pestpp-glm

May 11, 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
```

In [2]:
```
t_d = "template"
m_d = "master_glm"
```

In [3]:
```
pst = pyemu.Pst(os.path.join(t_d,"freyberg.pst"))
pst.write_par_summary_table(filename="none")
```

Out[3]:

	type	transform	count	initial value	\
gr_hk3	gr_hk3	log	705	0	
welflux_k02	welflux_k02	log	6	0	
cn_strt6	cn_strt6	log	1	0	
gr_ss4	gr_ss4	log	705	0	
gr_sy3	gr_sy3	log	705	0	
cn_sy6	cn_sy6	log	1	0	
cn_hk6	cn_hk6	log	1	0	
pp_hk1	pp_hk1	log	32	0	
cn_ss7	cn_ss7	log	1	0	
cn_rech4	cn_rech4	log	1	0	
gr_strt3	gr_strt3	log	705	0	

welflux	welflux	log	2	0 to 0.176091
pp_vka1	pp_vka1	log	32	0
cn_prsity8	cn_prsity8	log	1	0
pp_prsity0	pp_prsity0	log	32	0
cn_ss8	cn_ss8	log	1	0
drncond_k00	drncond_k00	log	10	0
pp_ss2	pp_ss2	log	32	0
pp_hk0	pp_hk0	log	32	0
pp_strt1	pp_strt1	log	32	0
gr_strt5	gr_strt5	log	705	0
cn_vka8	cn_vka8	log	1	0
pp_strt0	pp_strt0	log	32	0
pp_prsity1	pp_prsity1	log	32	0
gr_vka5	gr_vka5	log	705	0
cn_hk7	cn_hk7	log	1	0
cn_sy7	cn_sy7	log	1	0
pp_sy2	pp_sy2	log	32	0
pp_vka0	pp_vka0	log	32	0
gr_hk4	gr_hk4	log	705	0
...	...	...	...	...
gr_sy5	gr_sy5	log	705	0
pp_vka2	pp_vka2	log	32	0
cn_ss6	cn_ss6	log	1	0
pp_ss0	pp_ss0	log	32	0
gr_prsity3	gr_prsity3	log	705	0
cn_hk8	cn_hk8	log	1	0
cn_prsity7	cn_prsity7	log	1	0
cn_strt7	cn_strt7	log	1	0
pp_rech0	pp_rech0	log	32	0
gr_vka4	gr_vka4	log	705	0
cn_rech5	cn_rech5	log	1	-0.39794
pp_sy0	pp_sy0	log	32	0
cn_vka7	cn_vka7	log	1	0
pp_strt2	pp_strt2	log	32	0
cn_sy8	cn_sy8	log	1	0
pp_prsity2	pp_prsity2	log	32	0
pp_sy1	pp_sy1	log	32	0
gr_rech3	gr_rech3	log	705	0
cn_vka6	cn_vka6	log	1	0
gr_vka3	gr_vka3	log	705	0
pp_hk2	pp_hk2	log	32	0
pp_rech1	pp_rech1	log	32	0
strk	strk	log	40	0
cn_strt8	cn_strt8	log	1	0
cn_prsity6	cn_prsity6	log	1	0
gr_prsity4	gr_prsity4	log	705	0
gr_sy4	gr_sy4	log	705	0
pp_ss1	pp_ss1	log	32	0

gr_prsity5	gr_prsity5	log	705	0
gr_ss5	gr_ss5	log	705	0

	upper bound	lower bound	standard deviation
gr_hk3	1	-1	0.5
welflux_k02	1	-1	0.5
cn_strt6	0.0211893	-0.0222764	0.0108664
gr_ss4	1	-1	0.5
gr_sy3	0.243038	-0.60206	0.211275
cn_sy6	0.243038	-0.60206	0.211275
cn_hk6	1	-1	0.5
pp_hk1	1	-1	0.5
cn_ss7	1	-1	0.5
cn_rech4	0.0791812	-0.09691	0.0440228
gr_strt3	0.0211893	-0.0222764	0.0108664
welflux	0.176091 to 0.30103	-0.30103 to 0	0.0752575 to 0.11928
pp_vka1	1	-1	0.5
cn_prsity8	0.176091	-0.30103	0.11928
pp_prsity0	0.176091	-0.30103	0.11928
cn_ss8	1	-1	0.5
drncond_k00	1	-1	0.5
pp_ss2	1	-1	0.5
pp_hk0	1	-1	0.5
pp_strt1	0.0211893	-0.0222764	0.0108664
gr_strt5	0.0211893	-0.0222764	0.0108664
cn_vka8	1	-1	0.5
pp_strt0	0.0211893	-0.0222764	0.0108664
pp_prsity1	0.176091	-0.30103	0.11928
gr_vka5	1	-1	0.5
cn_hk7	1	-1	0.5
cn_sy7	0.243038	-0.60206	0.211275
pp_sy2	0.243038	-0.60206	0.211275
pp_vka0	1	-1	0.5
gr_hk4	1	-1	0.5
...	...	...	...
gr_sy5	0.243038	-0.60206	0.211275
pp_vka2	1	-1	0.5
cn_ss6	1	-1	0.5
pp_ss0	1	-1	0.5
gr_prsity3	0.176091	-0.30103	0.11928
cn_hk8	1	-1	0.5
cn_prsity7	0.176091	-0.30103	0.11928
cn_strt7	0.0211893	-0.0222764	0.0108664
pp_rech0	0.0413927	-0.0457575	0.0217875
gr_vka4	1	-1	0.5
cn_rech5	-0.09691	-1	0.225772
pp_sy0	0.243038	-0.60206	0.211275
cn_vka7	1	-1	0.5

```
pp_strt2          0.0211893       -0.0222764        0.0108664
cn_sy8             0.243038         -0.60206         0.211275
pp_prsity2         0.176091         -0.30103          0.11928
pp_sy1             0.243038         -0.60206         0.211275
gr_rech3          0.0413927       -0.0457575       0.0217875
cn_vka6                   1               -1             0.5
gr_vka3                   1               -1             0.5
pp_hk2                    1               -1             0.5
pp_rech1          0.0413927       -0.0457575       0.0217875
strk                      2               -2               1
cn_strt8          0.0211893       -0.0222764       0.0108664
cn_prsity6         0.176091         -0.30103          0.11928
gr_prsity4         0.176091         -0.30103          0.11928
gr_sy4             0.243038         -0.60206         0.211275
pp_ss1                    1               -1             0.5
gr_prsity5         0.176091         -0.30103          0.11928
gr_ss5                    1               -1             0.5

[65 rows x 7 columns]
```

1.0.1 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 [4]:
```
par = pst.parameter_data
```

In [5]:
```
# 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[5]: 719

In [6]:
```
# 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[6]: 719

In [7]:
```
par.loc[par.pargp.apply(lambda x: "pp" in x),"pargp"].unique()
```

Out[7]:
```
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 [8]:
```
#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[8]: 719

In [9]:
```
adj_par = par.loc[par.partrans=="log",:]
adj_par.pargp.value_counts().sort_values()
```

Out[9]:

cn_strt6	1
flow	1
cn_vka6	1
cn_sy6	1
cn_hk6	1
cn_prsity6	1
cn_vka8	1
cn_ss7	1
cn_rech4	1
cn_rech5	1
cn_prsity8	1
cn_sy8	1
cn_ss8	1
cn_hk8	1
cn_prsity7	1
cn_ss6	1
cn_strt7	1
cn_vka7	1
cn_strt8	1
cn_hk7	1
cn_sy7	1
welflux	2
welflux_k02	6
drncond_k00	10
pp_vka0	32
pp_ss1	32
pp_sy2	32
pp_prsity1	32
pp_strt0	32
pp_vka2	32
pp_strt1	32
pp_hk2	32
pp_ss2	32
pp_rech0	32
pp_hk0	32
pp_strt2	32
pp_prsity0	32
pp_sy1	32

```
pp_prsity2     32
pp_rech1       32
pp_vka1        32
pp_ss0         32
pp_hk1         32
pp_sy0         32
strk           40
Name: pargp, dtype: int64
```

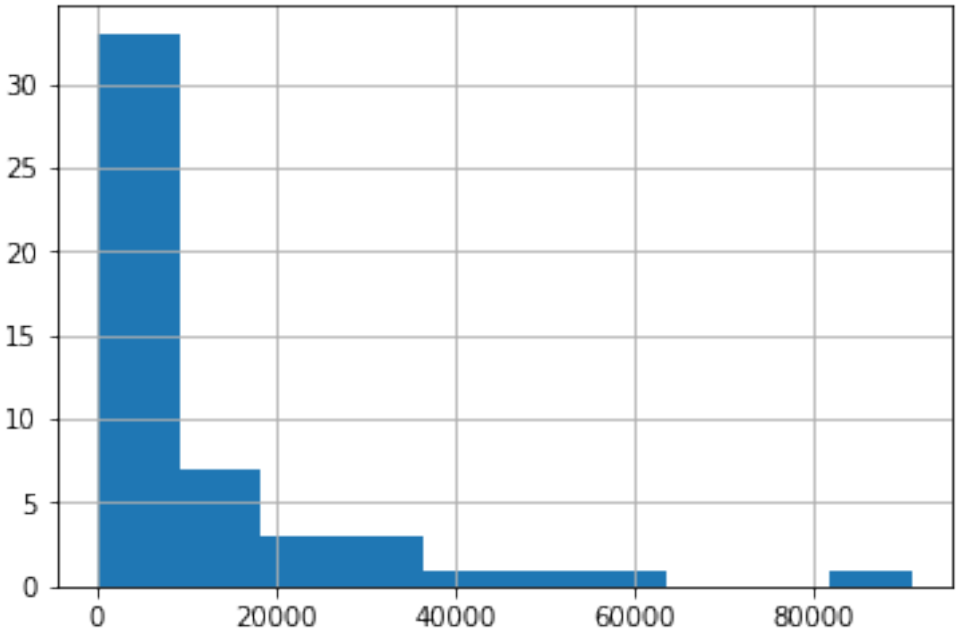
fix the future recharge pilot points, vka in layers 1 and 3 and the initial condition pilot points (we still have layer-scale pars for each of these types)

```
In [10]: 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[10]: 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 [11]: 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 [12]: pyemu.os_utils.start_slaves(t_d,"pestpp-glm","freyberg_pp.pst",num_slaves=20,slave_ro
                                      master_dir=m_d)

In [13]: 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 [14]: ax = oe.phi_vector.hist()#bins=np.linspace(0,100,20))
         oe.phi_vector.sort_values().iloc[:20]

Out[14]: real_name
         33      63.786419
         43      90.143013
         1      184.436878
         6      321.565514
         29     404.538556
         34     459.256343
```

```
30     489.588408
17     504.033667
8      604.530926
0      762.317848
5      952.700910
41     997.878498
38    1231.514009
37    1366.955952
28    1485.953746
48    1536.732656
47    1909.722843
7     2203.148239
2     2225.534662
25    2879.283881
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 [15]: oe_pt = oe.loc[oe.phi_vector.sort_values().index[:20],:] #just take the 20 lowest phi
```

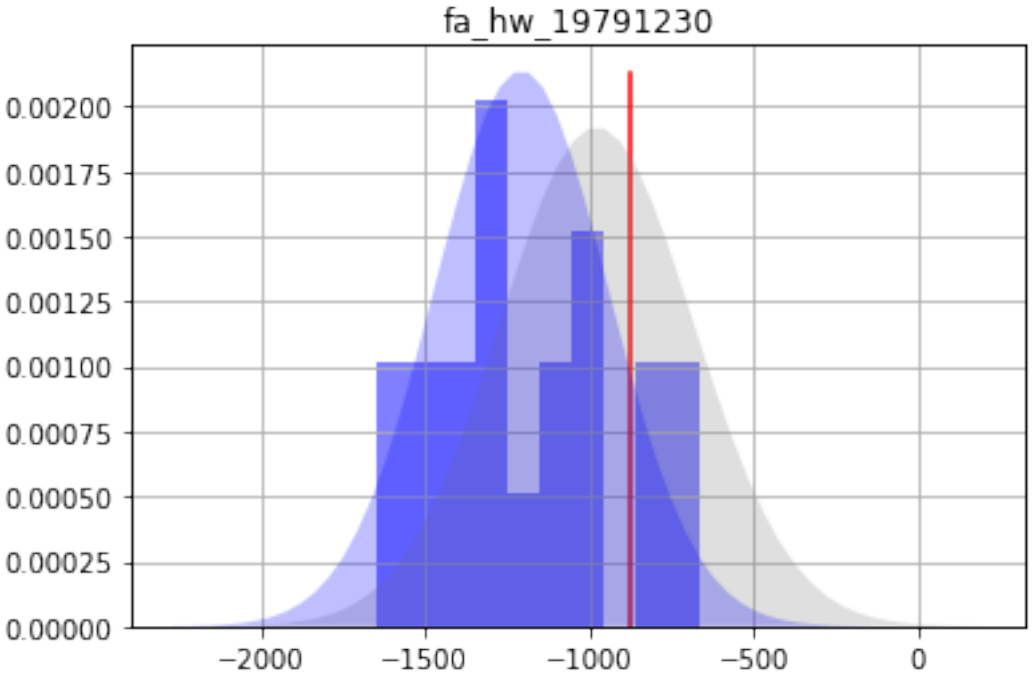
We can also load and plot the FOSM forecast results along side of the ensemble results:

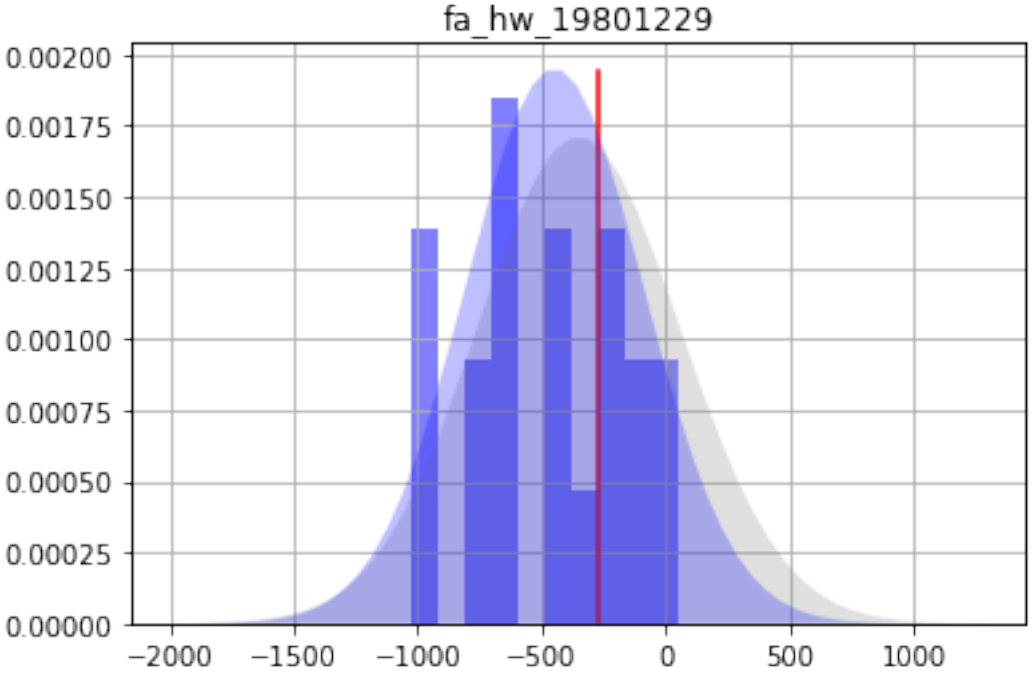
```
In [16]: f_df = pd.read_csv(os.path.join(m_d,"freyberg_pp.pred.usum.csv"),index_col=0)
         f_df.index = f_df.index.map(str.lower)
         f_df
```

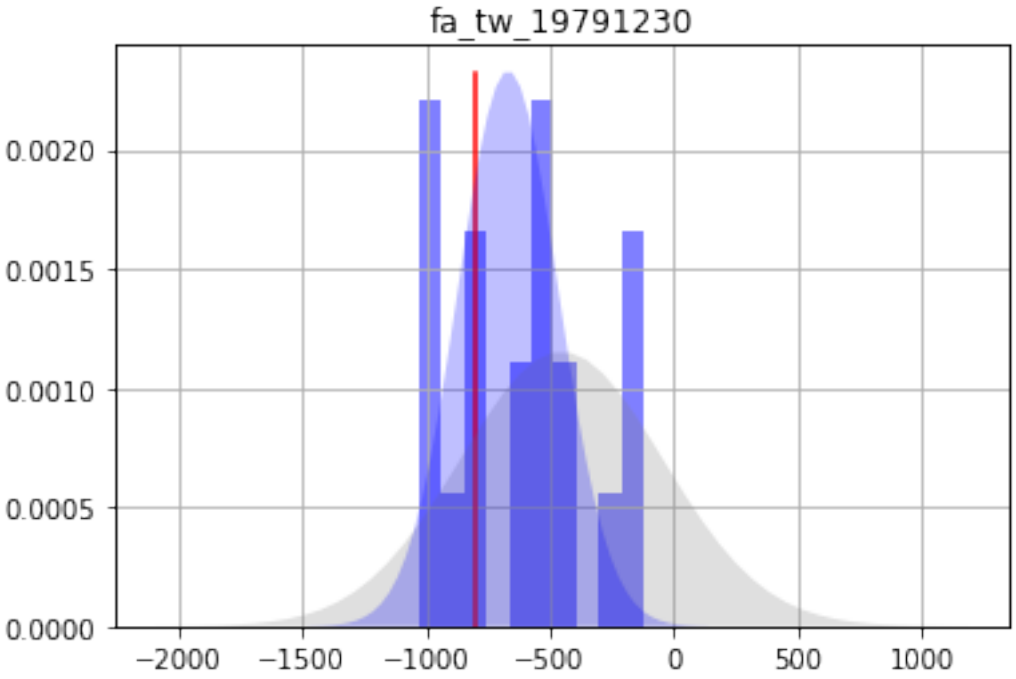
Out[16]:

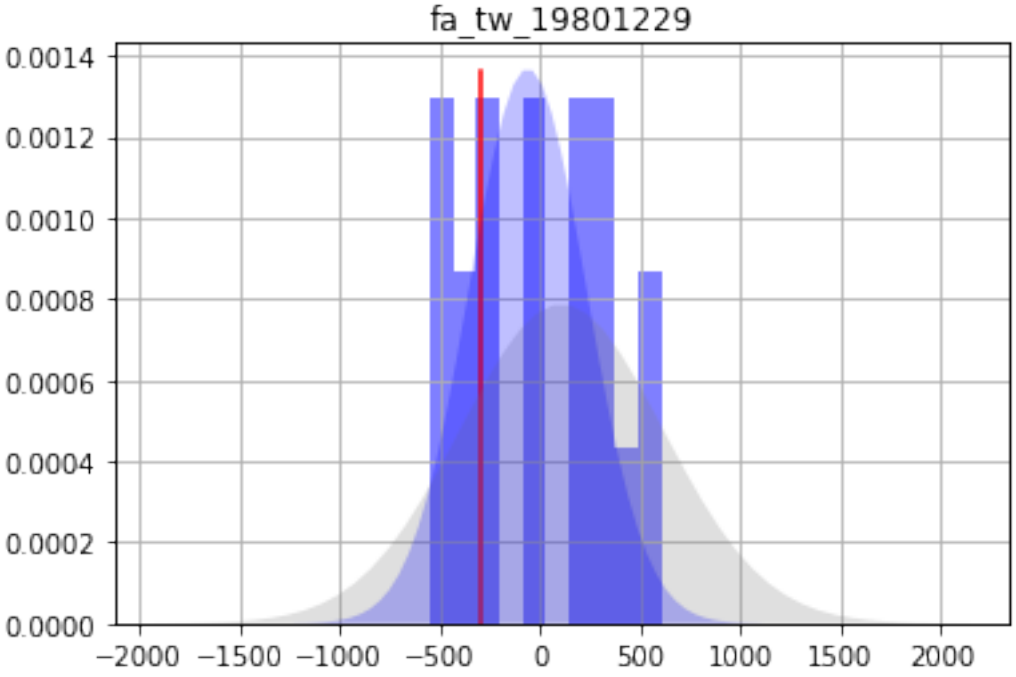
name	prior_mean	prior_stdev	prior_lower_bound	prior_upper_bound	post_mean	post_stdev	post_lower_bound	post_upper_bound
fa_hw_19791230	-977.2390	295.32800	-1567.8900	-386.5840	-1206.6900	265.343000	-1737.3800	-676.0050
fa_hw_19801229	-351.2160	409.77000	-1170.7600	468.3240	-448.8010	359.158000	-1167.1200	269.5150
fa_tw_19791230	-453.0330	409.35100	-1271.7400	365.6690	-668.7190	202.201000	-1073.1200	-264.3170
fa_tw_19801229	108.9600	506.73200	-904.5040	1122.4200	-60.3069	291.429000	-643.1650	522.5510
hds_00_013_002_000	39.6102	3.96314	31.6840	47.5365	44.9795	0.533666	43.9122	46.0468
hds_00_013_002_001	38.3838	4.05782	30.2681	46.4994	40.9583	0.824742	39.3088	42.6078
part_status	2.0000	0.00000	2.0000	2.0000	2.0000	0.000000	2.0000	2.0000
part_time	907.7020	570.98600	-234.2690	2049.6700	2468.6800	447.927000	1572.8200	3364.5300

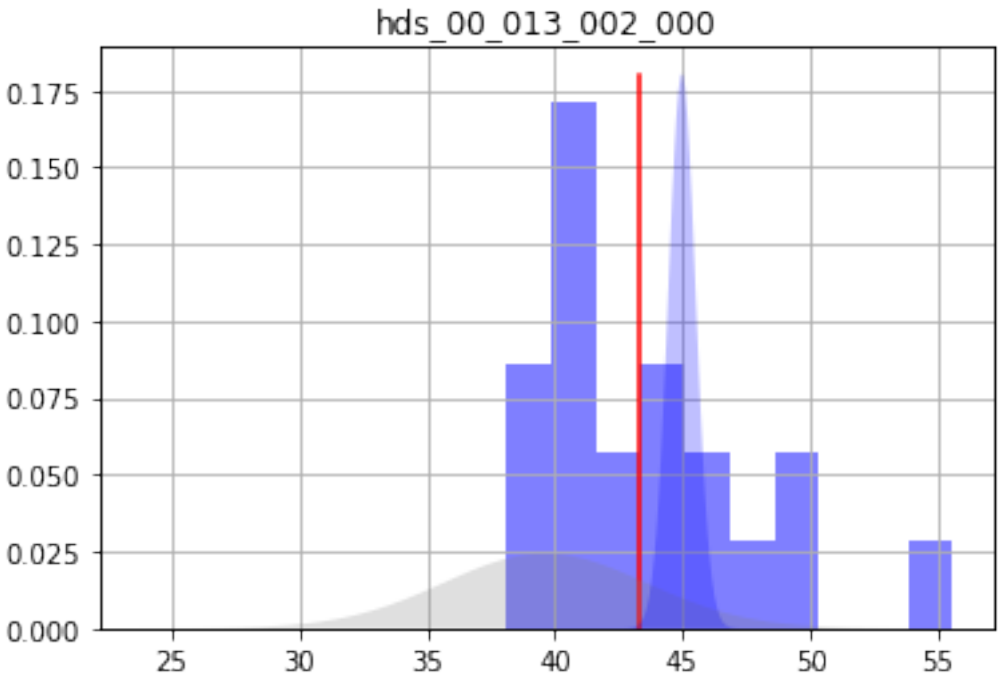
In [17]:
```
obs = pst.observation_data
fnames = pst.pestpp_options["forecasts"].split(",")
for forecast in fnames:
    ax = plt.subplot(111)
    oe_pt.loc[:,forecast].hist(ax=ax,color="b",alpha=0.5,normed=True)
    ax.plot([obs.loc[forecast,"obsval"],obs.loc[forecast,"obsval"]],ax.get_ylim(),"r"
    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()
```

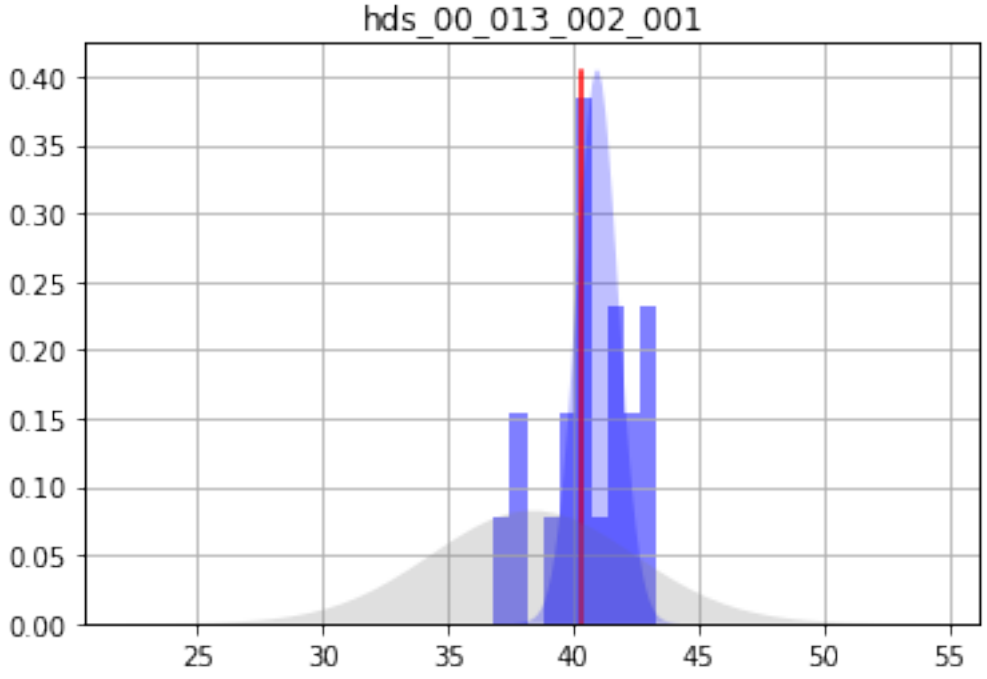

fa_hw_19791230
0.00200
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
−2000
−1500
−1000
−500
0

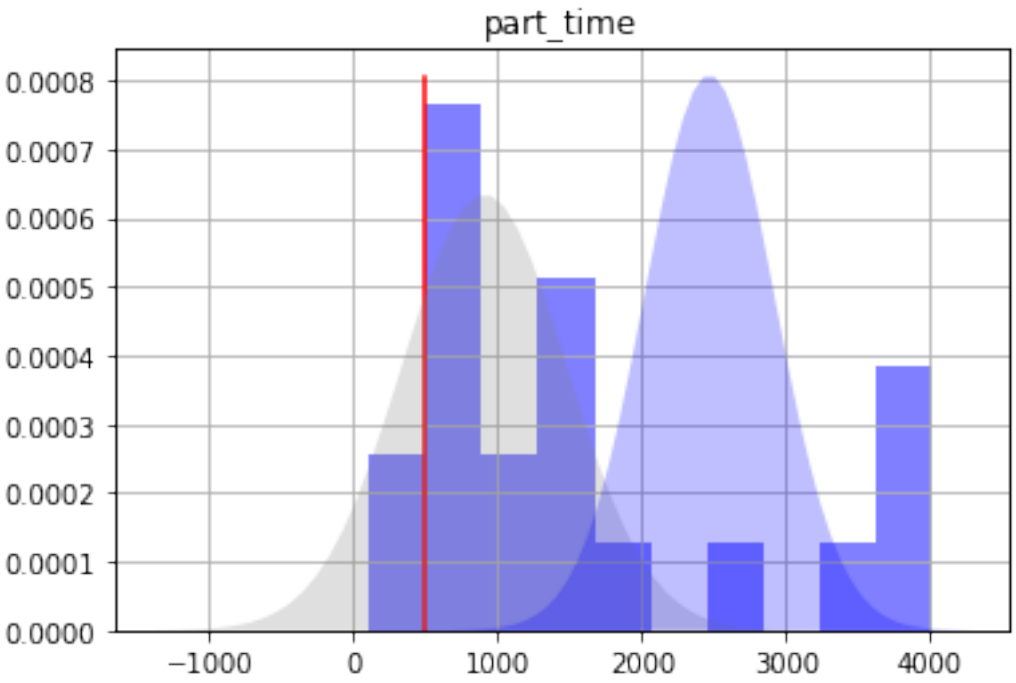

fa_hw_19801229
0.00200
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
−2000
−1500
−1000
−500
0
500
1000

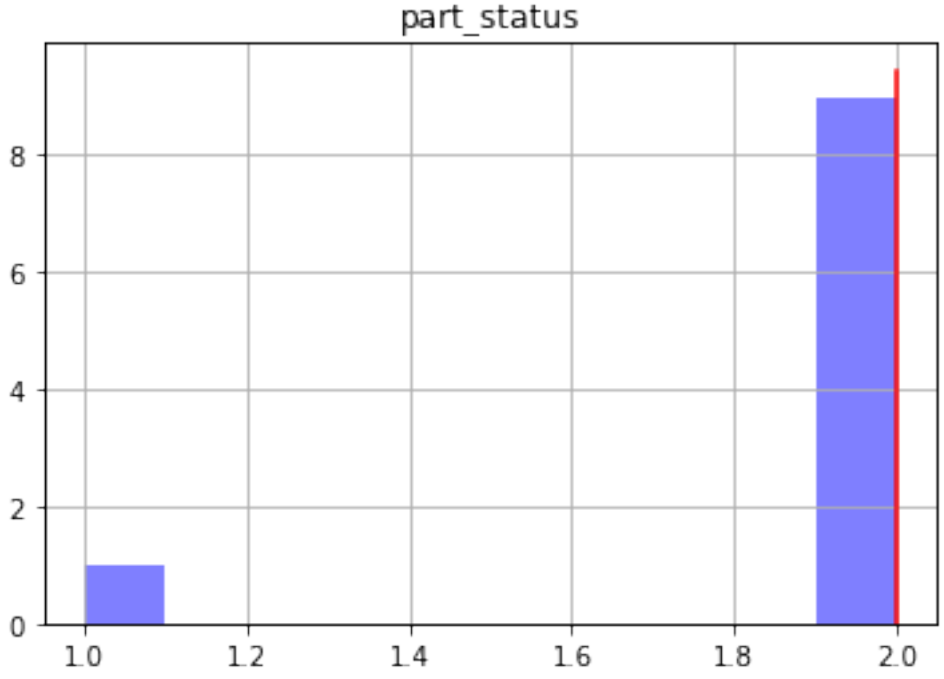

fa_tw_19791230
0.0020
0.0015
0.0010
0.0005
0.0000
−2000
−1500
−1000
−500
0
500
1000


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
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
25
30
35
40
45
50
55


hds_00_013_002_001
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
25
30
35
40
45
50
55


part_time
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
−1000
0
1000
2000
3000
4000


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

1.0.2 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 [18]: cov = pyemu.Cov.from_binary(os.path.join(t_d,"prior_cov.jcb"))
```

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

```
In [19]: pyemu.helpers.first_order_pearson_tikhonov(pst,cov)
```

```
getting CC matrix
processing
```

```
In [20]: pst.prior_information.head()
```

```
Out[20]:                                                    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 [21]: shutil.copy2(os.path.join(m_d,"freyberg_pp.jcb"),os.path.join(t_d,"restart_pp.jcb"))
```

```
Out[21]: 'template/restart_pp.jcb'
```

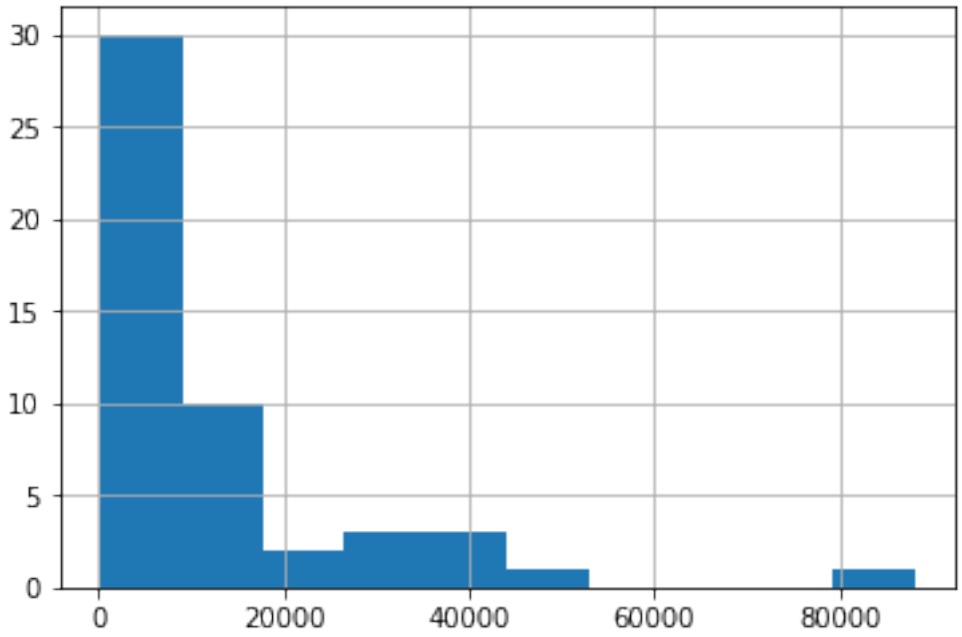
```
In [22]: pst.pestpp_options["base_jacobian"] = "restart_pp.jcb"
         pst.reg_data.phimlim = pst.nnz_obs
         pst.reg_data.phimaccept = pst.reg_data.phimlim * 1.1
         pst.write(os.path.join(t_d,"freyberg_pp.pst"))
```

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

```
In [23]: pyemu.os_utils.start_slaves(t_d,"pestpp-glm","freyberg_pp.pst",num_slaves=20,slave_ro
                                    master_dir=m_d)
```

```
In [24]: 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 [25]: ax = oe.phi_vector.hist()#bins=np.linspace(0,100,20))
         oe.phi_vector.sort_values().iloc[:20]

Out[25]: real_name
         33     216.592672
         47     303.224795
         17     466.756058
         30     658.484620
         29     672.767431
         41     695.510057
         34     785.550495
         0      904.885796
         38     918.962085
         22    1058.205607
         5     1130.472169
         28    1206.396515
         45    1386.504213
         1     1512.372225
         35    1687.069493
         8     1718.465045
         44    1719.941104
         6     2240.382246
         48    2618.349505
         16    2760.703071
         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 [26]: oe_pt = oe.loc[oe.phi_vector.sort_values().index[:20],:]
```

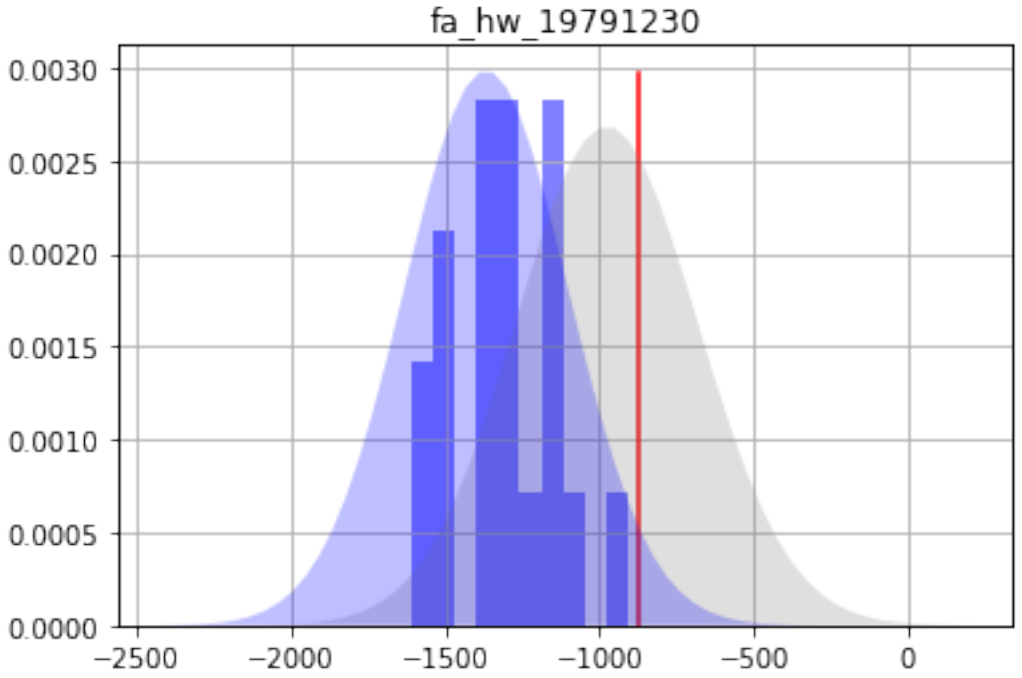
```
In [27]: f_df = pd.read_csv(os.path.join(m_d,"freyberg_pp.pred.usum.csv"),index_col=0)
         f_df.index = f_df.index.map(str.lower)
         f_df
```

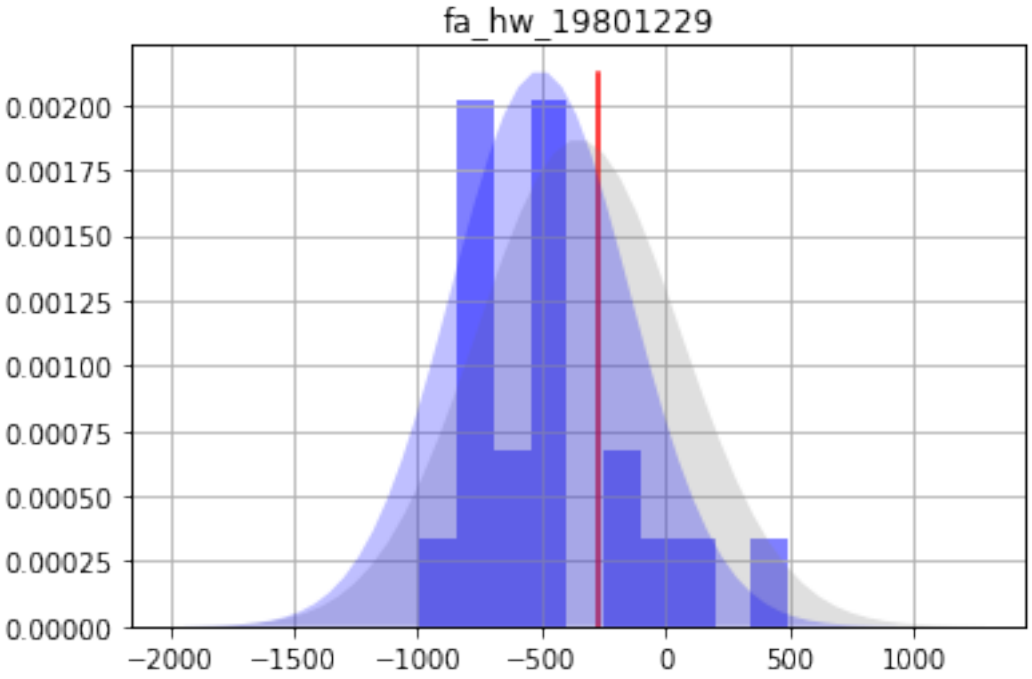
Out[27]:

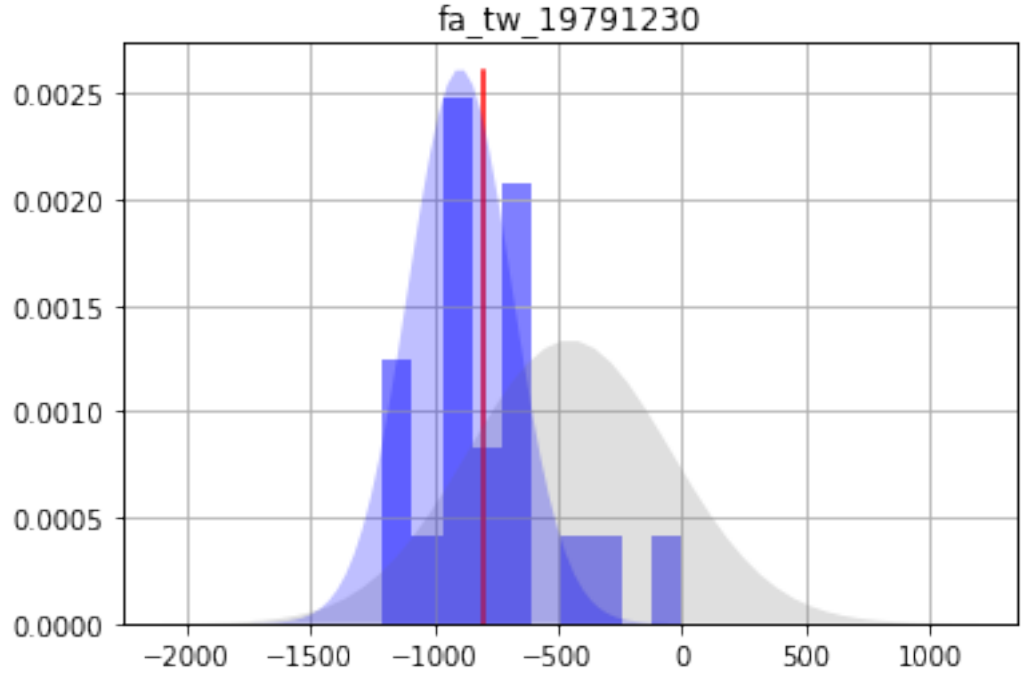
	prior_mean	prior_stdev	prior_lower_bound	prior_upper_bound	post_mean	post_stdev	post_lower_bound	post_upper_bound
name								
fa_hw_19791230	-977.2390	295.32800	-1567.8900	-386.5840	-1367.8100	265.661000	-1899.1300	-836.4890
fa_hw_19801229	-351.2160	409.77000	-1170.7600	468.3240	-506.4780	359.159000	-1224.7900	211.8400
fa_tw_19791230	-453.0330	409.35100	-1271.7400	365.6690	-894.7830	208.899000	-1312.5800	-476.9850
fa_tw_19801229	108.9600	506.73200	-904.5040	1122.4200	-164.6660	296.729000	-758.1240	428.7920
hds_00_013_002_000	39.6102	3.96314	31.6840	47.5365	44.8896	0.516271	43.8571	45.9221
hds_00_013_002_001	38.3838	4.05782	30.2681	46.4994	41.0046	0.814117	39.3764	42.6328
part_status	2.0000	0.00000	2.0000	2.0000	2.0000	0.000000	2.0000	2.0000
part_time	907.7020	570.98600	-234.2690	2049.6700	1778.7500	448.210000	882.3280	2675.1700

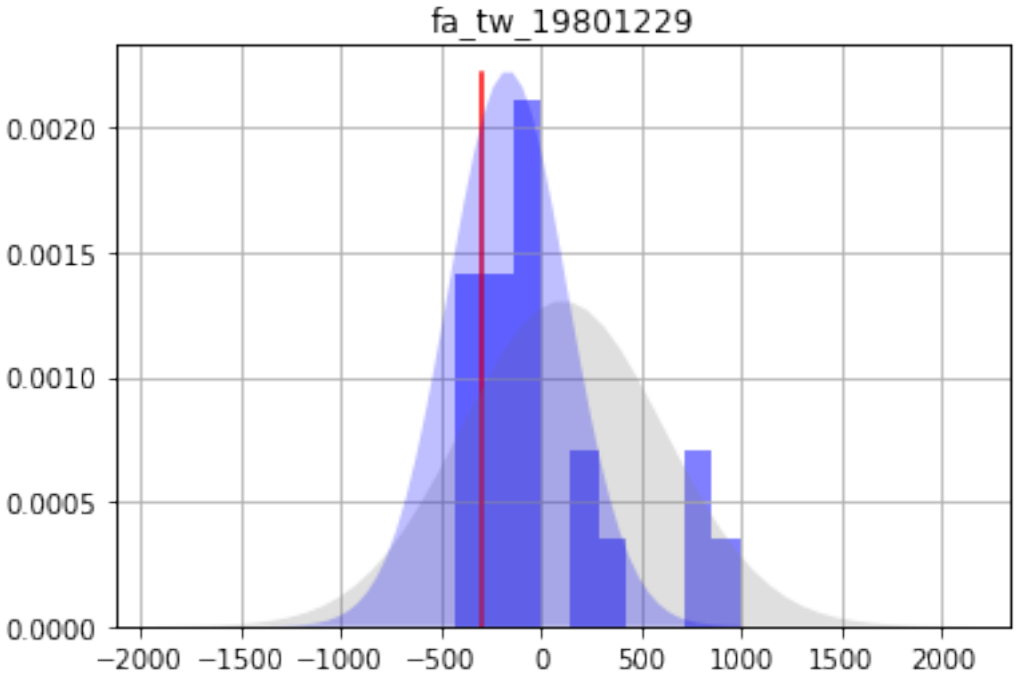
```
In [28]: obs = pst.observation_data
         fnames = pst.pestpp_options["forecasts"].split(",")
         for forecast in fnames:
             ax = plt.subplot(111)
             oe_pt.loc[:,forecast].hist(ax=ax,color="b",alpha=0.5,normed=True)
             ax.plot([obs.loc[forecast,"obsval"],obs.loc[forecast,"obsval"]],ax.get_ylim(),"r"
             axt = plt.twinx()
```

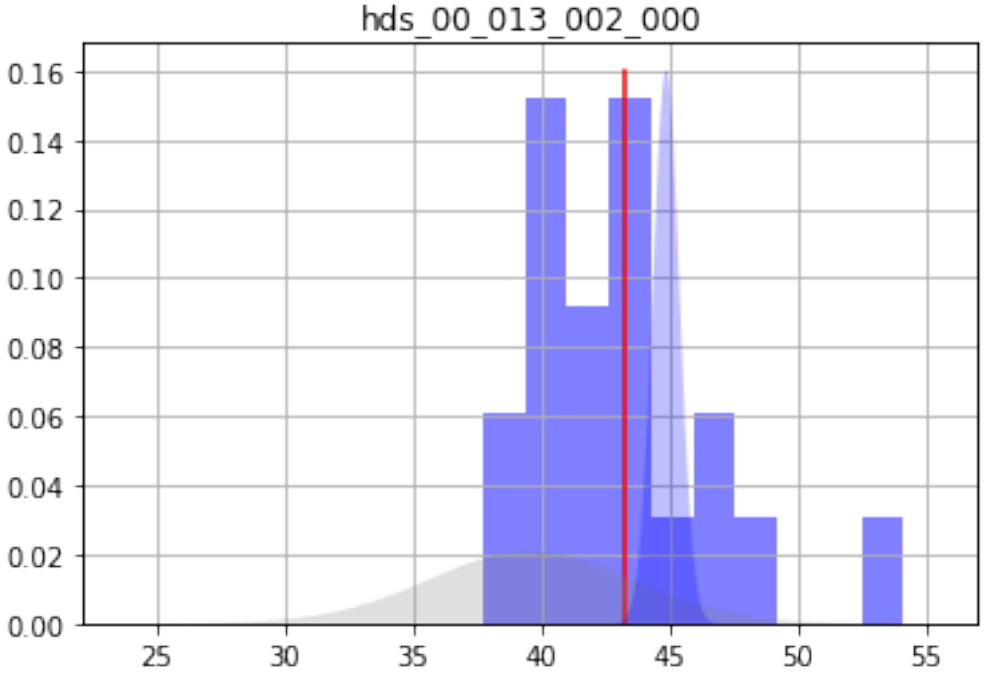
```
x,y = pyemu.plot_utils.gaussian_distribution(f_df.loc[forecast,"prior_mean"],f_df
axt.fill_between(x,0,y,facecolor="0.5",alpha=0.25)
x,y = pyemu.plot_utils.gaussian_distribution(f_df.loc[forecast,"post_mean"],f_df.
axt.fill_between(x,0,y,facecolor="b",alpha=0.25)
axt.set_ylim(0,axt.get_ylim()[1])
axt.set_yticks([])
ax.set_title(forecast)
plt.show()
```

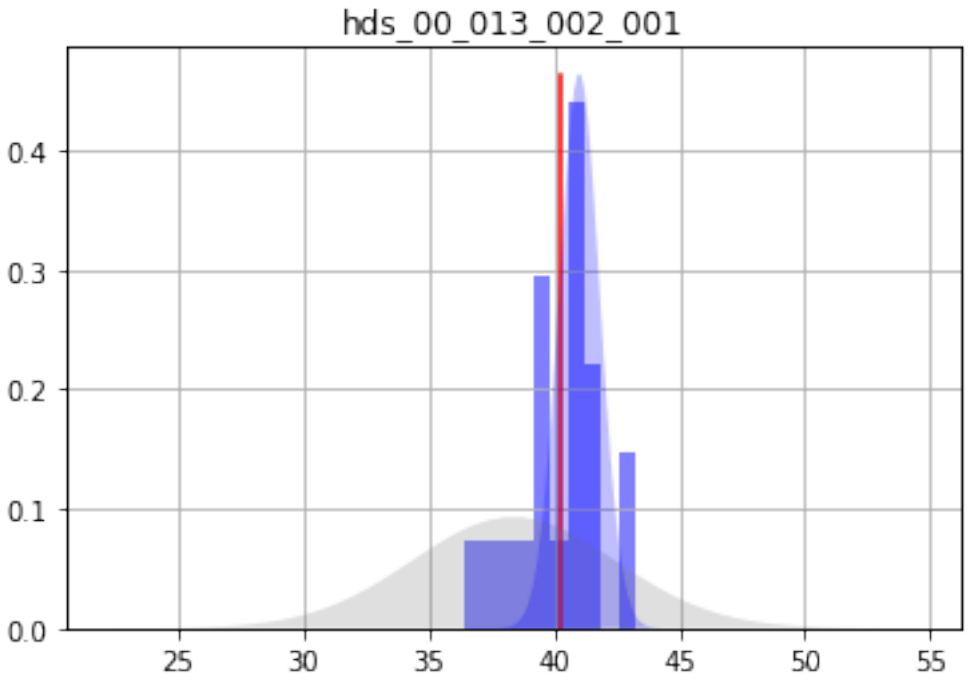


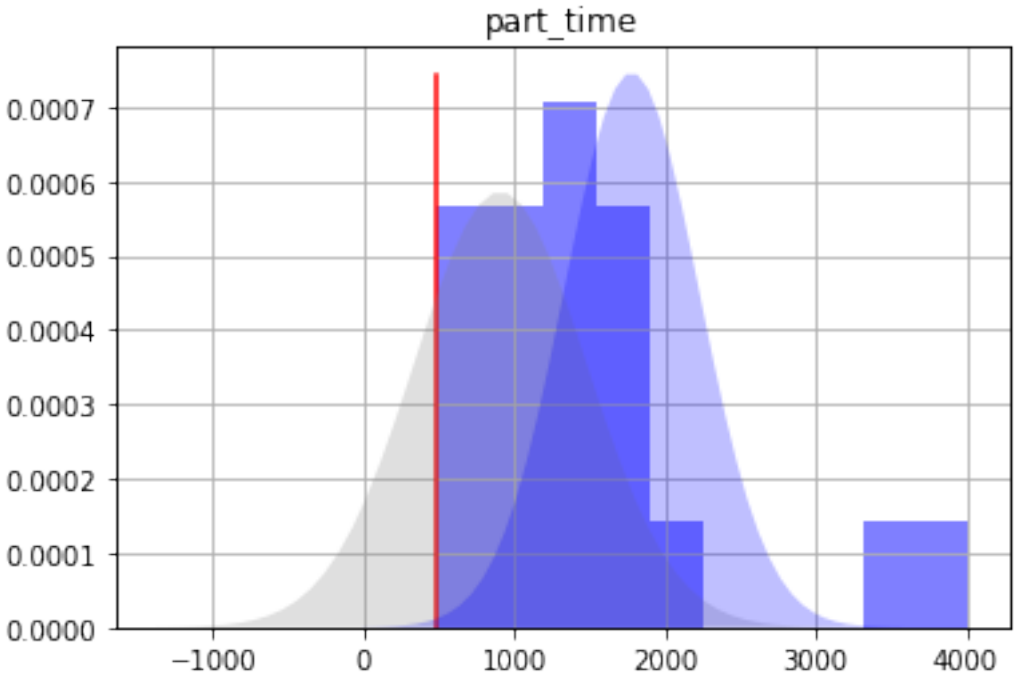

fa_hw_19801229
0.00200
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0.00000
−2000
−1500
−1000
−500
0
500
1000

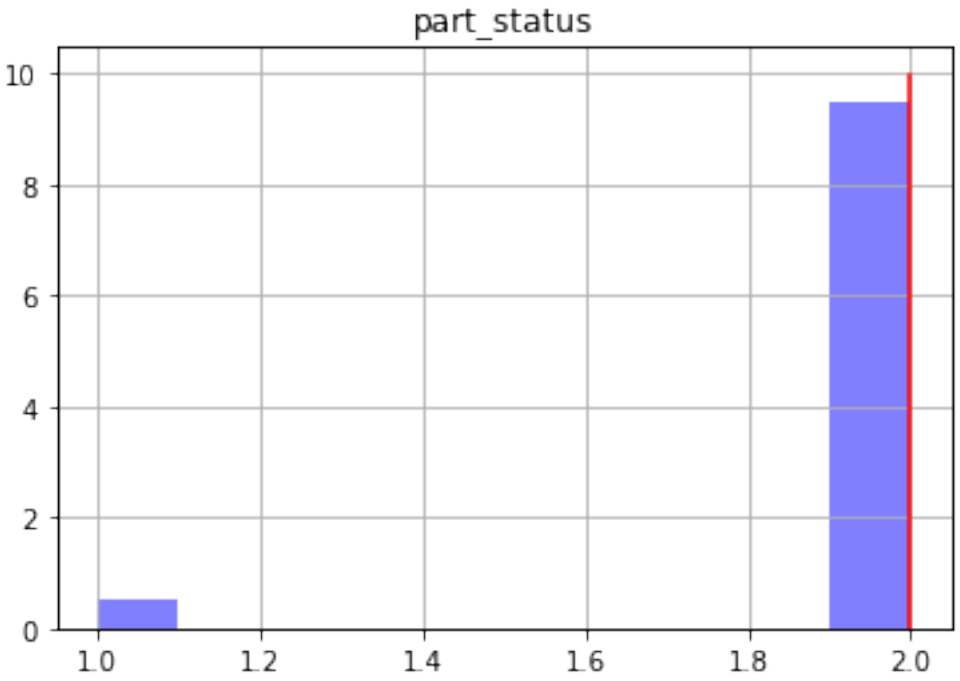

fa_tw_19791230
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
−2000
−1500
−1000
−500
0
500
1000


fa_tw_19801229
0.0020
0.0015
0.0010
0.0005
0.0000
−2000
−1500
−1000
−500
0
500
1000
1500
2000


hds_00_013_002_000
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
25
30
35
40
45
50
55


hds_00_013_002_001
0.4
0.3
0.2
0.1
0.0
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