pestpp-glm

May 31, 2019

1 PESTPP-GLM

In this notebook, we will run PESTPP-GLM in standard parameter estimation mode and regularization mode. In both cases, we will use the baked-in bayes-linear posterior monte carlo analysis to get posterior forecast PDFs. We will use the prior monte carlo outputs as the prior forecast PDF.

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
```

1.1 SUPER IMPORTANT: SET HOW MANY PARALLEL WORKERS TO USE

In [2]:
```
num_workers = 20
```

In [3]:
```
t_d = "template"
m_d = "master_glm"
```

In [4]:
```
pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
pst.write_par_summary_table(filename="none")
```

Out[4]:
```
                    type transform  count      initial value  \
cn_strt8        cn_strt8       log      1                  0
gr_vka3          gr_vka3       log    705                  0
cn_hk7             cn_hk7       log      1                  0
pp_ss1             pp_ss1       log     32                  0
gr_hk3             gr_hk3       log    705                  0
cn_strt6        cn_strt6       log      1                  0
cn_vka7          cn_vka7       log      1                  0
cn_vka6          cn_vka6       log      1                  0
```

pp_ss0	pp_ss0	log	32	0
gr_rech2	gr_rech2	log	705	0
gr_prsity5	gr_prsity5	log	705	0
gr_vka4	gr_vka4	log	705	0
gr_prsity3	gr_prsity3	log	705	0
gr_sy4	gr_sy4	log	705	0
gr_strt3	gr_strt3	log	705	0
pp_rech1	pp_rech1	log	32	0
drncond_k00	drncond_k00	log	10	0
cn_rech5	cn_rech5	log	1	-0.39794
gr_ss3	gr_ss3	log	705	0
gr_strt5	gr_strt5	log	705	0
pp_vka0	pp_vka0	log	32	0
gr_hk4	gr_hk4	log	705	0
gr_prsity4	gr_prsity4	log	705	0
pp_vka2	pp_vka2	log	32	0
gr_ss5	gr_ss5	log	705	0
gr_sy5	gr_sy5	log	705	0
pp_ss2	pp_ss2	log	32	0
pp_sy0	pp_sy0	log	32	0
gr_vka5	gr_vka5	log	705	0
cn_ss6	cn_ss6	log	1	0
...	...	...	...	...
cn_rech4	cn_rech4	log	1	0
cn_vka8	cn_vka8	log	1	0
cn_sy8	cn_sy8	log	1	0
pp_strt1	pp_strt1	log	32	0
cn_prsity7	cn_prsity7	log	1	0
pp_prsity1	pp_prsity1	log	32	0
flow	flow	log	1	0
cn_sy7	cn_sy7	log	1	0
gr_strt4	gr_strt4	log	705	0
cn_sy6	cn_sy6	log	1	0
gr_hk5	gr_hk5	log	705	0
gr_ss4	gr_ss4	log	705	0
gr_rech3	gr_rech3	log	705	0
pp_prsity0	pp_prsity0	log	32	0
welflux_k02	welflux_k02	log	6	0
pp_strt2	pp_strt2	log	32	0
cn_hk8	cn_hk8	log	1	0
strk	strk	log	40	0
cn_ss8	cn_ss8	log	1	0
pp_hk1	pp_hk1	log	32	0
pp_rech0	pp_rech0	log	32	0
pp_prsity2	pp_prsity2	log	32	0
cn_strt7	cn_strt7	log	1	0
pp_vka1	pp_vka1	log	32	0
cn_prsity6	cn_prsity6	log	1	0

cn_ss7	cn_ss7	log	1	0
welflux	welflux	log	2	0 to 0.176091
cn_hk6	cn_hk6	log	1	0
pp_hk0	pp_hk0	log	32	0
pp_hk2	pp_hk2	log	32	0

	upper bound	lower bound	standard deviation
cn_strt8	0.0211893	-0.0222764	0.0108664
gr_vka3	1	-1	0.5
cn_hk7	1	-1	0.5
pp_ss1	1	-1	0.5
gr_hk3	1	-1	0.5
cn_strt6	0.0211893	-0.0222764	0.0108664
cn_vka7	1	-1	0.5
cn_vka6	1	-1	0.5
pp_ss0	1	-1	0.5
gr_rech2	0.0413927	-0.0457575	0.0217875
gr_prsity5	0.176091	-0.30103	0.11928
gr_vka4	1	-1	0.5
gr_prsity3	0.176091	-0.30103	0.11928
gr_sy4	0.243038	-0.60206	0.211275
gr_strt3	0.0211893	-0.0222764	0.0108664
pp_rech1	0.0413927	-0.0457575	0.0217875
drncond_k00	1	-1	0.5
cn_rech5	-0.09691	-1	0.225772
gr_ss3	1	-1	0.5
gr_strt5	0.0211893	-0.0222764	0.0108664
pp_vka0	1	-1	0.5
gr_hk4	1	-1	0.5
gr_prsity4	0.176091	-0.30103	0.11928
pp_vka2	1	-1	0.5
gr_ss5	1	-1	0.5
gr_sy5	0.243038	-0.60206	0.211275
pp_ss2	1	-1	0.5
pp_sy0	0.243038	-0.60206	0.211275
gr_vka5	1	-1	0.5
cn_ss6	1	-1	0.5
...	...	...	...
cn_rech4	0.0791812	-0.09691	0.0440228
cn_vka8	1	-1	0.5
cn_sy8	0.243038	-0.60206	0.211275
pp_strt1	0.0211893	-0.0222764	0.0108664
cn_prsity7	0.176091	-0.30103	0.11928
pp_prsity1	0.176091	-0.30103	0.11928
flow	0.09691	-0.124939	0.0554622
cn_sy7	0.243038	-0.60206	0.211275
gr_strt4	0.0211893	-0.0222764	0.0108664
cn_sy6	0.243038	-0.60206	0.211275

```
gr_hk5                        1             -1             0.5
gr_ss4                        1             -1             0.5
gr_rech3              0.0413927     -0.0457575       0.0217875
pp_prsity0             0.176091       -0.30103         0.11928
welflux_k02                   1             -1             0.5
pp_strt2              0.0211893     -0.0222764       0.0108664
cn_hk8                        1             -1             0.5
strk                          2             -2               1
cn_ss8                        1             -1             0.5
pp_hk1                        1             -1             0.5
pp_rech0              0.0413927     -0.0457575       0.0217875
pp_prsity2             0.176091       -0.30103         0.11928
cn_strt7              0.0211893     -0.0222764       0.0108664
pp_vka1                       1             -1             0.5
cn_prsity6             0.176091       -0.30103         0.11928
cn_ss7                        1             -1             0.5
welflux      0.176091 to 0.30103  -0.30103 to      0  0.0752575 to 0.11928
cn_hk6                        1             -1             0.5
pp_hk0                        1             -1             0.5
pp_hk2                        1             -1             0.5

[65 rows x 7 columns]
```

1.1.1 reduce the number of adjustable parameters

This is the painful part: we cant use 10K+ pars because we cant wait around for that many runs and then the linear algebra of factoring a 10k+ by 10K+ matrix is also difficult. So that means we need to fix a lot a parameters #frownyface

```
In [5]: par = pst.parameter_data
```

```
In [6]: # grid-scale pars
        gr_pars = par.loc[par.pargp.apply(lambda x: "gr" in x),"parnme"]
        par.loc[gr_pars,"partrans"] = "fixed"
        pst.npar_adj
```

```
Out[6]: 719
```

```
In [7]: # these are the sfr conductance parameters - Ive left all 40 adjustable
        # but if you uncomment this, it will tie them into 1 parameter effectively
        # strk_pars = par.loc[par.pargp=="strk","parnme"]
        # p1 = strk_pars.iloc[0]
        # par.loc[strk_pars.iloc[1:],"partrans"] = "tied"
        # par.loc[strk_pars.iloc[1:],"partied"] = p1
        pst.npar_adj
```

```
Out[7]: 719
```

```
In [8]: par.loc[par.pargp.apply(lambda x: "pp" in x),"pargp"].unique()
```

Out[8]: array(['pp_hk0', 'pp_hk1', 'pp_hk2', 'pp_prsity0', 'pp_prsity1',
       'pp_prsity2', 'pp_rech0', 'pp_rech1', 'pp_ss0', 'pp_ss1', 'pp_ss2',
       'pp_strt0', 'pp_strt1', 'pp_strt2', 'pp_sy0', 'pp_sy1', 'pp_sy2',
       'pp_vka0', 'pp_vka1', 'pp_vka2'], dtype=object)

Fix the storage pilot points - we still have layer-scale storage pars adjustable

```
In [9]: #s_pars = par.loc[par.pargp.apply(lambda x: "pp" in x and ("ss" in x or "sy" in x)),"p
        #par.loc[s_pars,"partrans"] = "fixed"
        pst.npar_adj
```

```
Out[9]: 719
```

```
In [10]: adj_par = par.loc[par.partrans=="log",:]
         adj_par.pargp.value_counts().sort_values()
```

```
Out[10]: cn_strt8       1
         cn_sy6         1
         cn_vka6        1
         cn_vka7        1
         cn_hk6         1
         flow           1
         cn_strt6       1
         cn_hk7         1
         cn_rech5       1
         cn_prsity6     1
         cn_ss7         1
         cn_sy7         1
         cn_strt7       1
         cn_ss6         1
         cn_ss8         1
         cn_prsity8     1
         cn_hk8         1
         cn_rech4       1
         cn_vka8        1
         cn_sy8         1
         cn_prsity7     1
         welflux        2
         welflux_k02    6
         drncond_k00   10
         pp_hk1        32
         pp_ss0        32
         pp_vka1       32
         pp_prsity1    32
         pp_rech1      32
         pp_vka0       32
         pp_ss2        32
         pp_sy0        32
         pp_ss1        32
```

```
pp_strt2       32
pp_sy1         32
pp_hk2         32
pp_strt1       32
pp_strt0       32
pp_rech0       32
pp_prsity0     32
pp_hk0         32
pp_prsity2     32
pp_vka2        32
pp_sy2         32
strk           40
Name: pargp, dtype: int64
```

fix the future recharge pilot points, vka in layers 1 and 3 and the initial condition pilot points (we still have layer-scale pars for each of these types)

```
In [11]: fi_grps = ["pp_rech1","pp_vka0","pp_vka2","pp_strt0","pp_strt1","pp_strt2"]
         par.loc[par.pargp.apply(lambda x: x in fi_grps),"partrans"] = "fixed"
         pst.npar_adj
```

```
Out[11]: 527
```

Ok, thats better…so lets run PESTPP-GLM. We will use a single "base parameter" jacobian matrix as the basis for 6 super parameter iterations. Then we will draw 100 realizations from the FOSM posterior parameter covariance matrix and run those 100 realizations to get the psoterior forecast PDFs

```
In [12]: pst.control_data.noptmax = 3
         pst.pestpp_options["n_iter_base"] = -1
         pst.pestpp_options["n_iter_super"] = 3
         pst.pestpp_options["num_reals"] = 50 # this is how many ies uses
         pst.pestpp_options["parcov"] = "prior_cov.jcb"
         pst.write(os.path.join(t_d,"freyberg_pp.pst"))
```

```
noptmax:3, npar_adj:527, nnz_obs:14
```

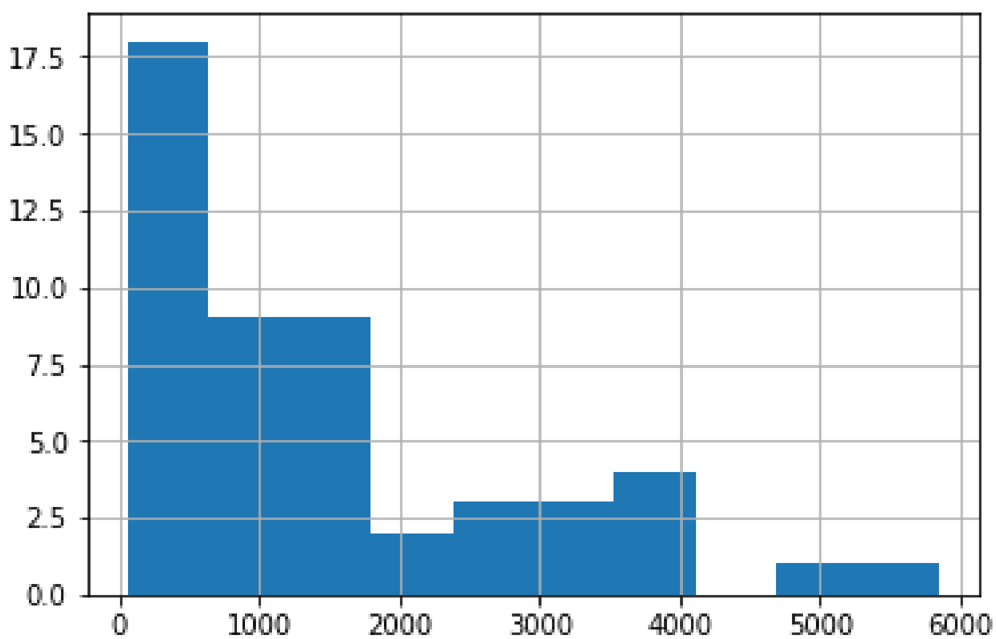
```
In [13]: pyemu.os_utils.start_slaves(t_d,"pestpp-glm","freyberg_pp.pst",num_slaves=num_workers
                                     master_dir=m_d)
```

```
In [14]: df = df=pd.read_csv(os.path.join(m_d,"freyberg_pp.post.obsen.csv"),index_col=0)
         oe = pyemu.ObservationEnsemble.from_dataframe(pst=pst,df=df)
```

```
In [15]: ax = oe.phi_vector.hist()#bins=np.linspace(0,100,20))
         oe.phi_vector.sort_values().iloc[:20]
```

```
Out[15]: real_name
         25     57.504700
```

```
49     59.965027
48     79.076502
22    121.743995
12    129.003257
21    235.929537
10    271.640362
26    303.531649
42    328.530748
29    370.831442
24    473.160536
47    474.512084
33    495.605323
44    505.914044
5     507.399642
1     537.591081
9     608.619145
19    616.414116
46    680.674779
37    805.598204
dtype: float64
```



Here we see the distribution of phi values across the 100 posterior realizations. Should we accept all of these??? The theoretical phi we should accept is number of nonzero obs (14).

To get a "posterior" ensemble, we need to throw out the realizations with large phi - lets just take the 20 best:

```
In [16]: oe_pt = oe.loc[oe.phi_vector.sort_values().index[:20],:] #just take the 20 lowest phi
```

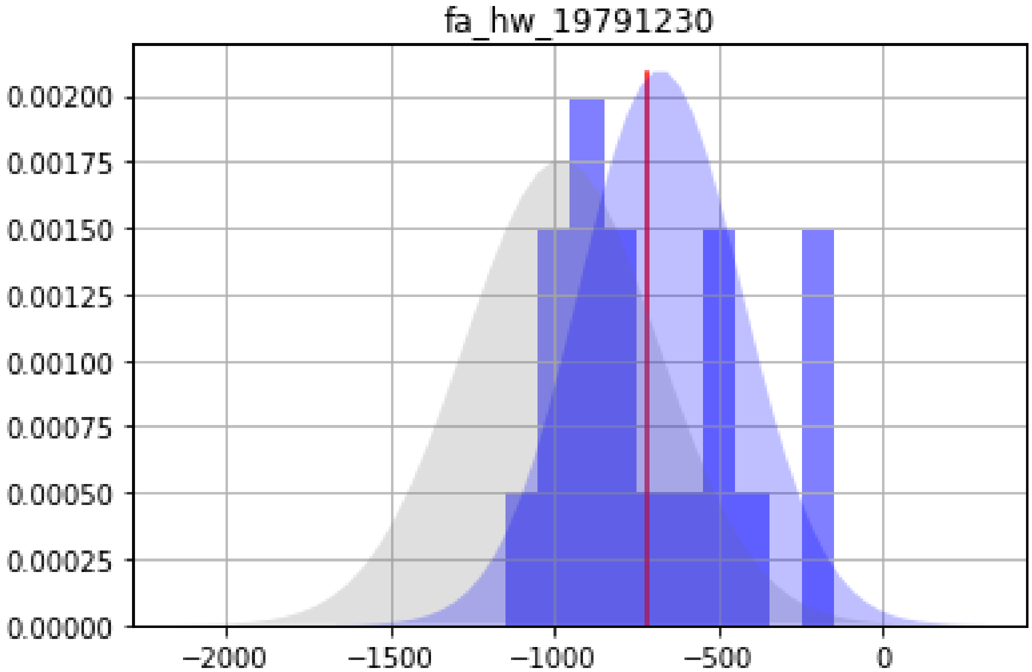
We can also load and plot the FOSM forecast results along side of the ensemble results:

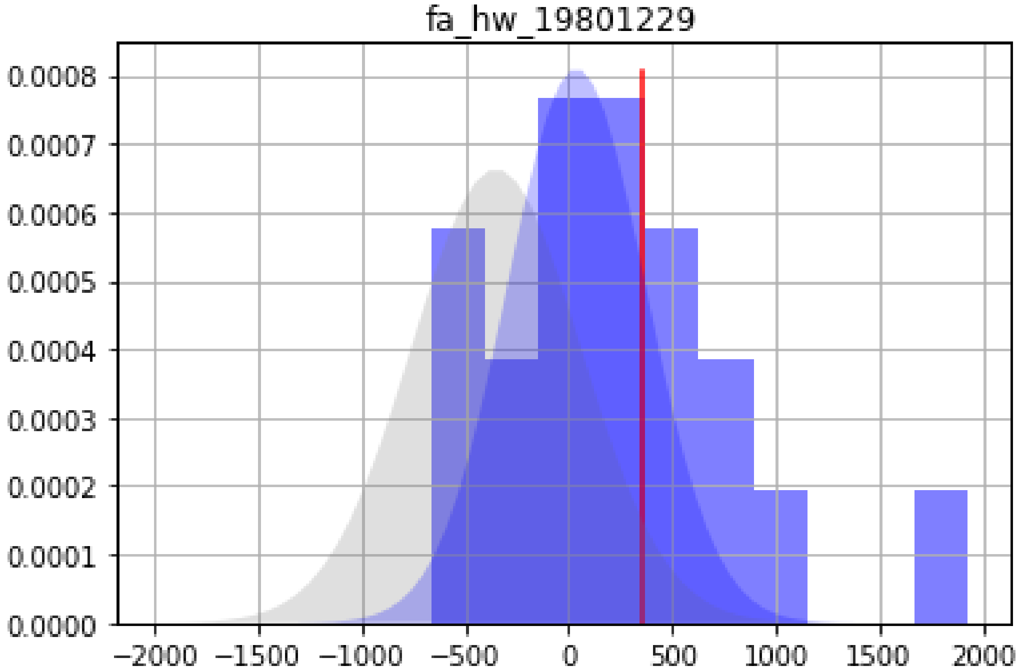
```
In [17]: f_df = pd.read_csv(os.path.join(m_d,"freyberg_pp.pred.usum.csv"),index_col=0)
         f_df.index = f_df.index.map(str.lower)
         f_df
```

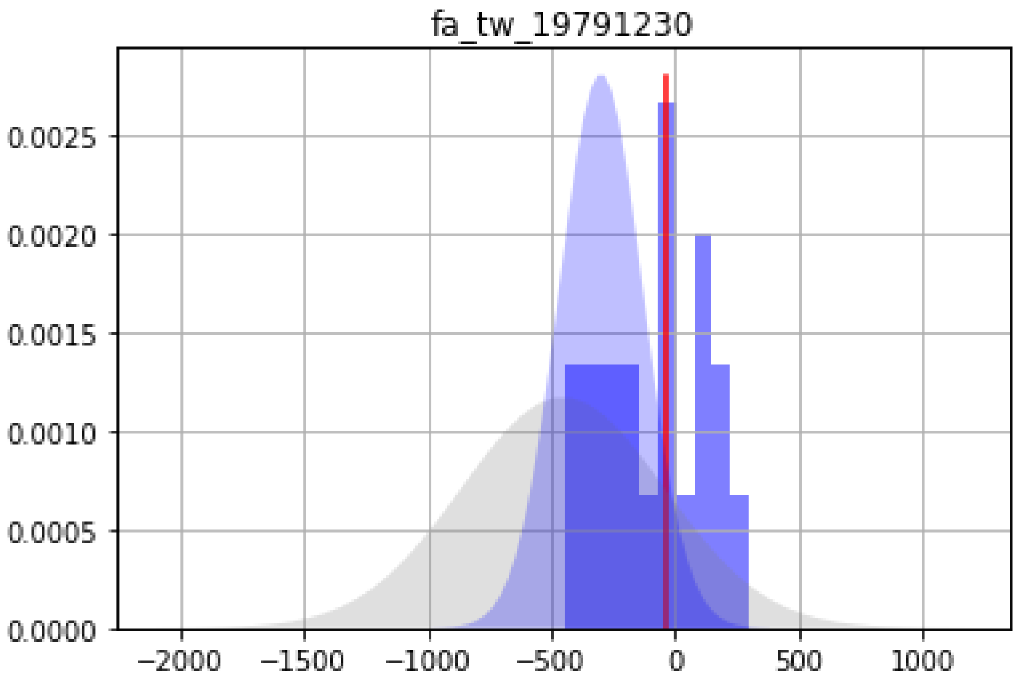
```
Out[17]:                     prior_mean  prior_stdev  prior_lower_bound  \
         name
         fa_hw_19791230       -977.2390    295.32800         -1567.8900
         fa_hw_19801229       -351.2160    409.77000         -1170.7600
         fa_tw_19791230       -453.0330    409.35100         -1271.7400
         fa_tw_19801229        108.9600    506.73200          -904.5040
         hds_00_013_002_000     39.6102      3.96314            31.6840
         hds_00_013_002_001     38.3838      4.05782            30.2681
         part_status             2.0000      0.00000             2.0000
         part_time             907.7020    570.98600          -234.2690

                             prior_upper_bound  post_mean  post_stdev  \
         name
         fa_hw_19791230              -386.5840  -675.0430  247.837000
         fa_hw_19801229               468.3240    37.6125  335.549000
         fa_tw_19791230               365.6690  -299.4690  170.577000
         fa_tw_19801229              1122.4200   333.7460  263.672000
         hds_00_013_002_000            47.5365    37.6359    0.273990
         hds_00_013_002_001            46.4994    36.4927    0.687833
         part_status                    2.0000     2.0000    0.000000
         part_time                   2049.6700   703.9470  436.945000

                             post_lower_bound  post_upper_bound
         name
         fa_hw_19791230            -1170.7200         -179.3700
         fa_hw_19801229             -633.4850          708.7100
         fa_tw_19791230             -640.6230           41.6858
         fa_tw_19801229             -193.5970          861.0890
         hds_00_013_002_000           37.0879           38.1838
         hds_00_013_002_001           35.1171           37.8684
         part_status                   2.0000            2.0000
         part_time                  -169.9420         1577.8400
```

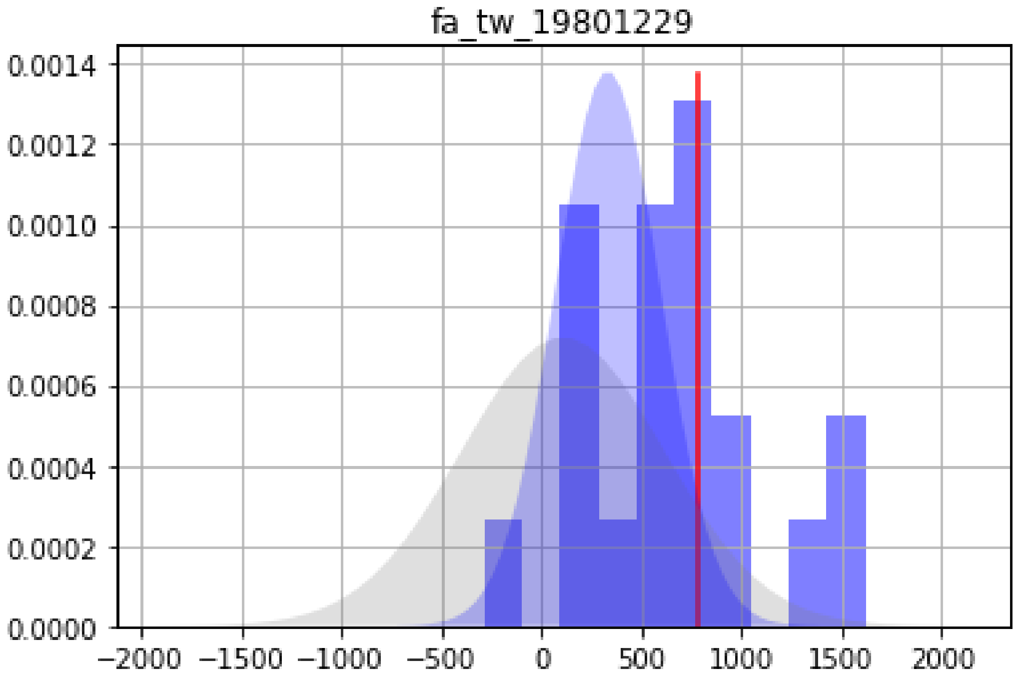
```
In [18]: obs = pst.observation_data
         fnames = pst.pestpp_options["forecasts"].split(",")
         for forecast in fnames:
             ax = plt.subplot(111)
             oe_pt.loc[:,forecast].hist(ax=ax,color="b",alpha=0.5,normed=True)
             ax.plot([obs.loc[forecast,"obsval"],obs.loc[forecast,"obsval"]],ax.get_ylim(),"r"
             axt = plt.twinx()
             x,y = pyemu.plot_utils.gaussian_distribution(f_df.loc[forecast,"prior_mean"],f_df
             axt.fill_between(x,0,y,facecolor="0.5",alpha=0.25)
```

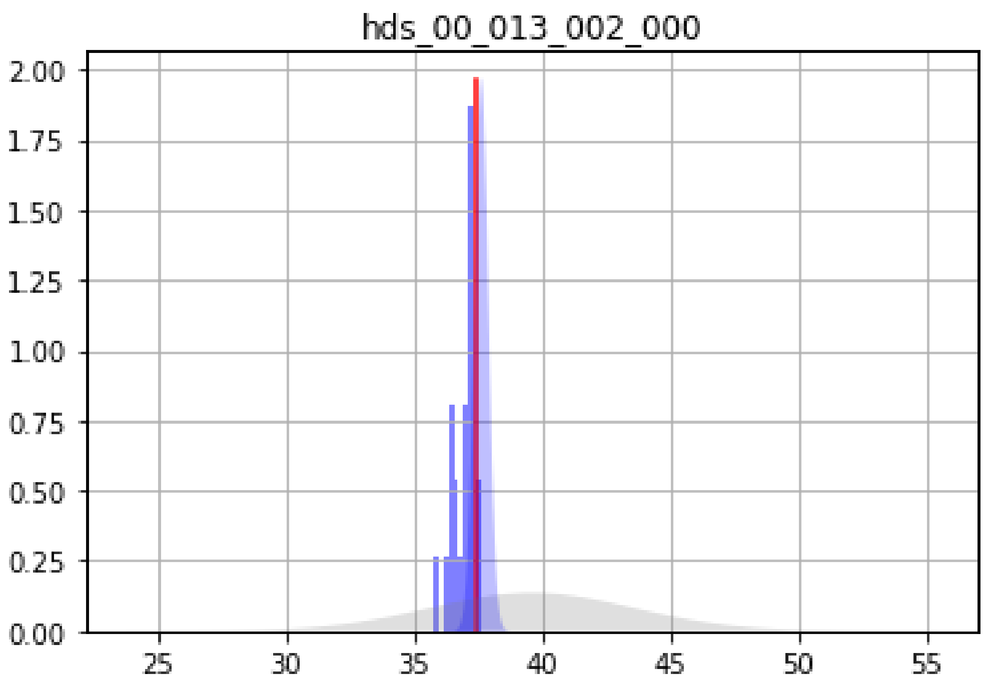
```
x,y = pyemu.plot_utils.gaussian_distribution(f_df.loc[forecast,"post_mean"],f_df.l
axt.fill_between(x,0,y,facecolor="b",alpha=0.25)
axt.set_ylim(0,axt.get_ylim()[1])
axt.set_yticks([])
ax.set_title(forecast)
plt.show()
```

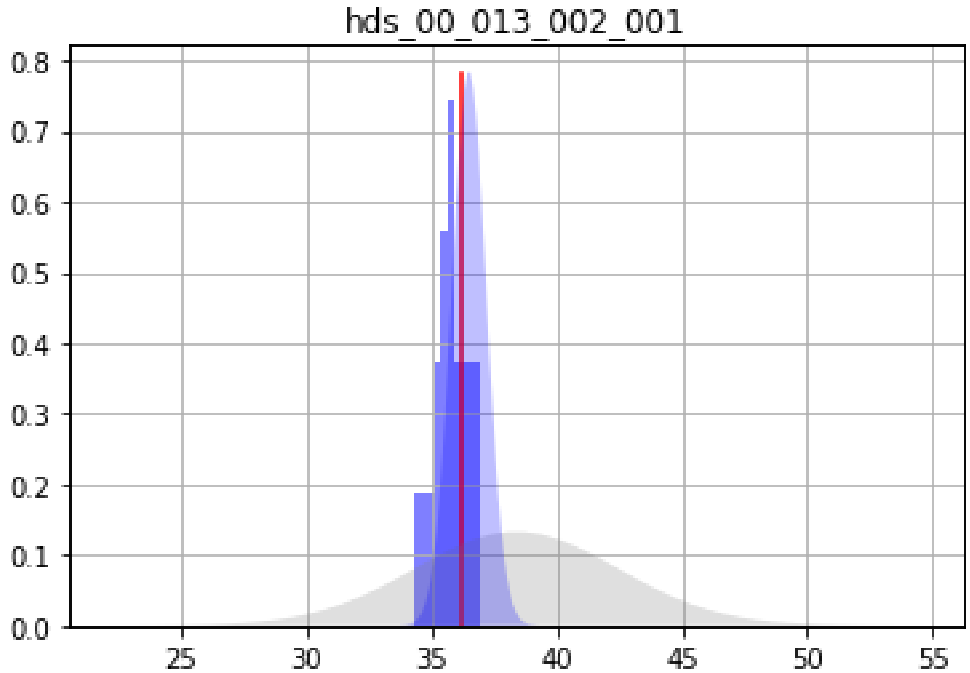


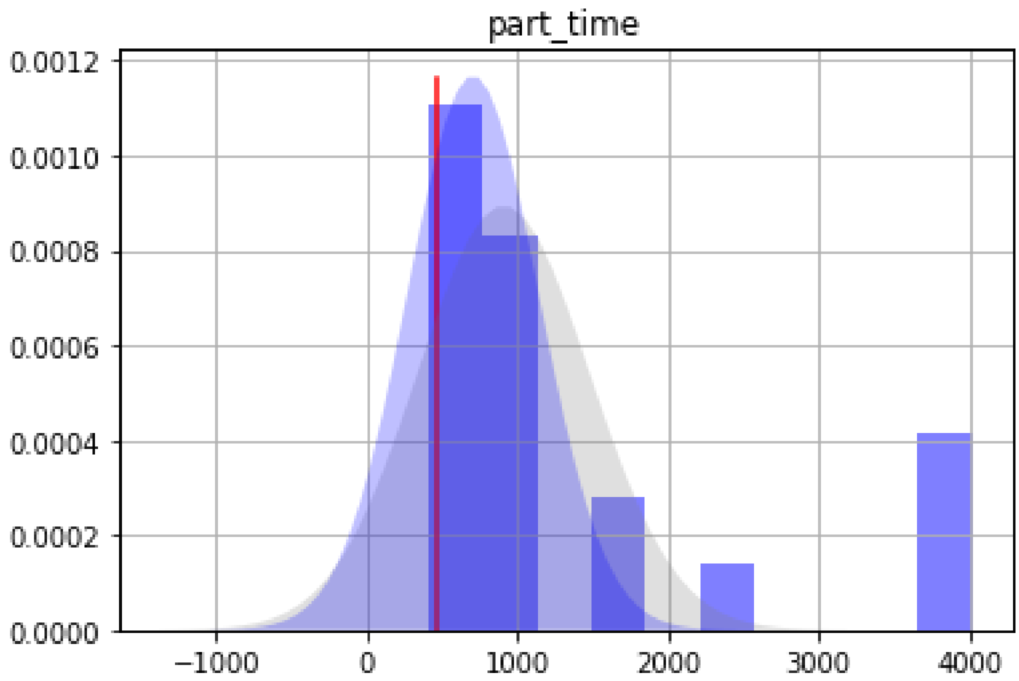

fa_hw_19801229
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
−2000
−1500
−1000
−500
0
500
1000
1500
2000

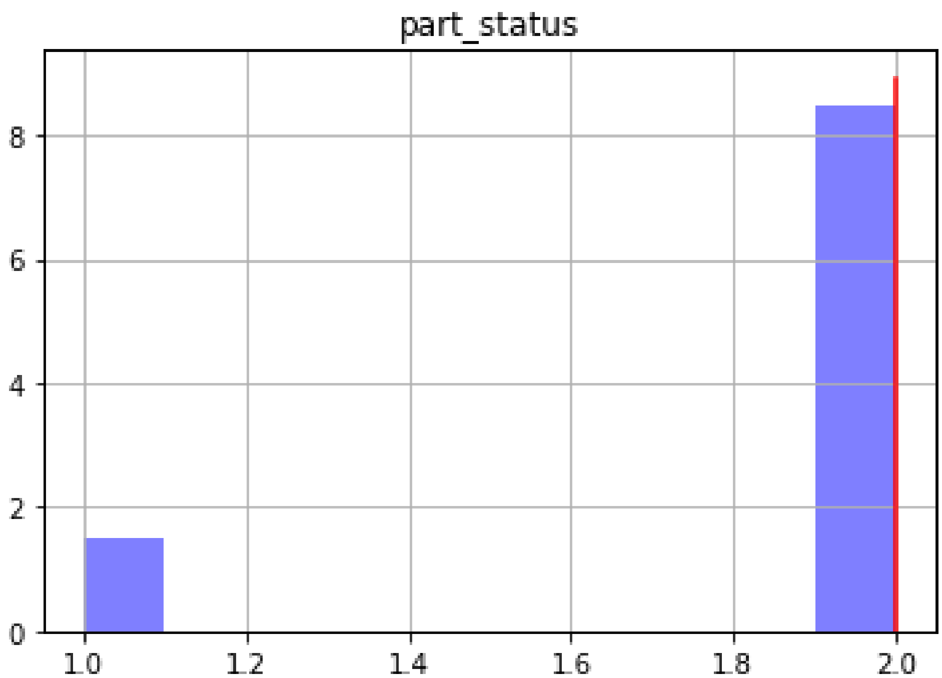

fa_tw_19791230
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
−2000
−1500
−1000
−500
0
500
1000


fa_tw_19801229
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
−2000
−1500
−1000
−500
0
500
1000
1500
2000


hds_00_013_002_000
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
25
30
35
40
45
50
55


hds_00_013_002_001
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
25
30
35
40
45
50
55


part_time
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
−1000
0
1000
2000
3000
4000



1.1.2 Setup of Tikhonov regularization

Now lets setup and use some formal regularization to bring the final phi up to around 14. We will use first-order regularization based on the covariance matrix we build earlier:

```
In [19]: cov = pyemu.Cov.from_binary(os.path.join(t_d,"prior_cov.jcb"))
```

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

```
In [20]: pyemu.helpers.first_order_pearson_tikhonov(pst,cov)
```

```
getting CC matrix
processing
```

```
In [21]: pst.prior_information.head()
```

```
Out[21]:                                                     equation    obgnme  \
         pilbl
         pcc_1  1.0 * log(dc0000390005) - 1.0 * log(dc0000390006) = 0.0  regul_cc
         pcc_2  1.0 * log(dc0000390005) - 1.0 * log(dc0000390007) = 0.0  regul_cc
         pcc_3  1.0 * log(dc0000390005) - 1.0 * log(dc0000390008) = 0.0  regul_cc
         pcc_4  1.0 * log(dc0000390005) - 1.0 * log(dc0000390009) = 0.0  regul_cc
         pcc_5  1.0 * log(dc0000390005) - 1.0 * log(dc0000390010) = 0.0  regul_cc
```

```
       pilbl    weight
pilbl
pcc_1  pcc_1  0.904837
pcc_2  pcc_2  0.818731
pcc_3  pcc_3  0.740818
pcc_4  pcc_4  0.670320
pcc_5  pcc_5  0.606531
```

In [22]:
```
shutil.copy2(os.path.join(m_d,"freyberg_pp.jcb"),os.path.join(t_d,"restart_pp.jcb"))
```

Out[22]:
```
'template/restart_pp.jcb'
```

In [23]:
```
pst.pestpp_options["base_jacobian"] = "restart_pp.jcb"
pst.reg_data.phimlim = pst.nnz_obs
pst.reg_data.phimaccept = pst.reg_data.phimlim * 1.1
pst.write(os.path.join(t_d,"freyberg_pp.pst"))
```

```
noptmax:3, npar_adj:527, nnz_obs:14
```

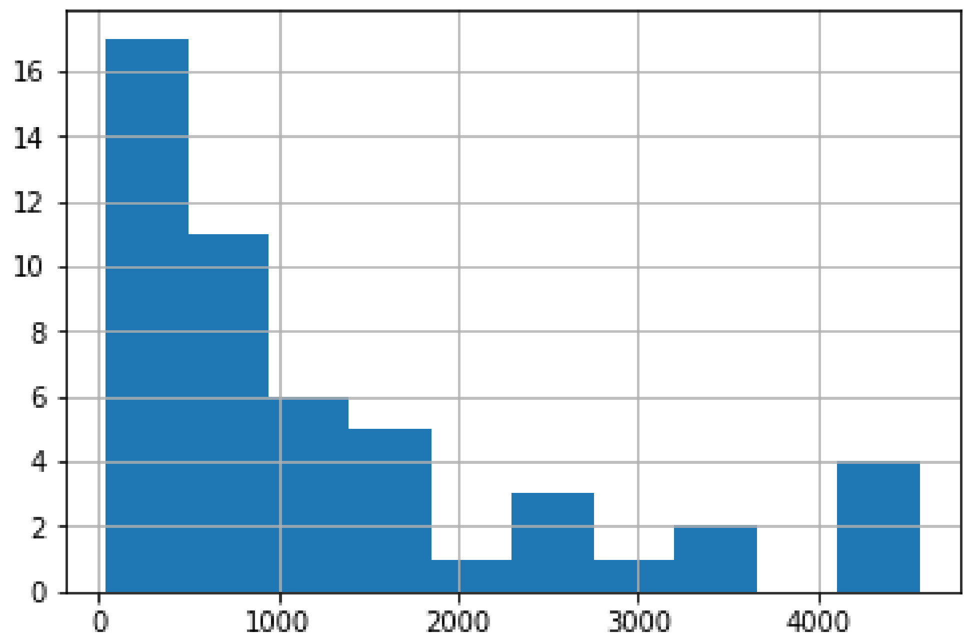
In [24]:
```
pyemu.os_utils.start_slaves(t_d,"pestpp-glm","freyberg_pp.pst",num_slaves=num_workers
                            master_dir=m_d)
```

In [25]:
```
df = df=pd.read_csv(os.path.join(m_d,"freyberg_pp.post.obsen.csv"),index_col=0)
oe = pyemu.ObservationEnsemble.from_dataframe(pst=pst,df=df)
```

In [26]:
```
ax = oe.phi_vector.hist()#bins=np.linspace(0,100,20))
oe.phi_vector.sort_values().iloc[:20]
```

Out[26]:
```
real_name
35     43.758830
10     66.299790
46     71.376560
12    118.219687
24    119.728221
8     140.461733
22    176.808587
21    200.494414
47    225.582200
30    255.737081
26    273.099728
44    290.829433
48    291.287982
5     321.916424
23    420.424853
38    430.891678
49    432.520905
29    522.375353
```

```
41    527.057867
33    566.113001
dtype: float64
```



Same as before, to get a "posterior" ensemble, we need to throw out the realizations with large phi - lets just take the 20 best:

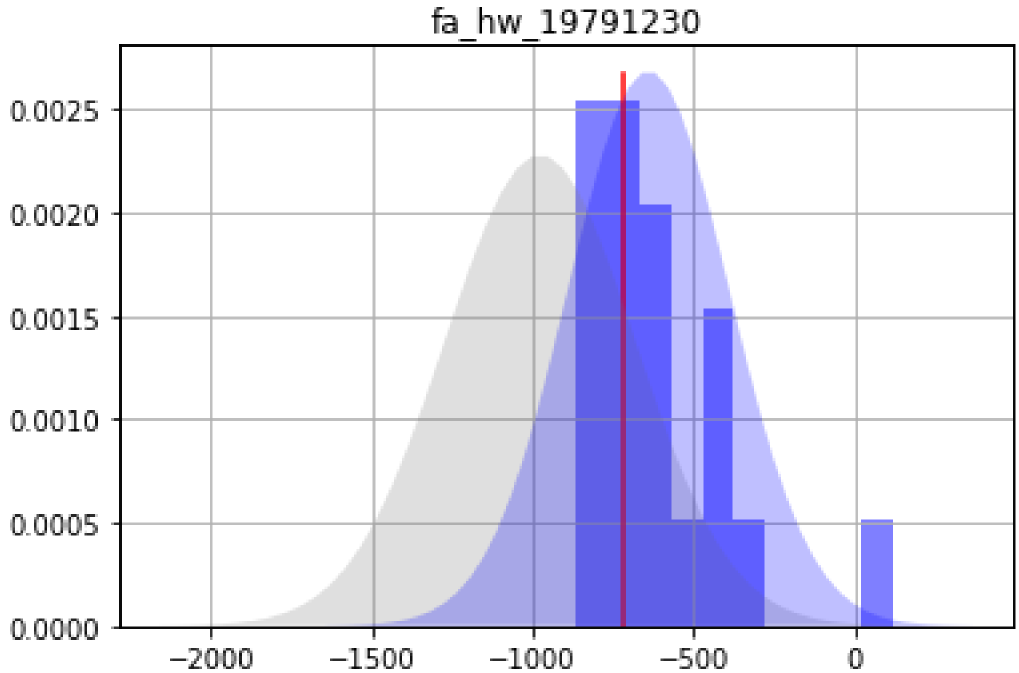
In [27]:
```
oe_pt = oe.loc[oe.phi_vector.sort_values().index[:20],:]
```

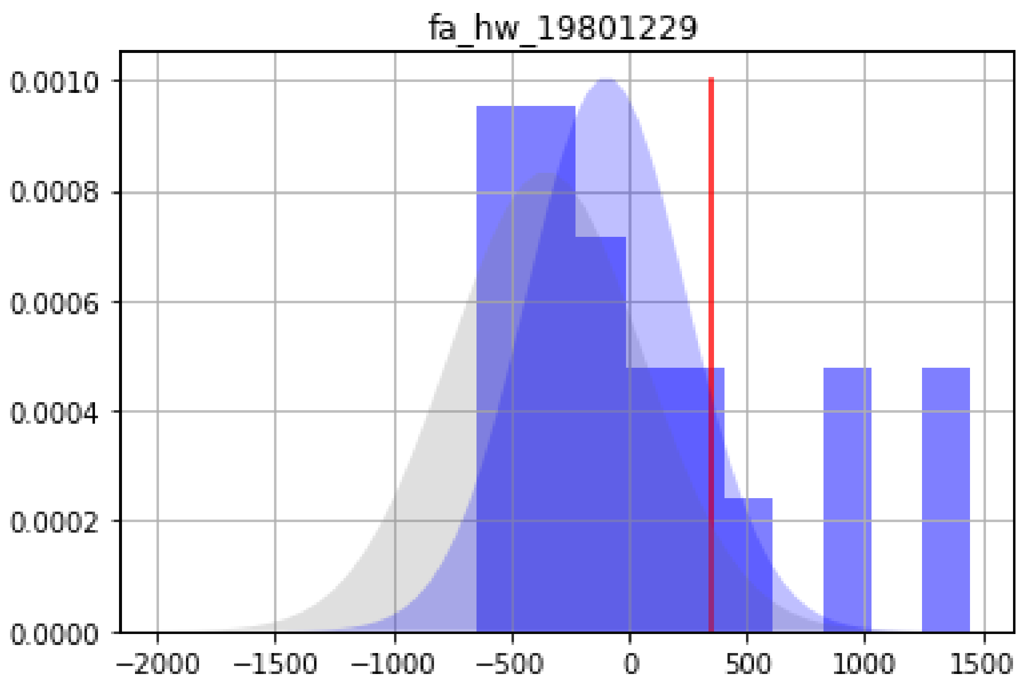
In [28]:
```
f_df = pd.read_csv(os.path.join(m_d,"freyberg_pp.pred.usum.csv"),index_col=0)
f_df.index = f_df.index.map(str.lower)
f_df
```

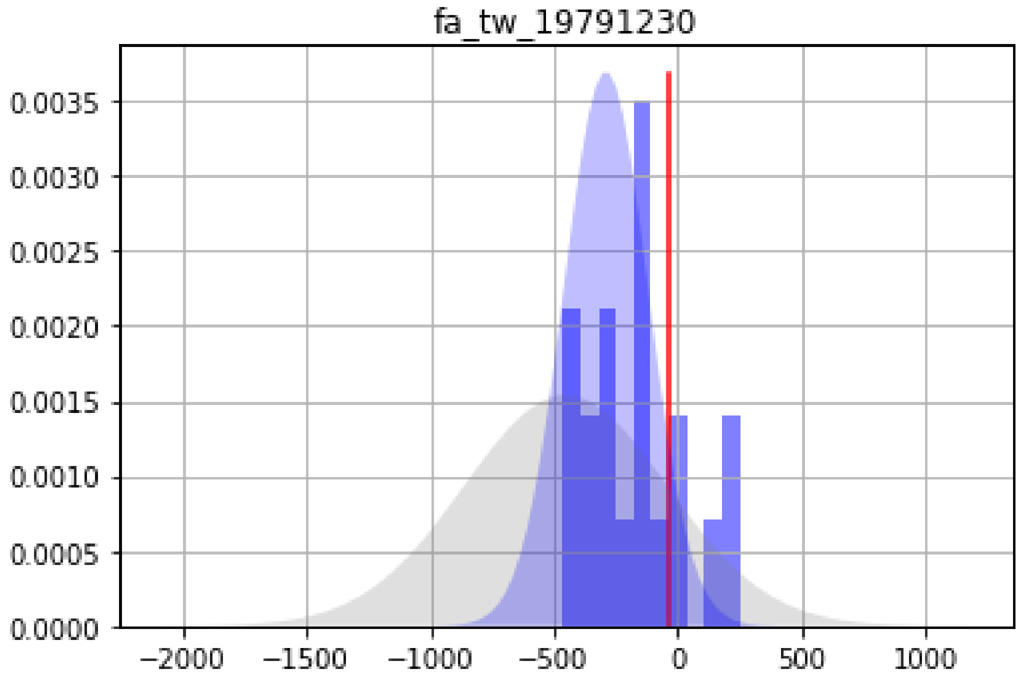
Out[28]:
```
                      prior_mean  prior_stdev  prior_lower_bound  \
name
fa_hw_19791230         -977.2390    295.32800         -1567.8900
fa_hw_19801229         -351.2160    409.77000         -1170.7600
fa_tw_19791230         -453.0330    409.35100         -1271.7400
fa_tw_19801229          108.9600    506.73200          -904.5040
hds_00_013_002_000       39.6102      3.96314            31.6840
hds_00_013_002_001       38.3838      4.05782            30.2681
part_status               2.0000      0.00000             2.0000
part_time               907.7020    570.98600          -234.2690

                      prior_upper_bound  post_mean  post_stdev  \
name
fa_hw_19791230                -386.5840  -640.3370  250.802000
```

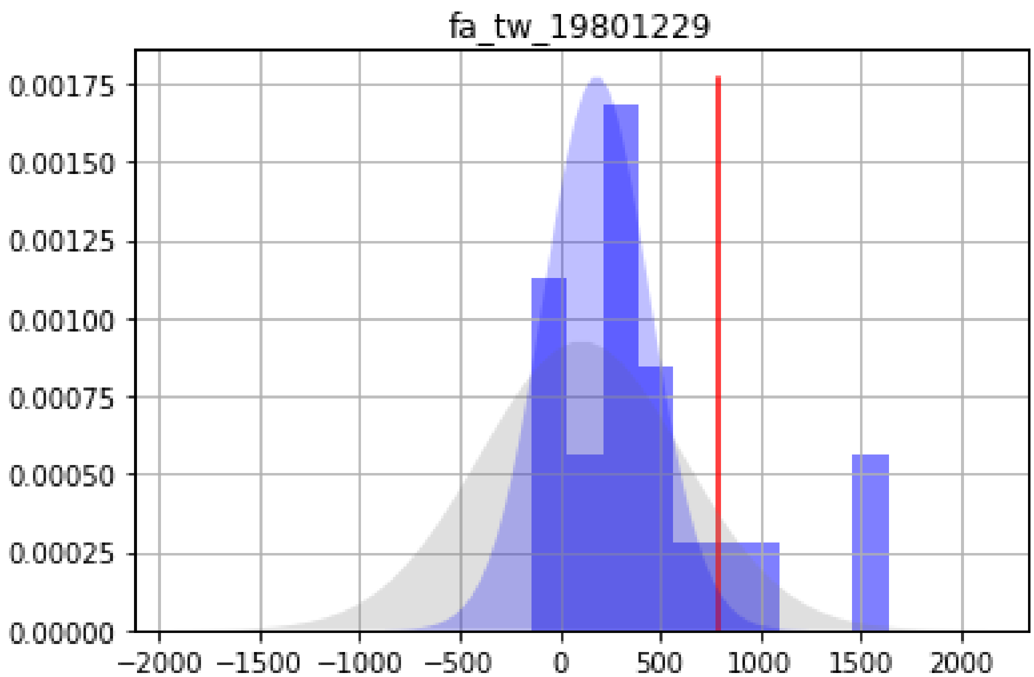
```
fa_hw_19801229          468.3240   -94.8227  339.971000
fa_tw_19791230          365.6690  -290.4730  170.616000
fa_tw_19801229         1122.4200   184.3570  263.999000
hds_00_013_002_000       47.5365    37.7893    0.274188
hds_00_013_002_001       46.4994    36.9350    0.687976
part_status               2.0000     2.0000    0.000000
part_time              2049.6700   966.2250  437.215000

                    post_lower_bound  post_upper_bound
name
fa_hw_19791230            -1141.9400         -138.7340
fa_hw_19801229             -774.7640          585.1190
fa_tw_19791230             -631.7040           50.7581
fa_tw_19801229             -343.6410          712.3540
hds_00_013_002_000           37.2409           38.3377
hds_00_013_002_001           35.5591           38.3110
part_status                   2.0000            2.0000
part_time                    91.7946         1840.6600
```

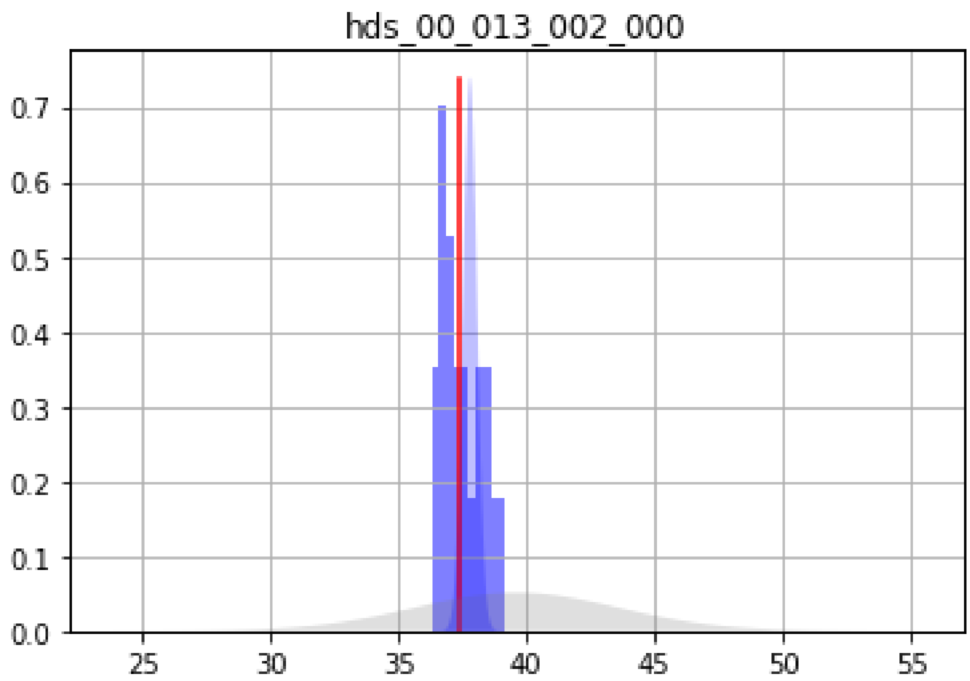
In [29]:
```
obs = pst.observation_data
fnames = pst.pestpp_options["forecasts"].split(",")
for forecast in fnames:
    ax = plt.subplot(111)
    oe_pt.loc[:,forecast].hist(ax=ax,color="b",alpha=0.5,normed=True)
    ax.plot([obs.loc[forecast,"obsval"],obs.loc[forecast,"obsval"]],ax.get_ylim(),"r"
    axt = plt.twinx()
    x,y = pyemu.plot_utils.gaussian_distribution(f_df.loc[forecast,"prior_mean"],f_df
    axt.fill_between(x,0,y,facecolor="0.5",alpha=0.25)
    x,y = pyemu.plot_utils.gaussian_distribution(f_df.loc[forecast,"post_mean"],f_df.
    axt.fill_between(x,0,y,facecolor="b",alpha=0.25)
    axt.set_ylim(0,axt.get_ylim()[1])
    axt.set_yticks([])
    ax.set_title(forecast)
    plt.show()
```

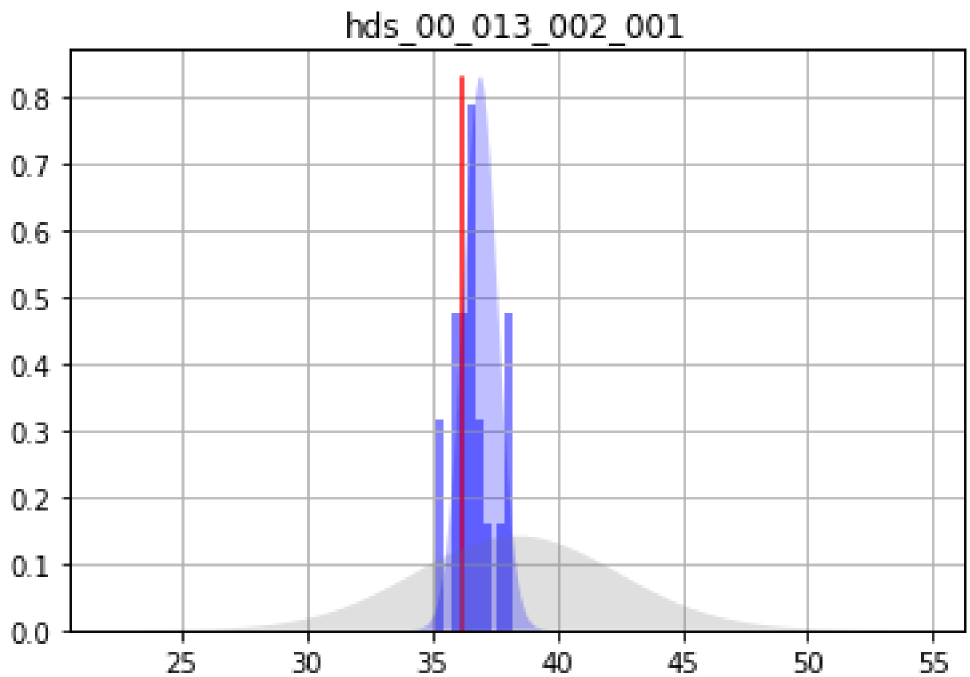

fa_hw_19791230
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
−2000
−1500
−1000
−500
0

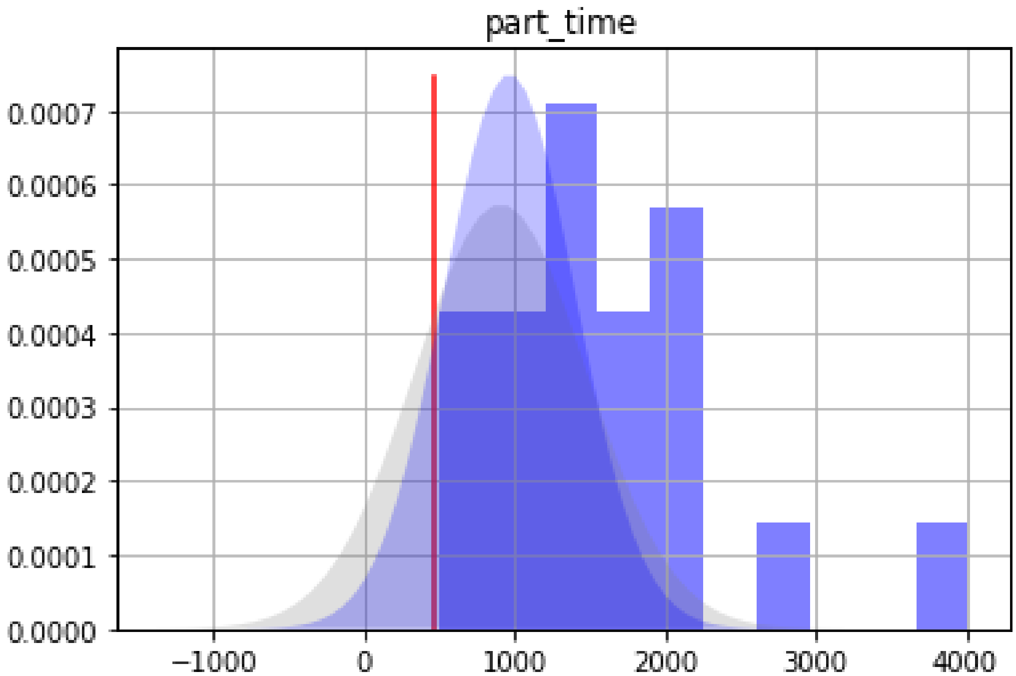

fa_hw_19801229
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
−2000
−1500
−1000
−500
0
500
1000
1500

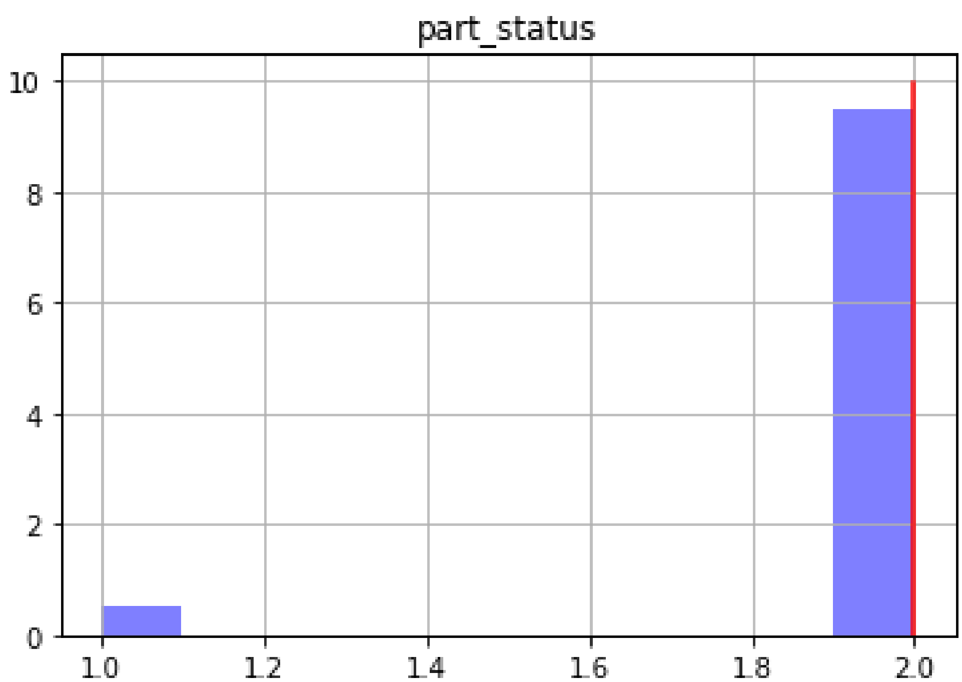

fa_tw_19791230
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
−2000
−1500
−1000
−500
0
500
1000


fa_tw_19801229
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
−2000
−1500
−1000
−500
0
500
1000
1500
2000


hds_00_013_002_000
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
25
30
35
40
45
50
55


hds_00_013_002_001
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
25
30
35
40
45
50
55


part_time
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
−1000
0
1000
2000
3000
4000


part_status
10
8
6
4
2
0
1.0
1.2
1.4
1.6
1.8
2.0