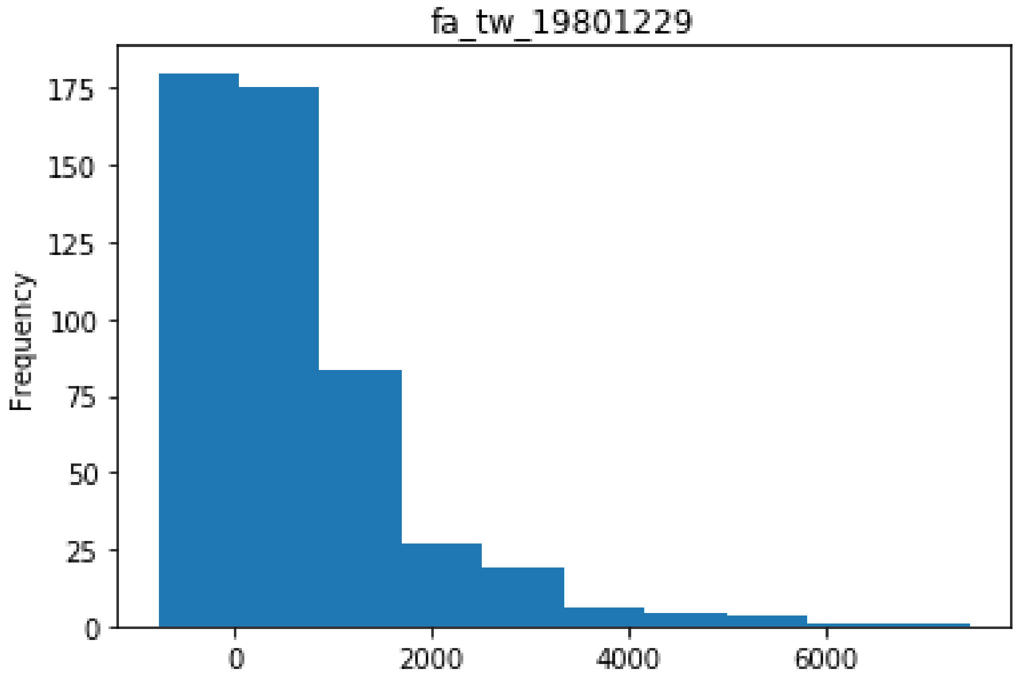
1.1.6 now we can plot the distributions of each forecast

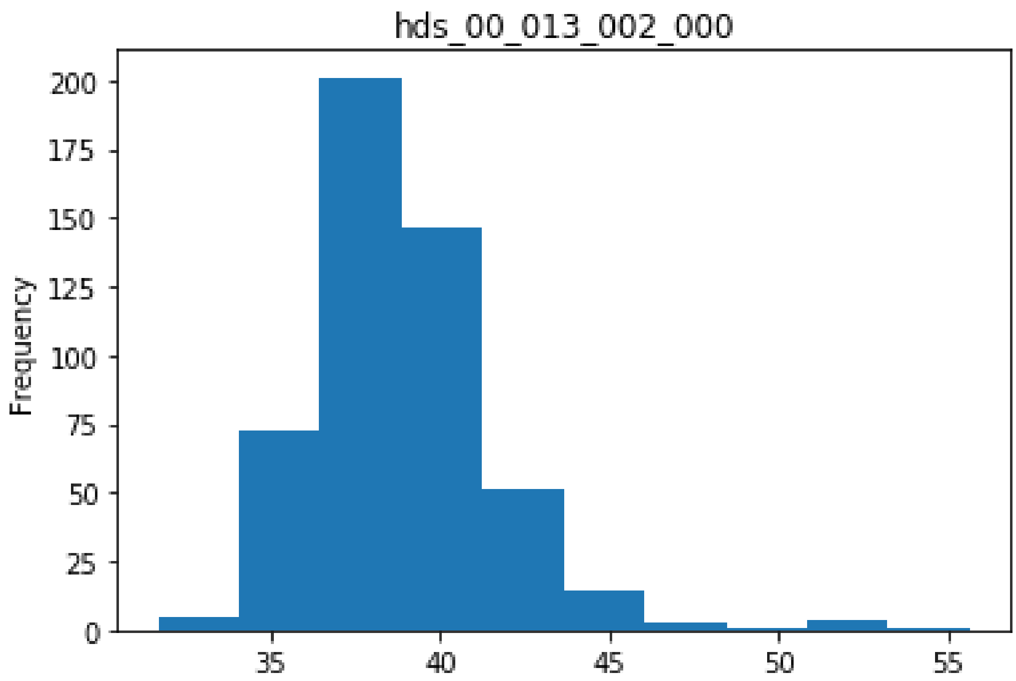
```
In [13]: for forecast in fnames:
             plt.figure()
             ax = obs_df.loc[:,forecast].plot(kind="hist")
             ax.set_title(forecast)
```

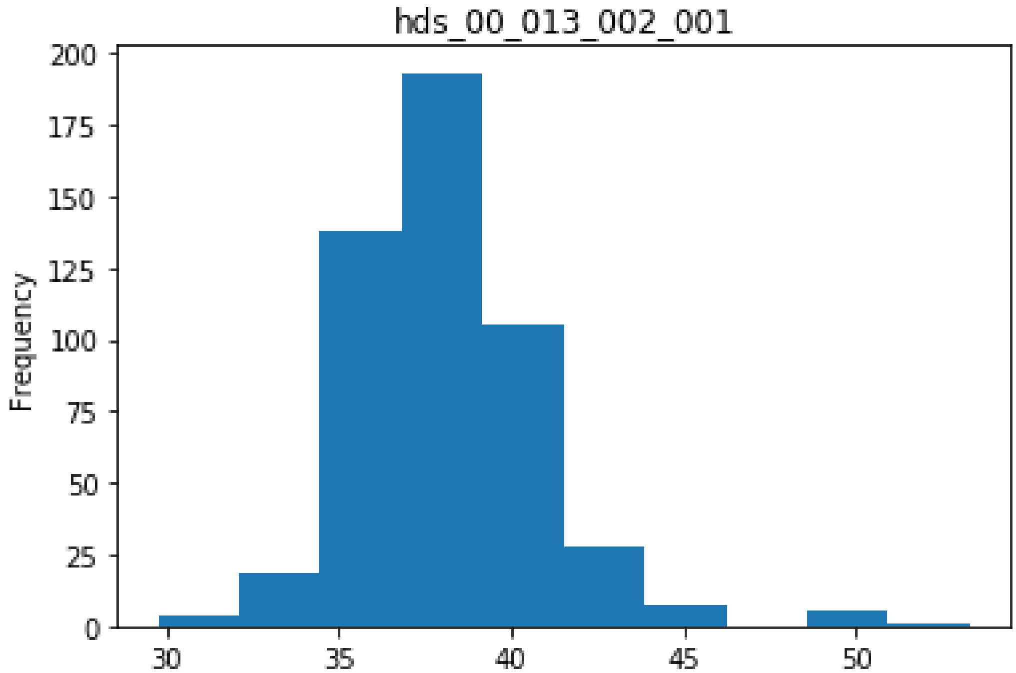


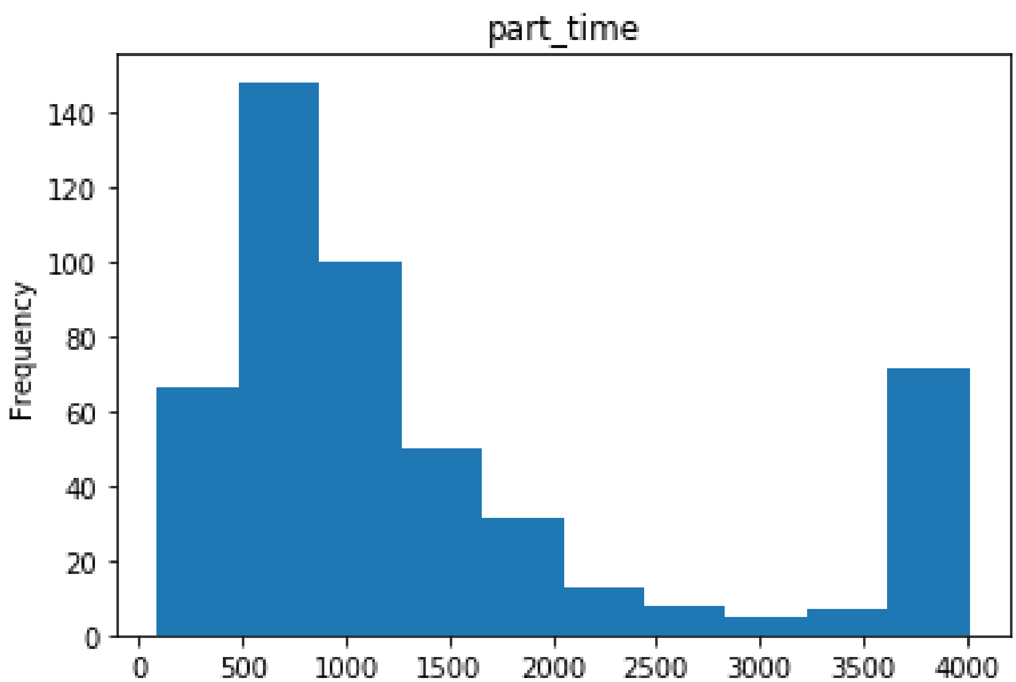

fa_hw_19801229
Frequency
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000

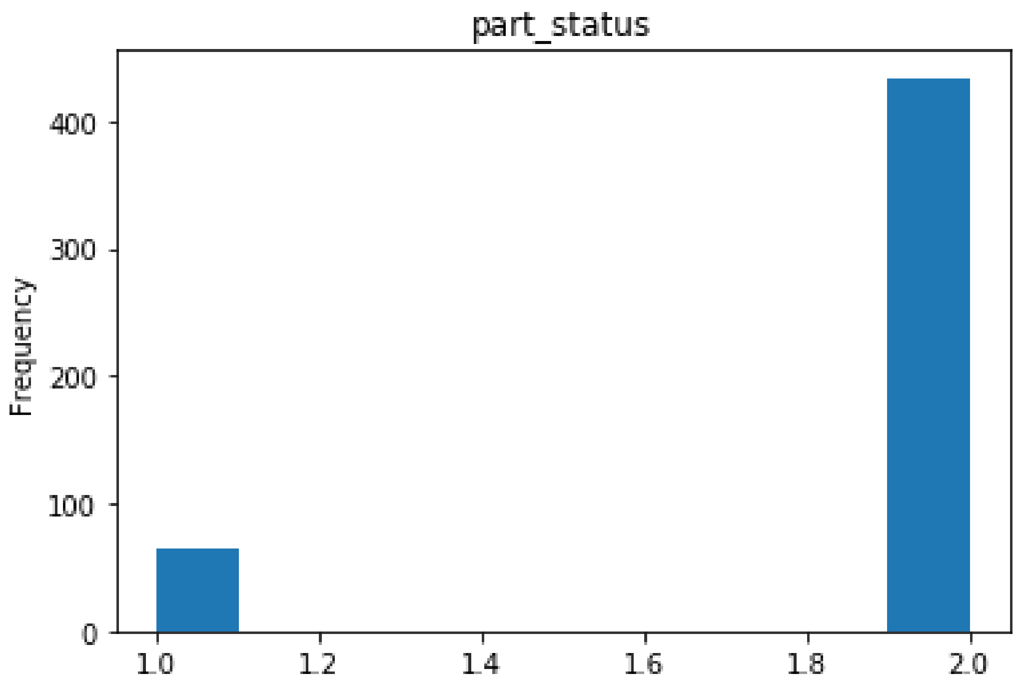

fa_tw_19791230
Frequency
200
175
150
125
100
75
50
25
0
−1000
0
1000
2000
3000
4000
5000


fa_tw_19801229
Frequency
175
150
125
100
75
50
25
0
0
2000
4000
6000


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
200
175
150
125
100
75
50
25
0
30
35
40
45
50

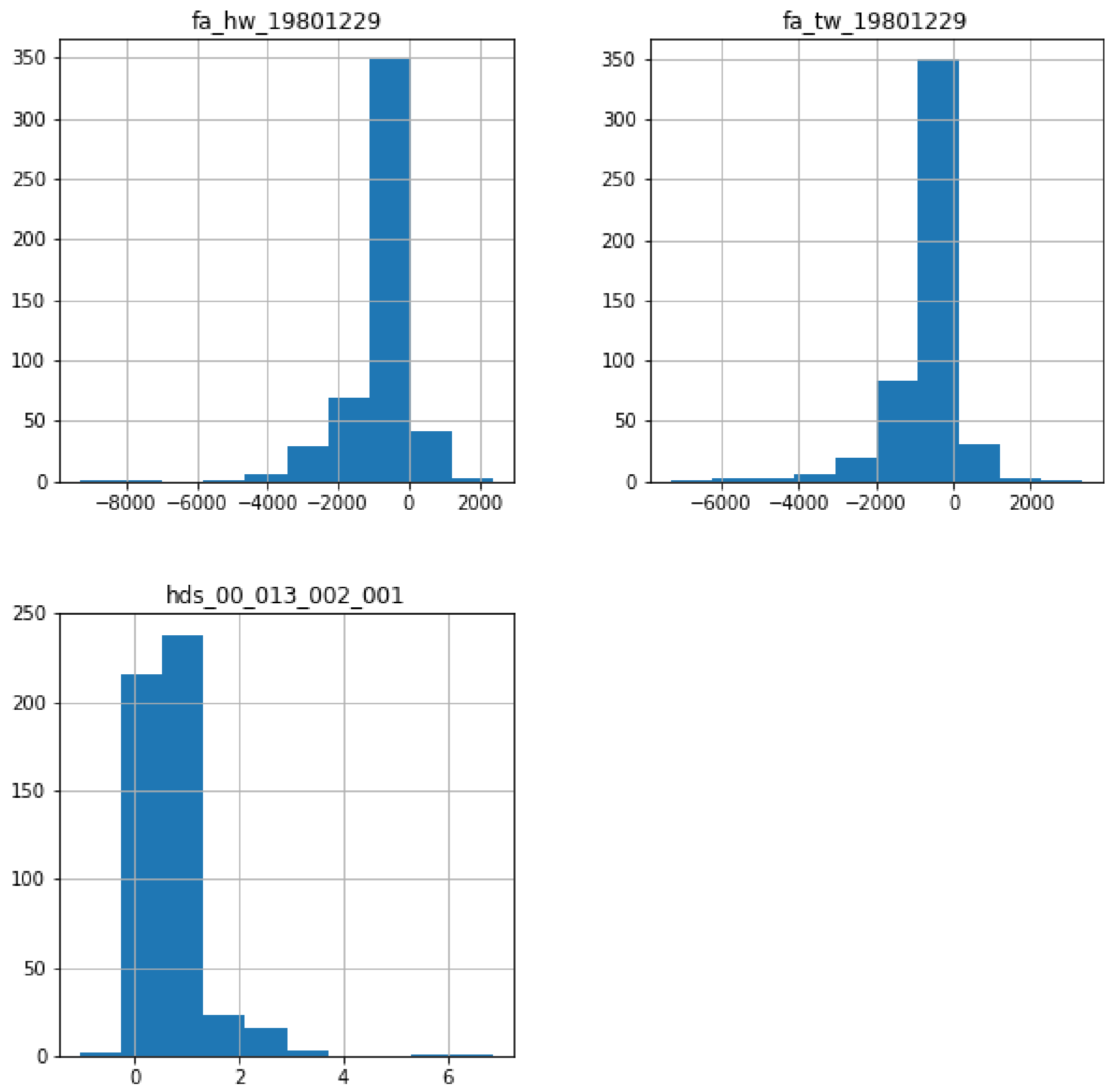

part_time
Frequency
140
120
100
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500
4000



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 [14]: 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))
```

```
Out[14]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1817751ba8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x1817e96240>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x1817eac6d8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x1818724c50>]],
               dtype=object)
```



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

1.1.7 Most modeling analyses should stop right here to avoid the ill-effects of history matching…

1.1.8 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 [15]: fnames
```

Out[15]: ['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']

```
In [16]: sorted_vals = obs_df.loc[:,"part_time"].sort_values()
         idx = sorted_vals.index[50]
         idx
```

Out[16]: 163

```
In [17]: sorted_vals
```

```
Out[17]: run_id
         262     90.16512
         3      113.14530
         335    143.61510
         92     155.49960
         101    199.40370
         170    206.95530
         330    223.15580
         302    233.50220
         60     238.79050
         97     242.29560
         69     246.31940
         17     246.62970
         442    257.56480
         77     260.90850
         193    265.03540
         94     275.45810
         318    292.71910
         400    296.17910
         233    296.43730
         11     300.17590
         229    305.10450
         187    316.45470
         267    318.25940
         126    324.65520
         36     339.84690
         180    339.86330
         410    341.97450
         197    342.89370
         433    343.39920
         57     345.79410
                  ...
```

```
425     4015.00000
424     4015.00000
86      4015.00000
106     4015.00000
85      4015.00000
81      4015.00000
112     4015.00000
80      4015.00000
79      4015.00000
420     4015.00000
115     4015.00000
117     4015.00000
91      4015.00000
122     4015.00000
128     4015.00000
451     4015.00000
132     4015.00000
151     4015.00000
454     4015.00000
395     4015.00000
456     4015.00000
457     4015.00000
65      4015.00000
176     4015.00000
383     4015.00000
382     4015.00000
177     4015.00000
181     4015.00000
412     4015.00000
167     4015.00000
Name: part_time, Length: 499, dtype: float64
```

In [18]:
```
obs_df.loc[idx,pst.nnz_obs_names]
```

Out[18]:
```
fo_39_19791230        12272.000000
hds_00_002_009_000       35.513611
hds_00_002_015_000       34.895115
hds_00_003_008_000       35.638062
hds_00_009_001_000       37.328785
hds_00_013_010_000       34.657013
hds_00_015_016_000       34.620155
hds_00_021_010_000       35.170753
hds_00_022_015_000       34.609005
hds_00_024_004_000       35.984737
hds_00_026_006_000       35.450657
hds_00_029_015_000       34.458454
hds_00_033_007_000       34.964649
hds_00_034_010_000       34.124809
Name: 163, dtype: float64
```

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

```
In [19]: obs_df.loc[idx,fnames]
```

```
Out[19]: fa_hw_19791230       -226.815090
         fa_hw_19801229         28.774520
         fa_tw_19791230       -436.949268
         fa_tw_19801229       -253.593190
         hds_00_013_002_000     38.293423
         hds_00_013_002_001     37.771458
         part_time             440.322900
         part_status             2.000000
         Name: 163, dtype: float64
```

1.1.9 Weights!!!

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 [20]: 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"] = 5.0
         obs.loc[obs.obgnme=="calflux","weight"] = 0.01
         obs.loc[pst.nnz_obs_names,"weight"]
```

```
Out[20]: obsnme
         fo_39_19791230        0.01
         hds_00_002_009_000    5.00
         hds_00_002_015_000    5.00
         hds_00_003_008_000    5.00
         hds_00_009_001_000    5.00
         hds_00_013_010_000    5.00
         hds_00_015_016_000    5.00
         hds_00_021_010_000    5.00
         hds_00_022_015_000    5.00
         hds_00_024_004_000    5.00
         hds_00_026_006_000    5.00
         hds_00_029_015_000    5.00
         hds_00_033_007_000    5.00
         hds_00_034_010_000    5.00
         Name: weight, dtype: float64
```

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 [21]: 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[21]: obsnme
fo_39_19791230        176.405235
hds_00_002_009_000      0.080031
hds_00_002_015_000      0.195748
hds_00_003_008_000      0.448179
hds_00_009_001_000      0.373512
hds_00_013_010_000     -0.195456
hds_00_015_016_000      0.190018
hds_00_021_010_000     -0.030271
hds_00_022_015_000     -0.020644
hds_00_024_004_000      0.082120
hds_00_026_006_000      0.028809
hds_00_029_015_000      0.290855
hds_00_033_007_000      0.152208
hds_00_034_010_000      0.024335
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 [22]: pst.write(os.path.join(t_d,"freyberg.pst"))
         pyemu.os_utils.run("pestpp-ies freyberg.pst",cwd=t_d)
```

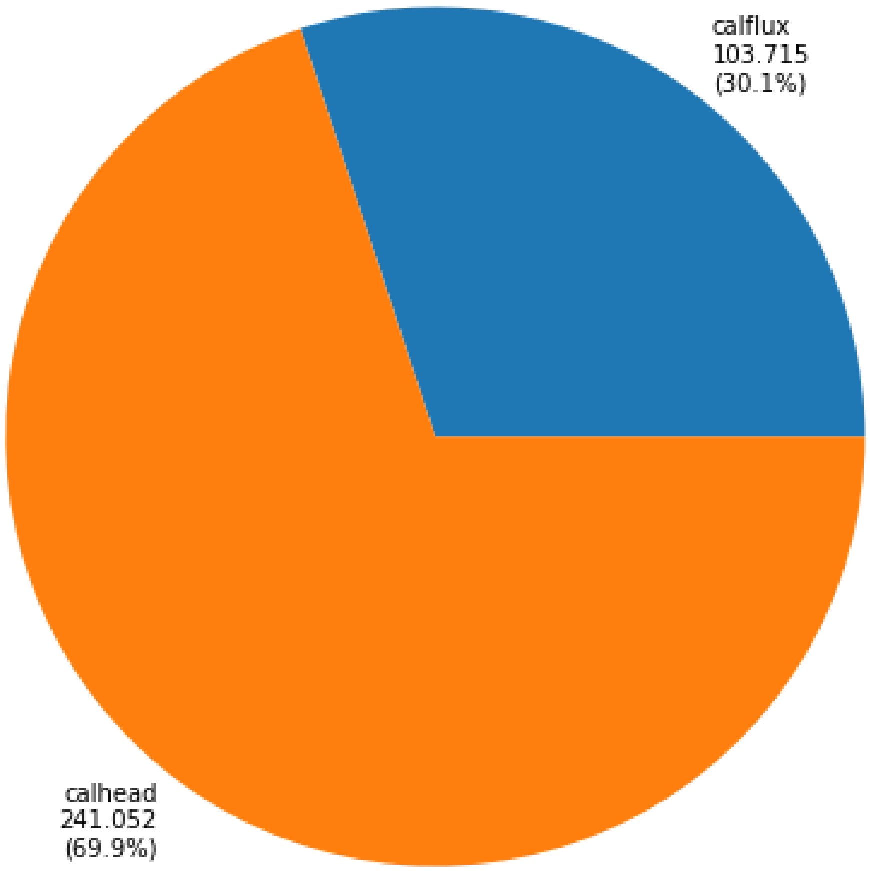
```
noptmax:0, npar_adj:14819, nnz_obs:14
```

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

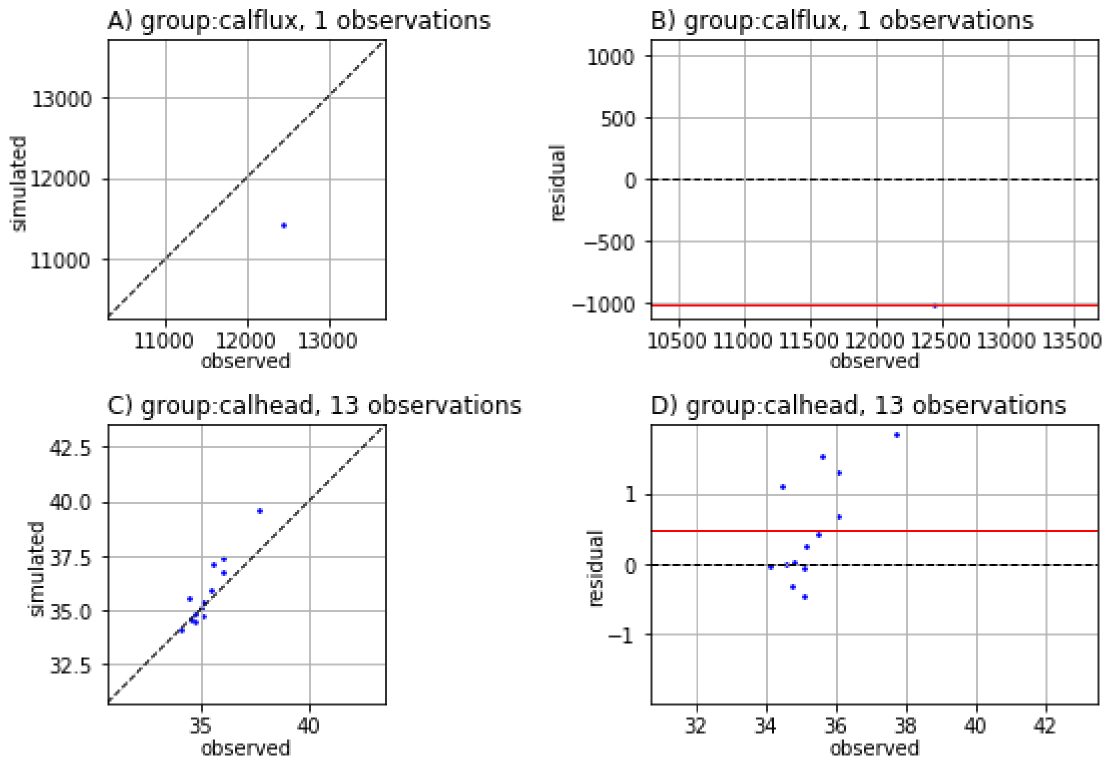
```
In [23]: pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
         print(pst.phi)
         plt.figure()
         pst.plot(kind='phi_pie');
         print('Here are the non-zero weighted observation names')

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

```
344.76729653776556
Here are the non-zero weighted observation names
```

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



<Figure size 576x756 with 0 Axes>


A) group:calflux, 1 observations
simulated
observed
B) group:calflux, 1 observations
residual
observed
C) group:calhead, 13 observations
simulated
observed
D) group:calhead, 13 observations
residual
observed

1.1.10 run the "truth" model once and inspect...

```
In [24]: 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.

```
In [25]: 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,:]
```

```
17.528847239522047
```

```
Out[25]:                                  name    group      measured      modelled  \
         name
         fo_39_19791230         fo_39_19791230  calflux  12448.405235  12272.000000
         hds_00_002_009_000  hds_00_002_009_000  calhead     35.593642     35.513611
         hds_00_002_015_000  hds_00_002_015_000  calhead     35.090862     34.895115
         hds_00_003_008_000  hds_00_003_008_000  calhead     36.086240     35.638062
         hds_00_009_001_000  hds_00_009_001_000  calhead     37.702297     37.328785
         hds_00_013_010_000  hds_00_013_010_000  calhead     34.461557     34.657013
         hds_00_015_016_000  hds_00_015_016_000  calhead     34.810173     34.620155
         hds_00_021_010_000  hds_00_021_010_000  calhead     35.140482     35.170753
         hds_00_022_015_000  hds_00_022_015_000  calhead     34.588361     34.609005
         hds_00_024_004_000  hds_00_024_004_000  calhead     36.066857     35.984737
         hds_00_026_006_000  hds_00_026_006_000  calhead     35.479466     35.450657
         hds_00_029_015_000  hds_00_029_015_000  calhead     34.749309     34.458454
         hds_00_033_007_000  hds_00_033_007_000  calhead     35.116857     34.964649
         hds_00_034_010_000  hds_00_034_010_000  calhead     34.149144     34.124809

                               residual  weight
         name
         fo_39_19791230      176.405235    0.01
         hds_00_002_009_000    0.080031    5.00
         hds_00_002_015_000    0.195748    5.00
         hds_00_003_008_000    0.448179    5.00
         hds_00_009_001_000    0.373512    5.00
         hds_00_013_010_000   -0.195456    5.00
         hds_00_015_016_000    0.190018    5.00
         hds_00_021_010_000   -0.030271    5.00
         hds_00_022_015_000   -0.020644    5.00
         hds_00_024_004_000    0.082120    5.00
         hds_00_026_006_000    0.028809    5.00
```

```
hds_00_029_015_000    0.290855    5.00
hds_00_033_007_000    0.152208    5.00
hds_00_034_010_000    0.024335    5.00
```

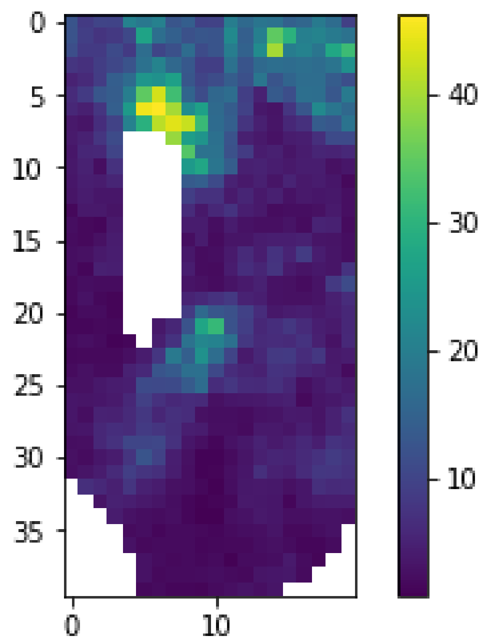
The residual should be exactly the noise values from above. Lets load the model (that was just run using the true pars) and check some things

```
In [26]: m = flopy.modflow.Modflow.load("freyberg.nam",model_ws=m_d)
```

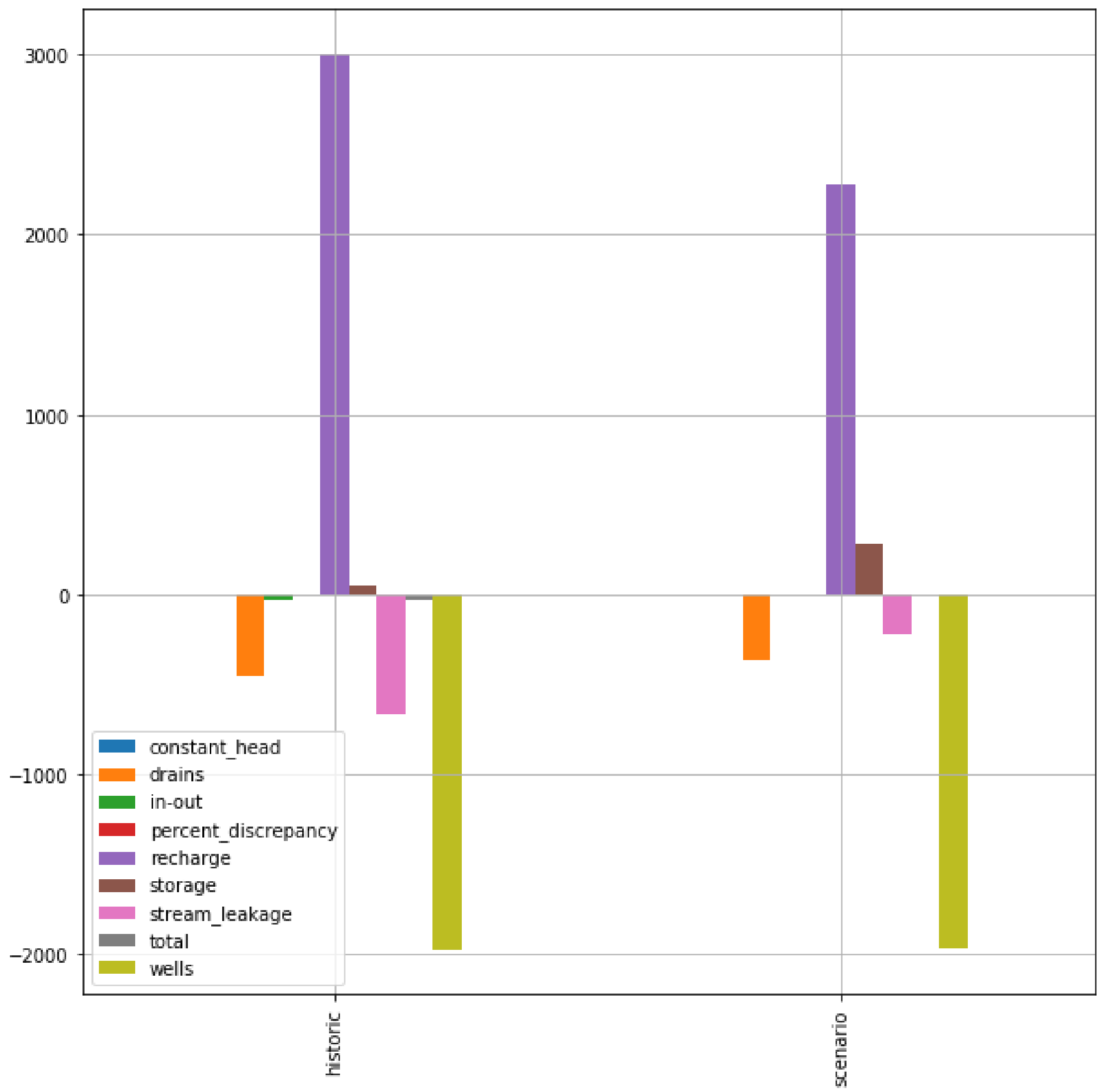
```
In [27]: a = m.upw.hk[2].array
         #a = m.rch.rech[0].array
         a = np.ma.masked_where(m.bas6.ibound[0].array==0,a)
         print(a.min(),a.max())
         c = plt.imshow(a)
         plt.colorbar()
```

```
0.7852934 46.1939
```

```
Out[27]: <matplotlib.colorbar.Colorbar at 0x18190832e8>
```



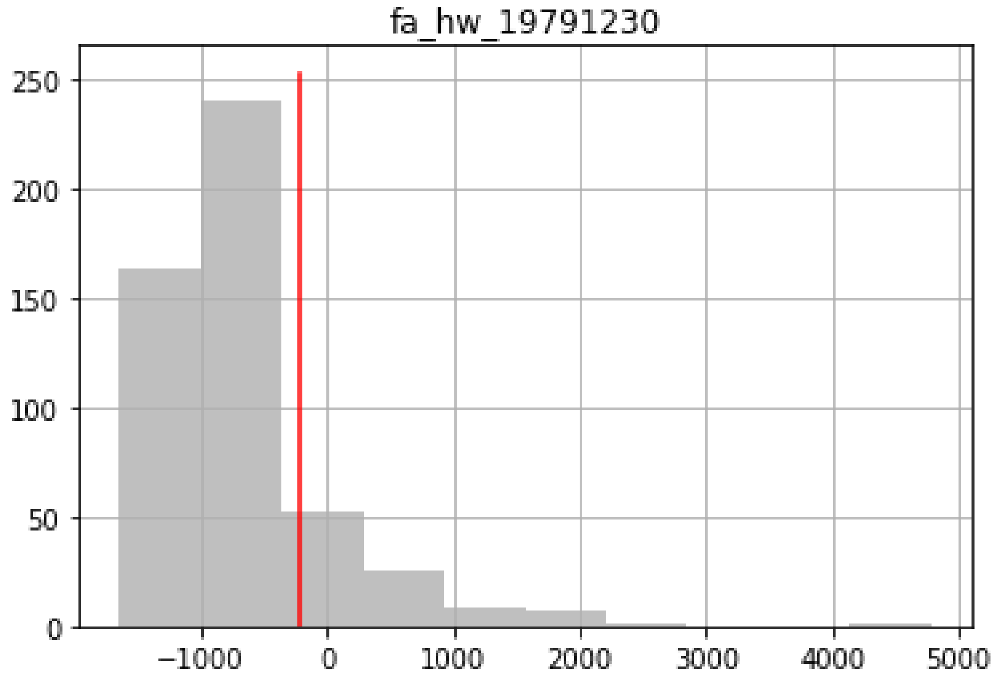

```
In [28]: lst = flopy.utils.MfListBudget(os.path.join(m_d,"freyberg.list"))
         df = lst.get_dataframes(diff=True)[0]
         ax = df.plot(kind="bar",figsize=(10,10), grid=True)
         a = ax.set_xticklabels(["historic","scenario"],rotation=90)
```

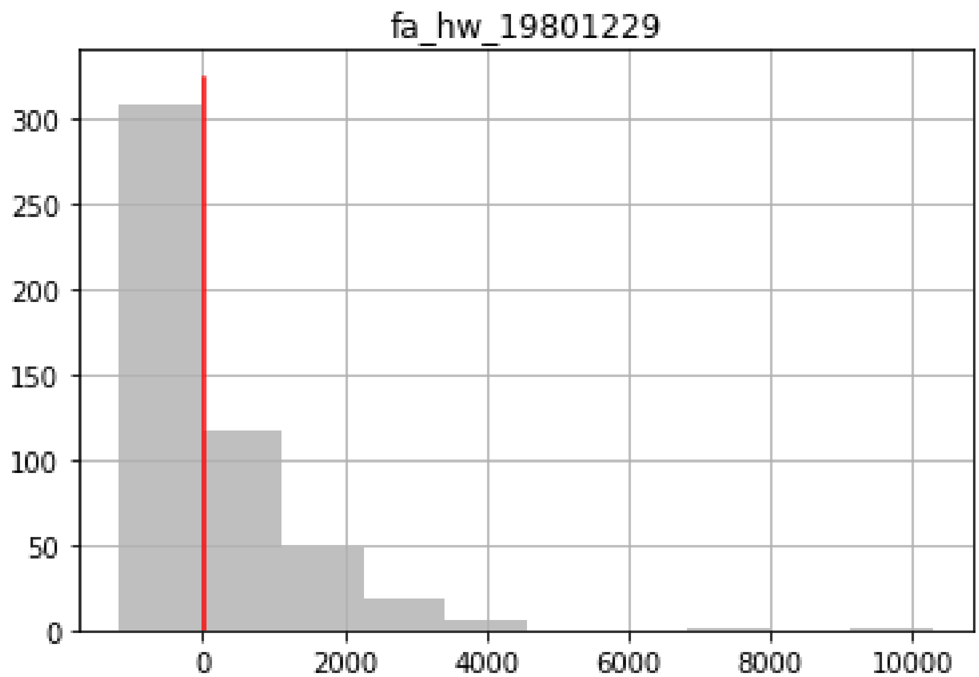


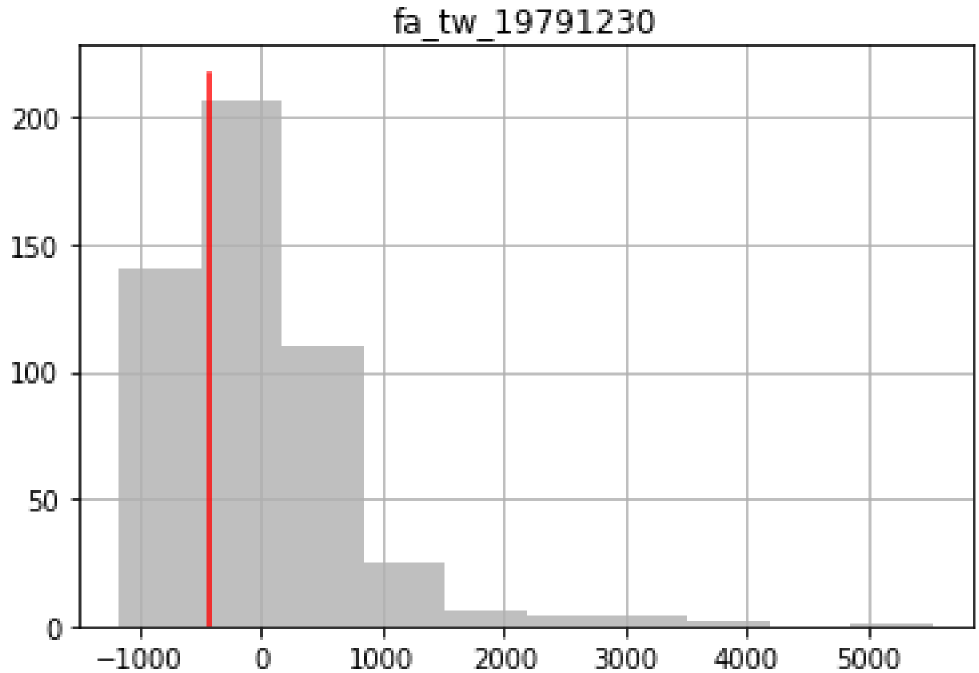
1.1.11 see how our existing observation ensemble compares to the truth

forecasts:

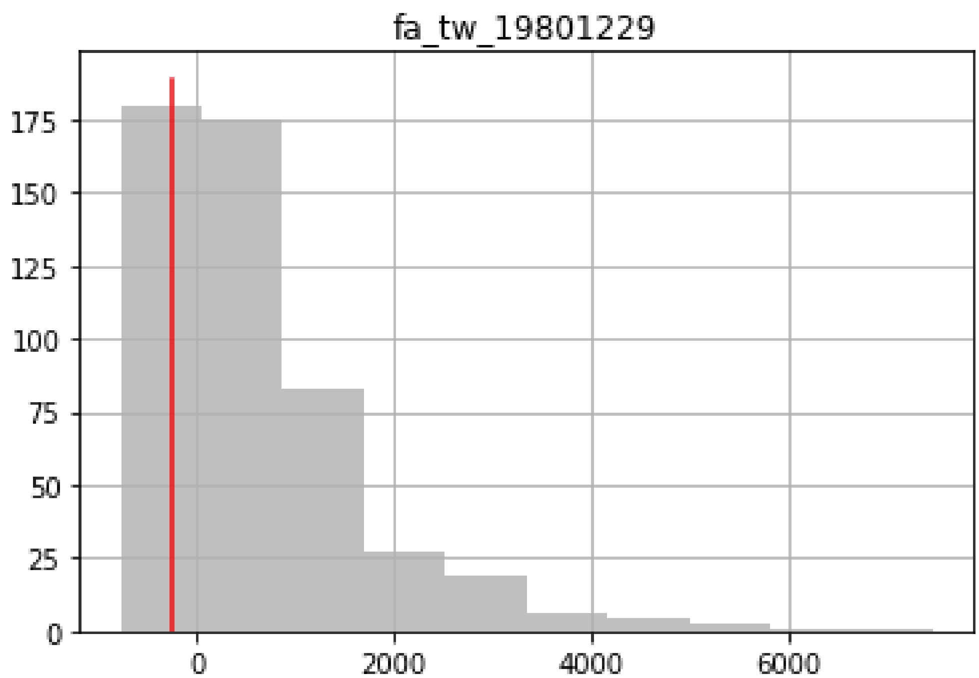
```
In [29]: obs = pst.observation_data
         plt.figure()
         for forecast in fnames:
             ax = plt.subplot(111)
             obs_df.loc[:,forecast].hist(ax=ax,color="0.5",alpha=0.5)
             ax.plot([obs.loc[forecast,"obsval"],obs.loc[forecast,"obsval"]],ax.get_ylim(),"r"
             ax.set_title(forecast)
             plt.show()
```


fa_hw_19791230
250
200
150
100
50
0
−1000
0
1000
2000
3000
4000
5000

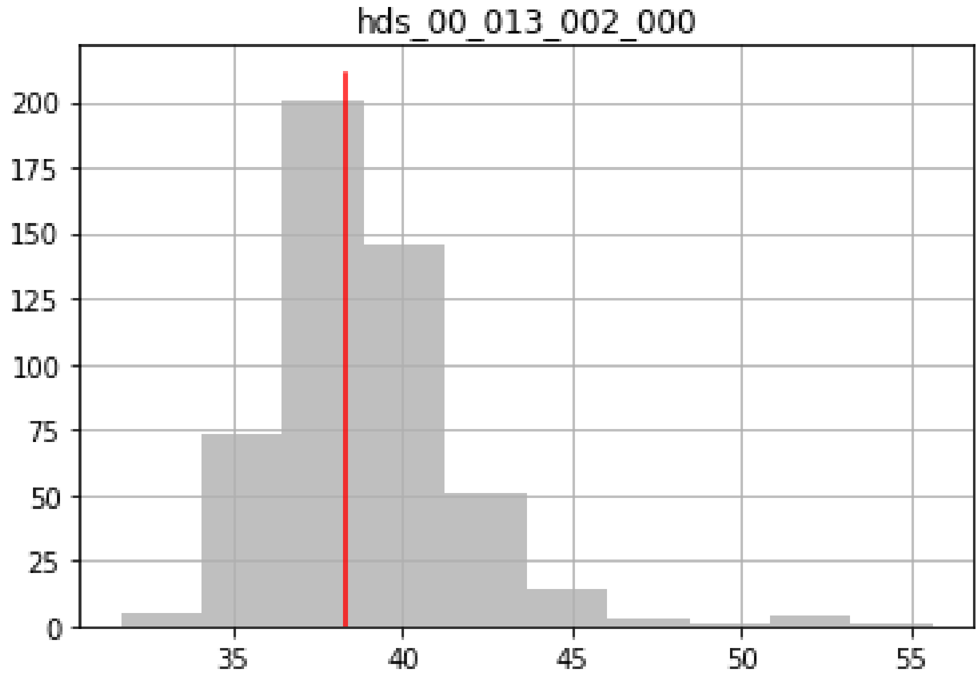

fa_hw_19801229
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000

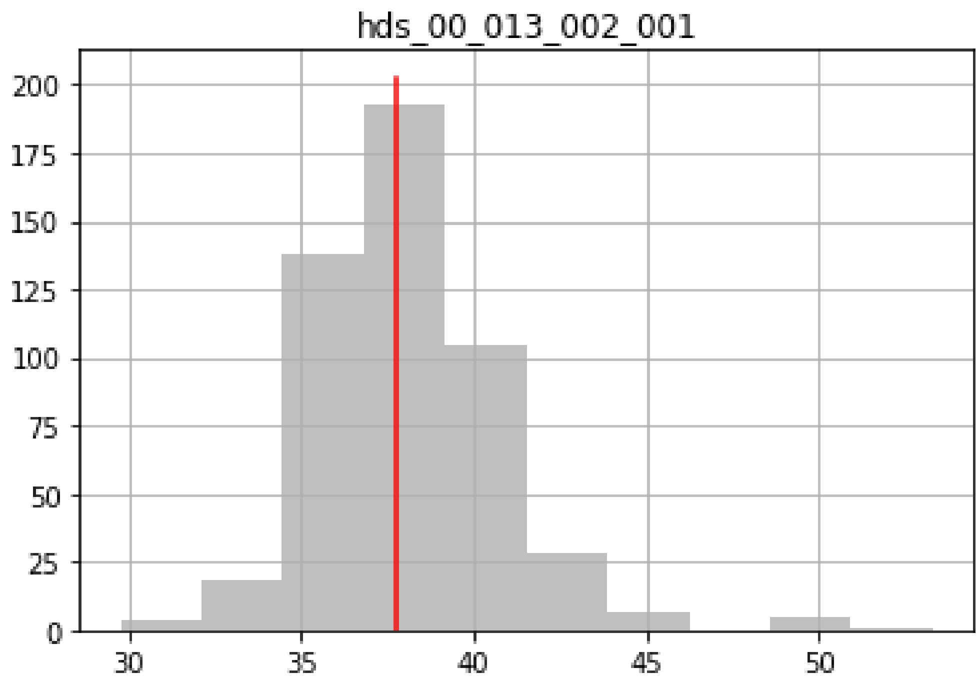


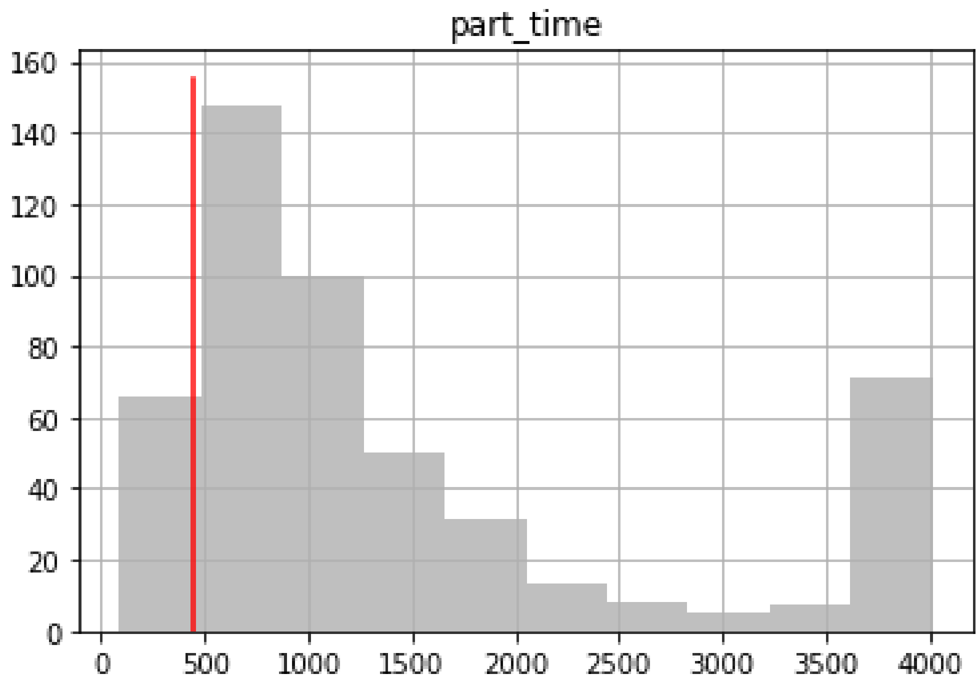
fa_tw_19791230
200
150
100
50
0
−1000
0
1000
2000
3000
4000
5000

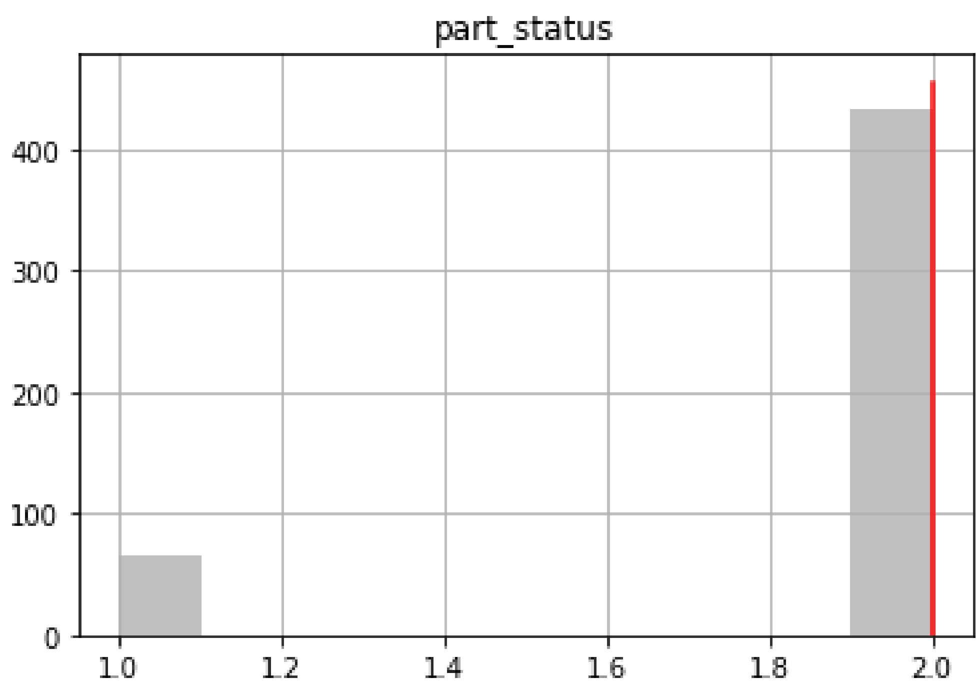


fa_tw_19801229
175
150
125
100
75
50
25
0
0
2000
4000
6000


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

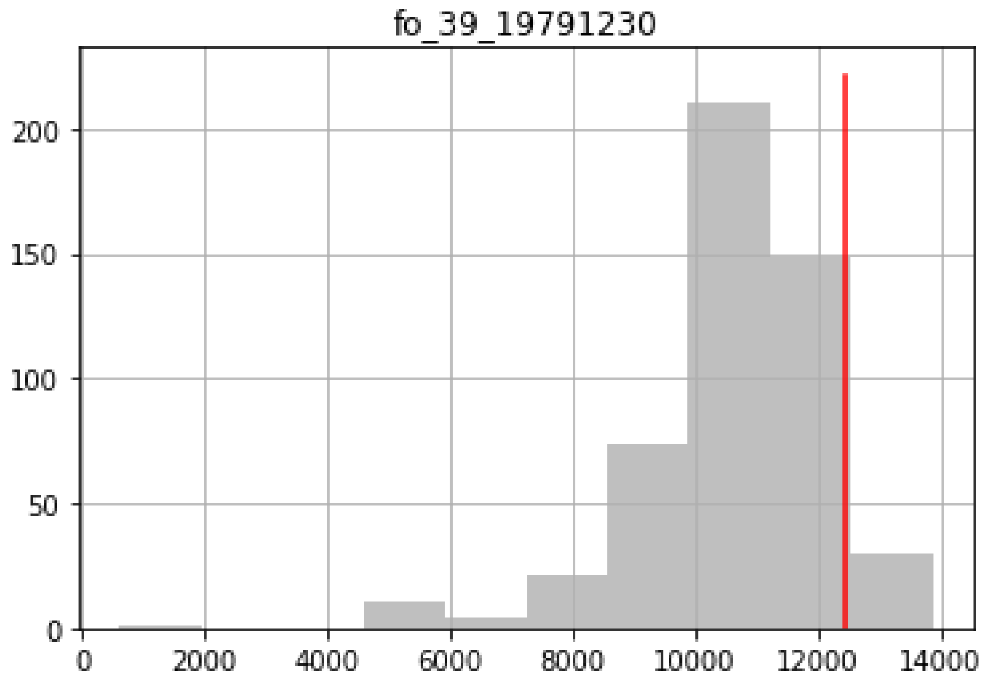

hds_00_013_002_001
0
25
50
75
100
125
150
175
200
30
35
40
45
50

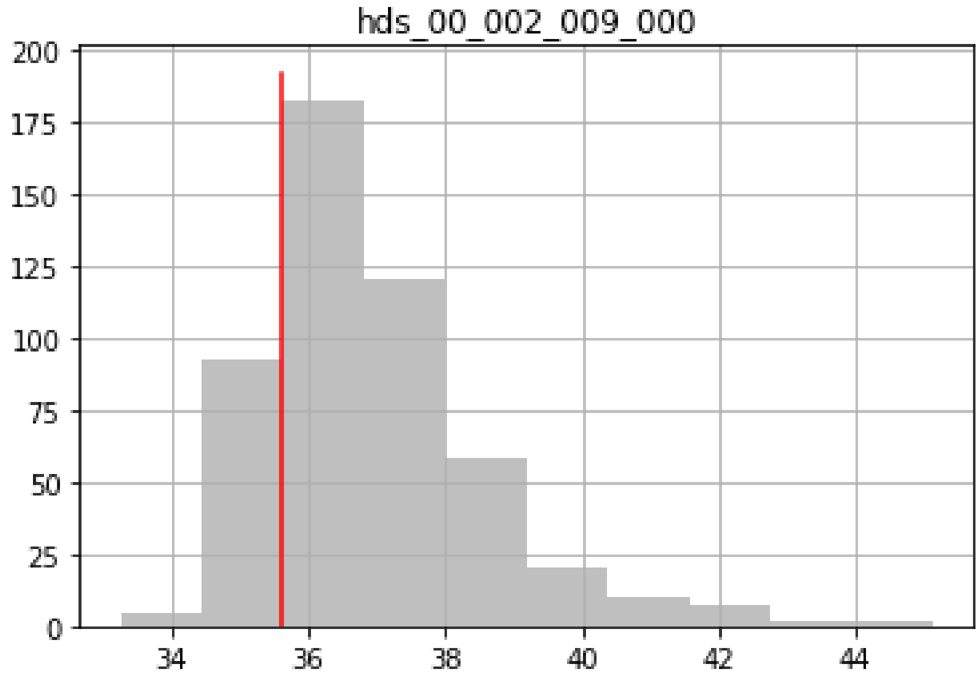


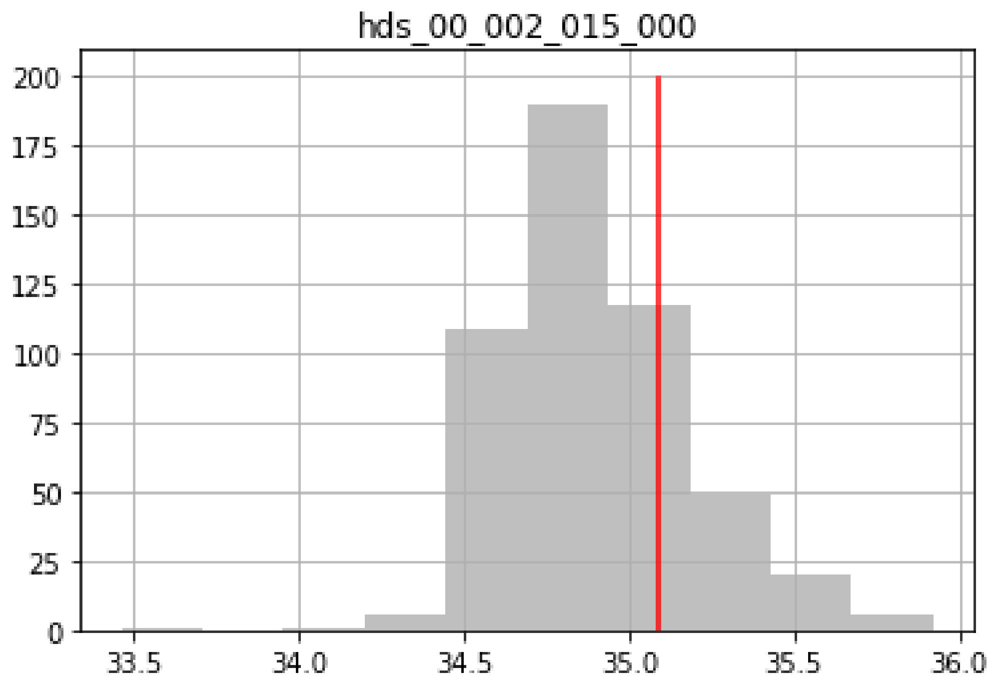


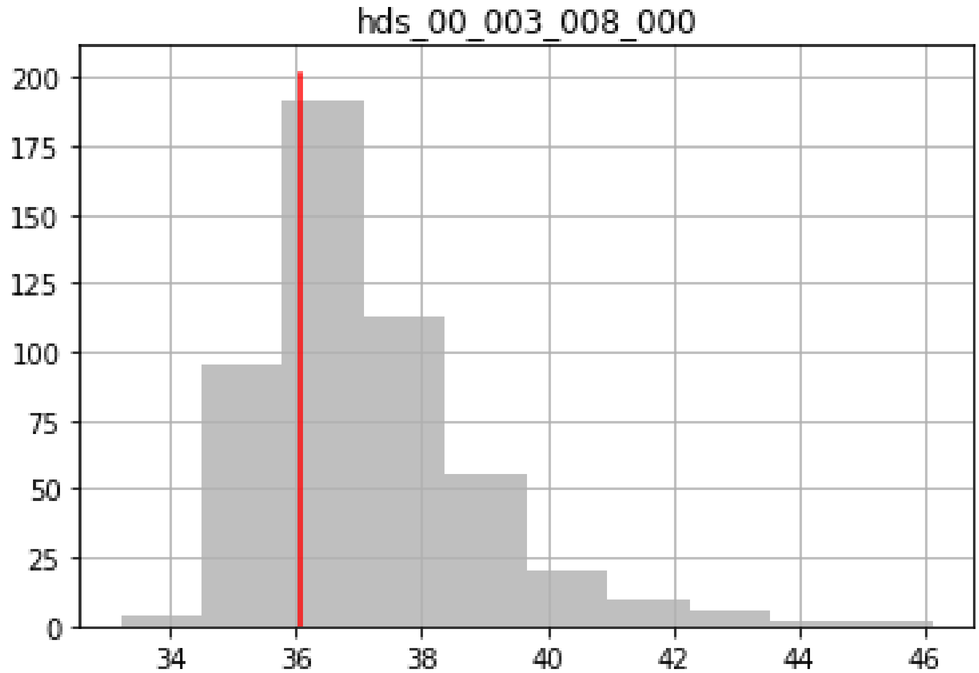
observations:

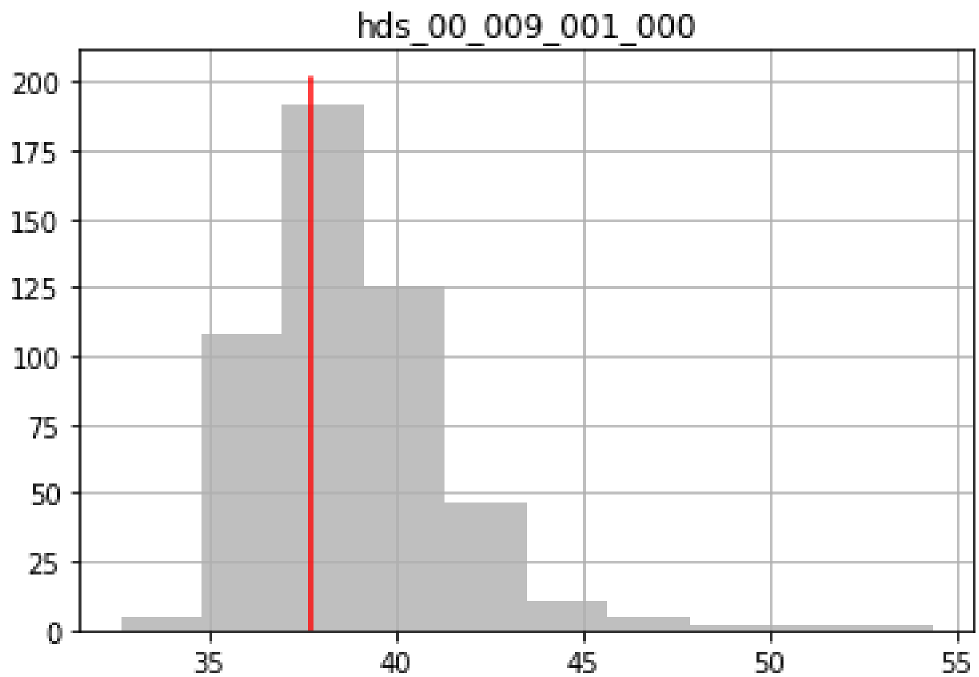
In [30]:
```
for oname in pst.nnz_obs_names:
    ax = plt.subplot(111)
    obs_df.loc[:,oname].hist(ax=ax,color="0.5",alpha=0.5)
    ax.plot([obs.loc[oname,"obsval"],obs.loc[oname,"obsval"]],ax.get_ylim(),"r")
    ax.set_title(oname)
    plt.show()
```

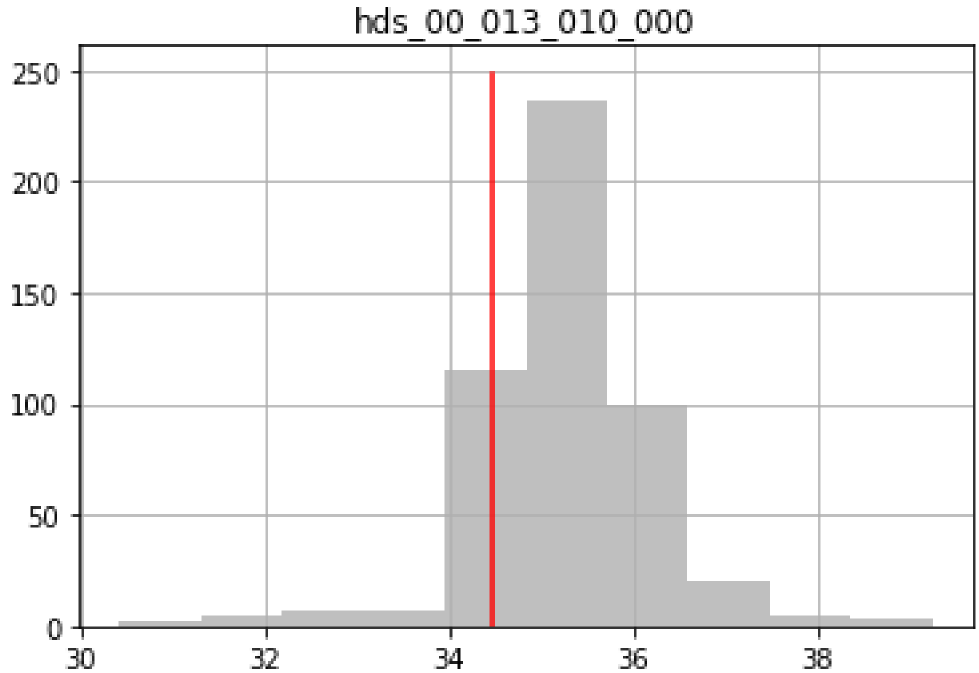


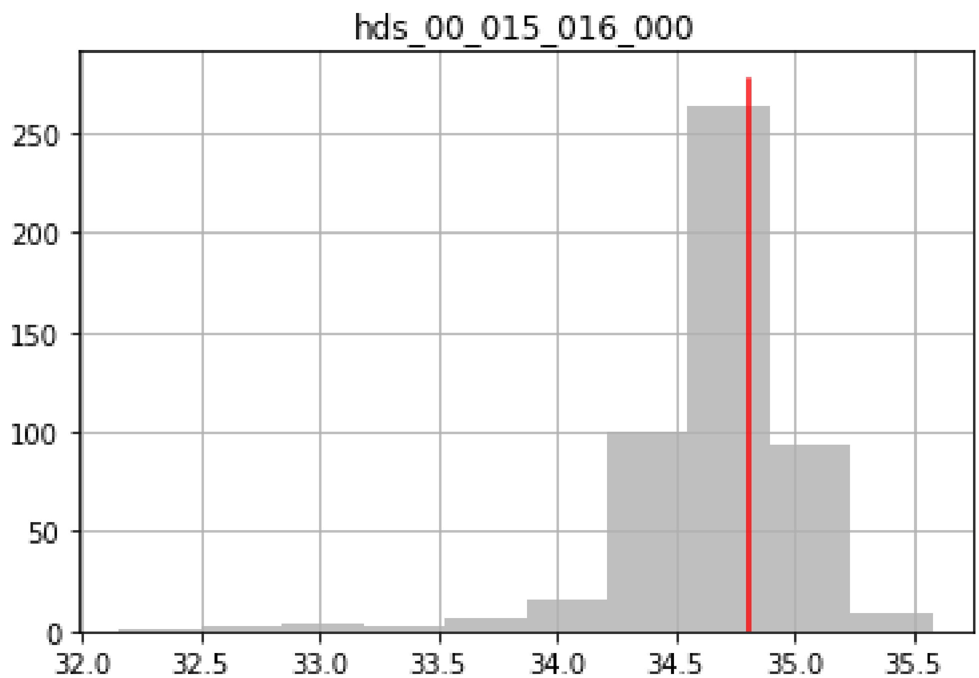

hds_00_002_009_000
200
175
150
125
100
75
50
25
0
34
36
38
40
42
44

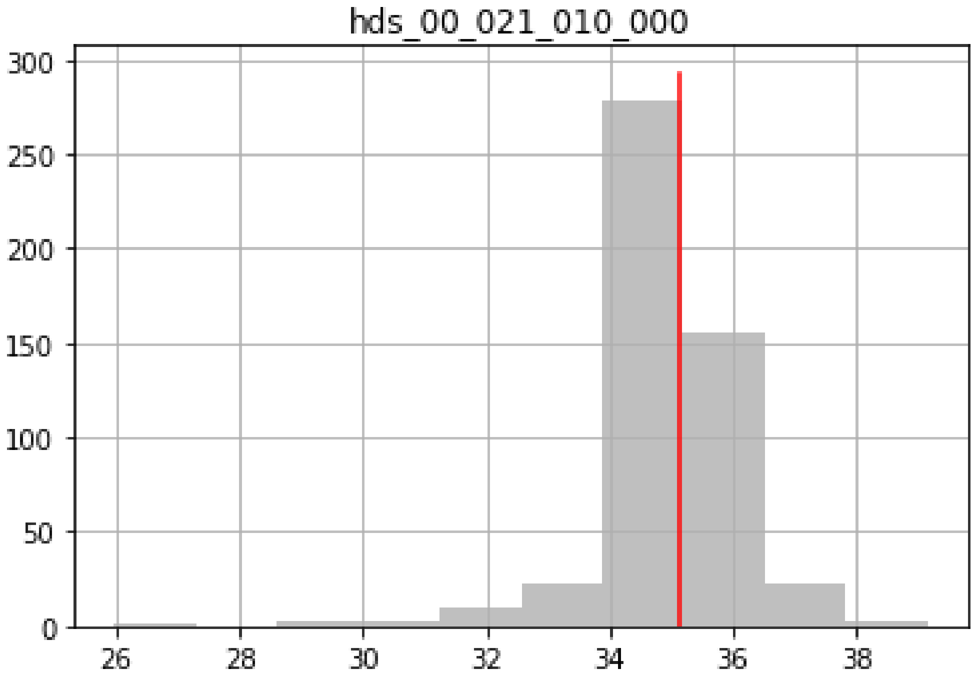

hds_00_002_015_000
200
175
150
125
100
75
50
25
0
33.5
34.0
34.5
35.0
35.5
36.0

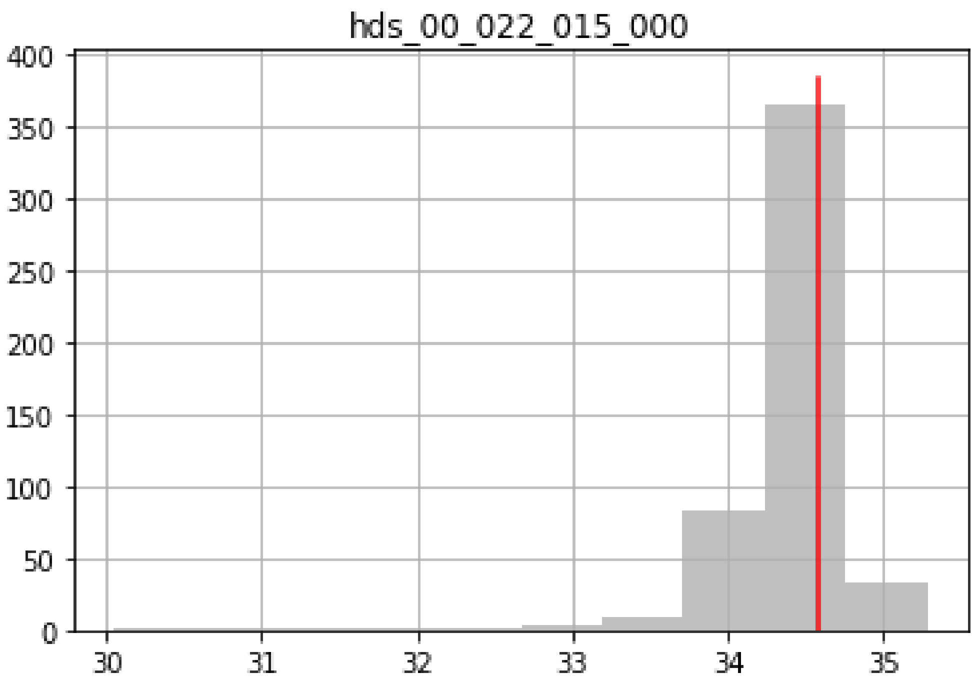

hds_00_003_008_000
200
175
150
125
100
75
50
25
0
34
36
38
40
42
44
46

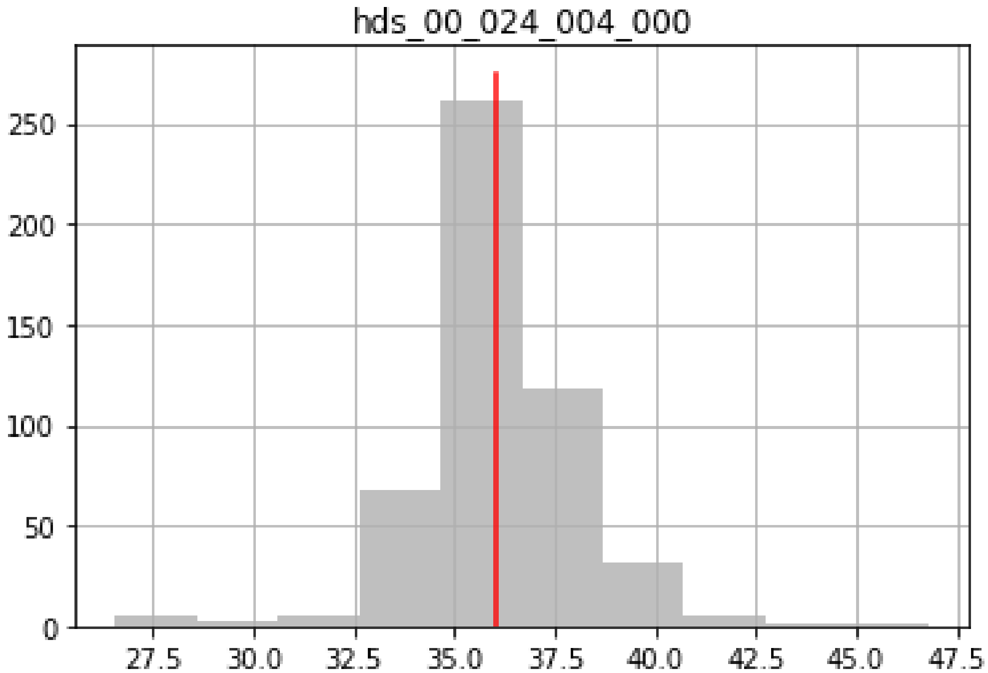

hds_00_009_001_000
200
175
150
125
100
75
50
25
0
35
40
45
50
55


hds_00_013_010_000
250
200
150
100
50
0
30
32
34
36
38

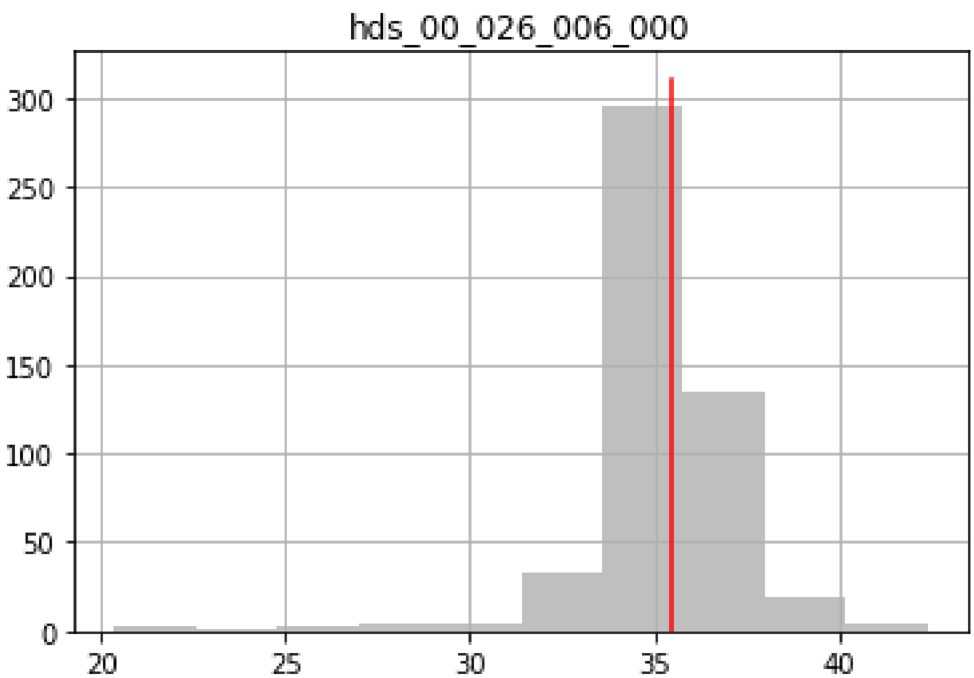

hds_00_015_016_000
250
200
150
100
50
0
32.0
32.5
33.0
33.5
34.0
34.5
35.0
35.5


hds_00_021_010_000
300
250
200
150
100
50
0
26
28
30
32
34
36
38

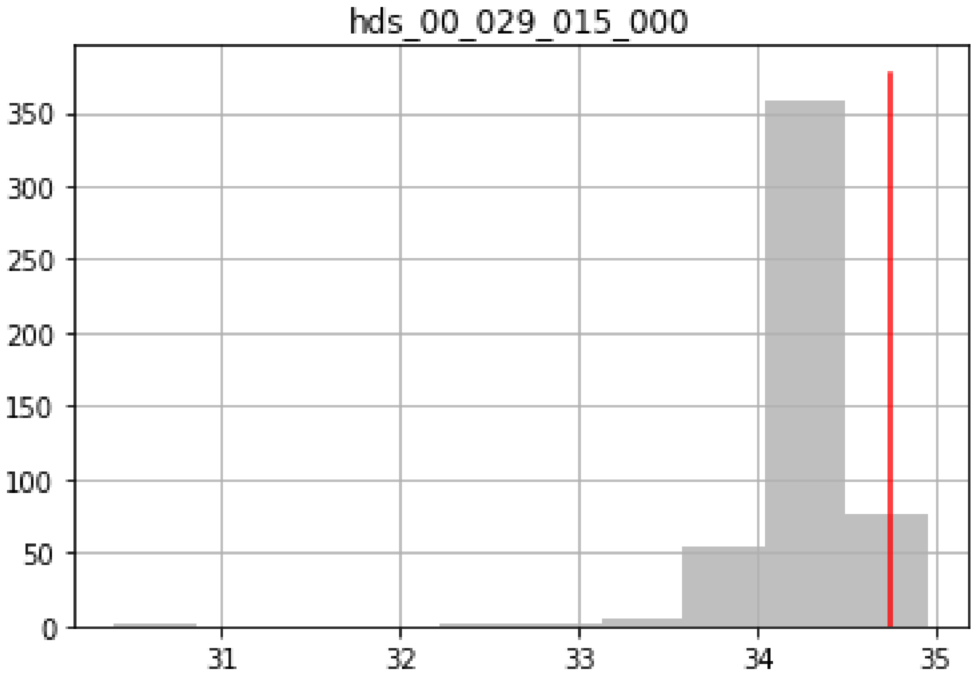

hds_00_022_015_000
400
350
300
250
200
150
100
50
0
30
31
32
33
34
35

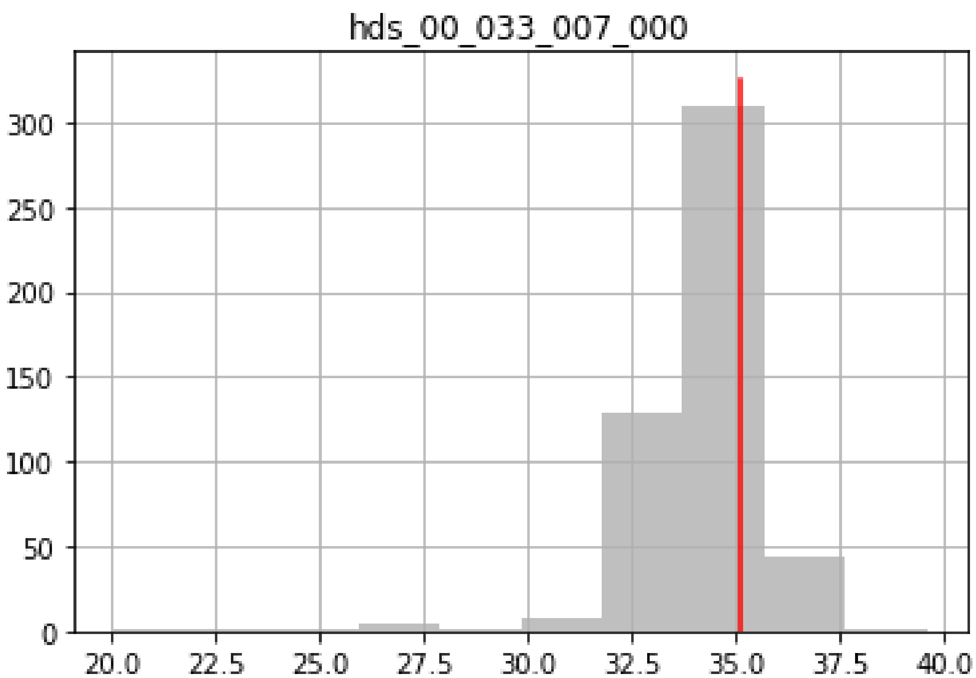


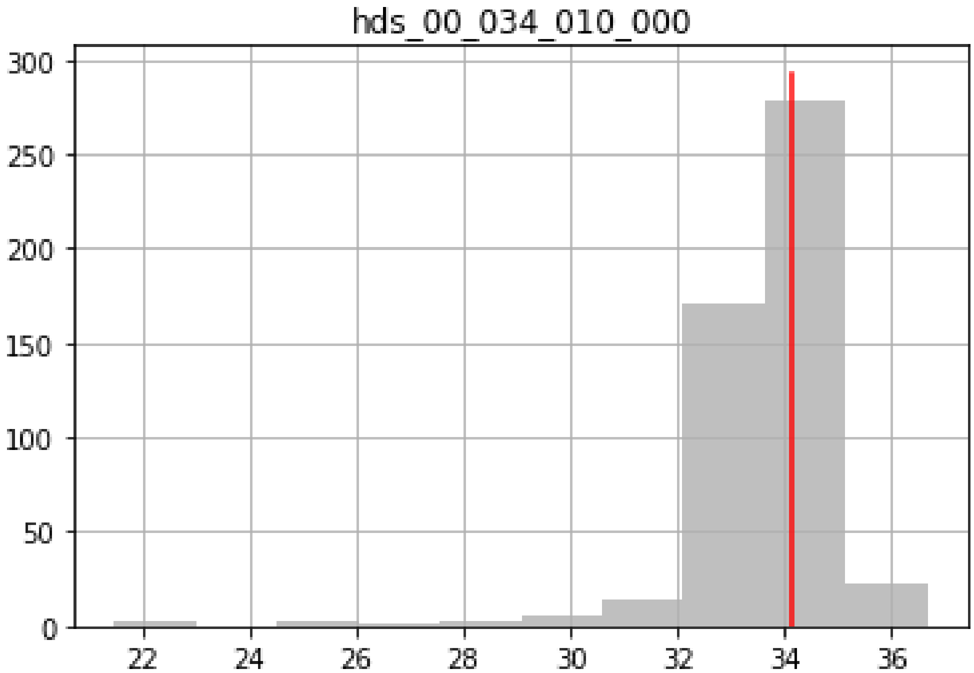
hds_00_024_004_000
0
50
100
150
200
250
27.5
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5



hds_00_026_006_000
0
50
100
150
200
250
300
20
25
30
35
40


hds_00_029_015_000
350
300
250
200
150
100
50
0
31
32
33
34
35


hds_00_033_007_000
300
250
200
150
100
50
0
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0


hds_00_034_010_000
300
250
200
150
100
50
0
22
24
26
28
30
32
34
36