pestpp-opt

May 2, 2019

1 Run PESTPP-OPT

In this notebook we will setup and solve a mgmt optimization problem around how much groundwater can be pumped while maintaining sw-gw exchange

In [1]:
```
import os
import shutil
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
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_opt"
```

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

Out[3]:

	type	transform	count	initial value \
cn_hk6	cn_hk6	log	1	0
cn_hk7	cn_hk7	log	1	0
cn_hk8	cn_hk8	log	1	0
cn_prsity6	cn_prsity6	log	1	0
cn_prsity7	cn_prsity7	log	1	0
cn_prsity8	cn_prsity8	log	1	0
cn_rech4	cn_rech4	log	1	0
cn_rech5	cn_rech5	log	1	-0.39794
cn_ss6	cn_ss6	log	1	0
cn_ss7	cn_ss7	log	1	0
cn_ss8	cn_ss8	log	1	0
cn_strt6	cn_strt6	log	1	0
cn_strt7	cn_strt7	log	1	0
cn_strt8	cn_strt8	log	1	0

cn_sy6	cn_sy6	log	1	0
cn_sy7	cn_sy7	log	1	0
cn_sy8	cn_sy8	log	1	0
cn_vka6	cn_vka6	log	1	0
cn_vka7	cn_vka7	log	1	0
cn_vka8	cn_vka8	log	1	0
drncond_k00	drncond_k00	log	10	0
flow	flow	log	1	0
gr_hk3	gr_hk3	log	705	0
gr_hk4	gr_hk4	log	705	0
gr_hk5	gr_hk5	log	705	0
gr_prsity3	gr_prsity3	log	705	0
gr_prsity4	gr_prsity4	log	705	0
gr_prsity5	gr_prsity5	log	705	0
gr_rech2	gr_rech2	log	705	0
gr_rech3	gr_rech3	log	705	0
...	...	...	...	...
gr_strt5	gr_strt5	log	705	0
gr_sy3	gr_sy3	log	705	0
gr_sy4	gr_sy4	log	705	0
gr_sy5	gr_sy5	log	705	0
gr_vka3	gr_vka3	log	705	0
gr_vka4	gr_vka4	log	705	0
gr_vka5	gr_vka5	log	705	0
pp_hk0	pp_hk0	log	32	0
pp_hk1	pp_hk1	log	32	0
pp_hk2	pp_hk2	log	32	0
pp_prsity0	pp_prsity0	log	32	0
pp_prsity1	pp_prsity1	log	32	0
pp_prsity2	pp_prsity2	log	32	0
pp_rech0	pp_rech0	log	32	0
pp_rech1	pp_rech1	log	32	0
pp_ss0	pp_ss0	log	32	0
pp_ss1	pp_ss1	log	32	0
pp_ss2	pp_ss2	log	32	0
pp_strt0	pp_strt0	log	32	0
pp_strt1	pp_strt1	log	32	0
pp_strt2	pp_strt2	log	32	0
pp_sy0	pp_sy0	log	32	0
pp_sy1	pp_sy1	log	32	0
pp_sy2	pp_sy2	log	32	0
pp_vka0	pp_vka0	log	32	0
pp_vka1	pp_vka1	log	32	0
pp_vka2	pp_vka2	log	32	0
strk	strk	log	40	0
welflux	welflux	log	2	0 to 0.176091
welflux_k02	welflux_k02	log	6	0

	upper bound	lower bound	standard deviation
cn_hk6	1	-1	0.5
cn_hk7	1	-1	0.5
cn_hk8	1	-1	0.5
cn_prsity6	0	-1	0.25
cn_prsity7	0	-1	0.25
cn_prsity8	0	-1	0.25
cn_rech4	0.0791812	-0.09691	0.0440228
cn_rech5	-0.09691	-1	0.225772
cn_ss6	1	-1	0.5
cn_ss7	1	-1	0.5
cn_ss8	1	-1	0.5
cn_strt6	0.0211893	-0.0222764	0.0108664
cn_strt7	0.0211893	-0.0222764	0.0108664
cn_strt8	0.0211893	-0.0222764	0.0108664
cn_sy6	0.243038	-0.60206	0.211275
cn_sy7	0.243038	-0.60206	0.211275
cn_sy8	0.243038	-0.60206	0.211275
cn_vka6	1	-1	0.5
cn_vka7	1	-1	0.5
cn_vka8	1	-1	0.5
drncond_k00	1	-1	0.5
flow	0.09691	-0.124939	0.0554622
gr_hk3	1	-1	0.5
gr_hk4	1	-1	0.5
gr_hk5	1	-1	0.5
gr_prsity3	0	-1	0.25
gr_prsity4	0	-1	0.25
gr_prsity5	0	-1	0.25
gr_rech2	0.0413927	-0.0457575	0.0217875
gr_rech3	0.0413927	-0.0457575	0.0217875
...	...	...	...
gr_strt5	0.0211893	-0.0222764	0.0108664
gr_sy3	0.243038	-0.60206	0.211275
gr_sy4	0.243038	-0.60206	0.211275
gr_sy5	0.243038	-0.60206	0.211275
gr_vka3	1	-1	0.5
gr_vka4	1	-1	0.5
gr_vka5	1	-1	0.5
pp_hk0	1	-1	0.5
pp_hk1	1	-1	0.5
pp_hk2	1	-1	0.5
pp_prsity0	0	-1	0.25
pp_prsity1	0	-1	0.25
pp_prsity2	0	-1	0.25
pp_rech0	0.0413927	-0.0457575	0.0217875
pp_rech1	0.0413927	-0.0457575	0.0217875
pp_ss0	1	-1	0.5

```
pp_ss1                        1              -1                  0.5
pp_ss2                        1              -1                  0.5
pp_strt0              0.0211893      -0.0222764            0.0108664
pp_strt1              0.0211893      -0.0222764            0.0108664
pp_strt2              0.0211893      -0.0222764            0.0108664
pp_sy0                 0.243038        -0.60206             0.211275
pp_sy1                 0.243038        -0.60206             0.211275
pp_sy2                 0.243038        -0.60206             0.211275
pp_vka0                       1              -1                  0.5
pp_vka1                       1              -1                  0.5
pp_vka2                       1              -1                  0.5
strk                          2              -2                    1
welflux      0.176091 to 0.30103  -0.30103 to      0  0.0752575 to 0.11928
welflux_k02                   1              -1                  0.5

[65 rows x 7 columns]
```

define our decision varible group and also set some ++args

In [4]:
```
pst.pestpp_options = {}
#dvg = ["welflux_k02","welflux"]
dvg = ["welflux_k02"]
pst.pestpp_options["opt_dec_var_groups"] = dvg
pst.pestpp_options["opt_direction"] = "max"
```

For the first run, we wont use chance constraints, so just fix all non-decision-variable parameter. We also need to set some realistic bounds on the `welflux` multiplier decision variables. Finally, we need to specify a larger derivative increment for the decision varible group

In [5]:
```
par = pst.parameter_data
par.loc[:,"partrans"] = "fixed"

#turn off pumping in the scenario
par.loc["welflux_001","parlbnd"] = 0.0
par.loc["welflux_001","parval1"] = 0.0
dvg_pars = par.loc[par.pargp.apply(lambda x: x in dvg),"parnme"]
par.loc[dvg_pars,"partrans"] = "none"
par.loc[dvg_pars,"parlbnd"] = 0.0
par.loc[dvg_pars,"parubnd"] = 2.0
par.loc[dvg_pars,"parval1"] = 1.0

pst.rectify_pgroups()
pst.parameter_groups.loc[dvg,"inctyp"] = "absolute"
pst.parameter_groups.loc[dvg,"inctyp"] = "absolute"
pst.parameter_groups.loc[dvg,"derinc"] = 0.25

pst.parameter_groups.loc[dvg,:]
```

Out[5]:
```
                pargpnme   inctyp  derinc  derinclb  forcen  derincmul  \
pargpnme
```

```
welflux_k02  welflux_k02  absolute    0.25       0.0  switch        2.0

                 dermthd  splitthresh  splitreldiff splitaction  extra
pargpnme
welflux_k02  parabolic      0.00001          0.5     smaller    NaN
```

1.0.1 define constraints

model-based constraints are identified in pestpp-opt by an obs group that starts with "less_than" or "greater_than" and a weight greater than zero. So first, we turn off all of the weights and get names for the sw-gw exchange observations

In [6]:
```python
obs = pst.observation_data
obs.loc[:,"weight"] = 0.0
swgw_hist = obs.loc[obs.obsnme.apply(lambda x: "fa" in x and( "hw" in x or "tw" in x))
obs.loc[swgw_hist,:]
```

Out[6]:

obsnme	obsnme	obsval	weight	obgnme	extra
fa_hw_19791230	fa_hw_19791230	-1265.70460	0.0	flaqx	NaN
fa_hw_19801229	fa_hw_19801229	-555.44350	0.0	flaqx	NaN
fa_tw_19791230	fa_tw_19791230	-257.45412	0.0	flaqx	NaN
fa_tw_19801229	fa_tw_19801229	210.09800	0.0	flaqx	NaN

We need to change the obs group (`obgnme`) so that `pestpp-opt` will recognize these two model outputs as constraints. The `obsval` becomes the RHS of the constraint. We also need to set a lower bound constraint on the total abstraction rate (good thing we included all those list file budget components as observations!)

In [7]:
```python
obs.loc[swgw_hist,"obgnme"] = "less_than"
obs.loc[swgw_hist,"weight"] = 1.0

obs.loc[swgw_hist,"obsval"] = -300

tot_abs_rate = ["flx_wells_19791230"]#,"flx_wells_19801229"]
obs.loc[tot_abs_rate,"obgnme"] = "less_than"
obs.loc[tot_abs_rate,"weight"] = 1.0
obs.loc[tot_abs_rate,"obsval"] = -600.0
pst.less_than_obs_constraints
```

Out[7]:
```
obsnme
fa_hw_19791230            fa_hw_19791230
fa_hw_19801229            fa_hw_19801229
fa_tw_19791230            fa_tw_19791230
fa_tw_19801229            fa_tw_19801229
flx_wells_19791230    flx_wells_19791230
Name: obsnme, dtype: object
```

In [8]:
```python
pst.control_data.noptmax = 1
pst.write(os.path.join(t_d,"freyberg_opt.pst"))
```

```
In [9]: pyemu.os_utils.start_slaves(t_d,"pestpp-opt","freyberg_opt.pst",num_slaves=10,master_di
```

Let's load and inspect the response matrix

```
In [10]: jco = pyemu.Jco.from_binary(os.path.join(m_d,"freyberg_opt.1.jcb")).to_dataframe().loc
         jco
```

```
Out[10]:                     wf0200090016  wf0200110013  wf0200200014  wf0200260010  \
         fa_hw_19791230         137.57200     126.32400      46.30000      21.90800
         fa_hw_19801229          22.58400      28.65600      12.03600      12.29200
         fa_tw_19791230           6.50728      14.53516      93.28136      92.42320
         fa_tw_19801229           4.10836       7.60104      15.29948      30.88604
         flx_wells_19791230    -150.00000    -150.00000    -150.00000    -150.00000

                             wf0200290006  wf0200340012
         fa_hw_19791230          18.12000        4.8320
         fa_hw_19801229          13.12800        3.3560
         fa_tw_19791230          71.84608       82.9612
         fa_tw_19801229          34.79872       17.5232
         flx_wells_19791230    -150.00000     -150.0000
```

Let's also load the optimal decision variable values:

```
In [11]: par_df = pyemu.pst_utils.read_parfile(os.path.join(m_d,"freyberg_opt.1.par"))
         print(par_df.loc[dvg_pars,"parval1"].sum())
         par_df.loc[dvg_pars,:]
```

```
8.1332977617072
```

```
Out[11]:                     parnme   parval1  scale  offset
         parnme
         wf0200090016  wf0200090016  2.000000    1.0     0.0
         wf0200110013  wf0200110013  2.000000    1.0     0.0
         wf0200200014  wf0200200014  2.000000    1.0     0.0
         wf0200260010  wf0200260010  0.133298    1.0     0.0
         wf0200290006  wf0200290006  0.000000    1.0     0.0
         wf0200340012  wf0200340012  2.000000    1.0     0.0
```

The sum of these values is the optimal objective function value. However, since these are just mulitpliers on the pumping rate, this number isnt too meaningful. Instead, lets look at the residuals file

```
In [12]: pst = pyemu.Pst(os.path.join(m_d,"freyberg_opt.pst"),resfile=os.path.join(m_d,"freybe
         pst.res.loc[pst.nnz_obs_names,:]
```

```
Out[12]:                                  name      group  measured   modelled  \
         name
         fa_hw_19791230          fa_hw_19791230  less_than    -300.0  -699.3735
```

```
fa_hw_19801229          fa_hw_19801229  less_than    -300.0  -714.4580
fa_tw_19791230          fa_tw_19791230  less_than    -300.0  -407.7249
fa_tw_19801229          fa_tw_19801229  less_than    -300.0  -299.7868
flx_wells_19791230  flx_wells_19791230  less_than    -600.0 -1219.9948

                    residual  weight
name
fa_hw_19791230      399.3735     1.0
fa_hw_19801229      414.4580     1.0
fa_tw_19791230      107.7249     1.0
fa_tw_19801229       -0.2132     1.0
flx_wells_19791230  619.9948     1.0
```

Sweet as! lots of room in the optimization problem. The bounding constraint is the one closest to its RHS

1.0.2 Opt under uncertainty part 1: FOSM chance constraints

To activate the chance constraint process, we need to specific a risk != 0.5

In [13]:
```
pst.pestpp_options["opt_risk"] = 0.4
```

For the FOSM-based chance constraints, we also need to have at least one adjustable non-dec-var parameter so that we can propogate parameter uncertainty to model-based constraints (this can also be posterior FOSM is non-constraint, non-zero-weight observations are specified). For this simple demo, lets just use the constant multiplier parameters in the prior uncertainty stance:

In [14]:
```
cn_pars = par.loc[par.pargp.apply(lambda x: "cn" in x),"parnme"]
cn_pars
```

Out[14]:
```
parnme
hk6_cn            hk6_cn
hk7_cn            hk7_cn
hk8_cn            hk8_cn
prsity6_cn    prsity6_cn
prsity7_cn    prsity7_cn
prsity8_cn    prsity8_cn
rech4_cn        rech4_cn
rech5_cn        rech5_cn
ss6_cn            ss6_cn
ss7_cn            ss7_cn
ss8_cn            ss8_cn
strt6_cn        strt6_cn
strt7_cn        strt7_cn
strt8_cn        strt8_cn
sy6_cn            sy6_cn
sy7_cn            sy7_cn
sy8_cn            sy8_cn
vka6_cn          vka6_cn
```

```
vka7_cn          vka7_cn
vka8_cn          vka8_cn
Name: parnme, dtype: object
```

In [15]:
```
par = pst.parameter_data
par.loc[cn_pars,"partrans"] = "log"
pst.control_data.noptmax = 1
pst.write(os.path.join(t_d,"freyberg_opt_uu1.pst"))
pst.npar_adj
```

Out[15]: 26

In [16]:
```
pyemu.os_utils.start_slaves(t_d,"pestpp-opt","freyberg_opt_uu1.pst",num_slaves=20,mas
```

In [17]:
```
pst = pyemu.Pst(os.path.join(m_d,"freyberg_opt_uu1.pst"),resfile=os.path.join(m_d,"fr
pst.res.loc[pst.nnz_obs_names,:]
```

Out[17]:

name	name	group	measured	modelled	\
fa_hw_19791230	fa_hw_19791230	less_than	-300.0	-666.13442	
fa_hw_19801229	fa_hw_19801229	less_than	-300.0	-682.60800	
fa_tw_19791230	fa_tw_19791230	less_than	-300.0	-223.47050	
fa_tw_19801229	fa_tw_19801229	less_than	-300.0	-208.37540	
flx_wells_19791230	flx_wells_19791230	less_than	-600.0	-1586.33800	

name	residual	weight
fa_hw_19791230	366.13442	1.0
fa_hw_19801229	382.60800	1.0
fa_tw_19791230	-76.52950	1.0
fa_tw_19801229	-91.62460	1.0
flx_wells_19791230	986.33800	1.0

In [18]:
```
par_df = pyemu.pst_utils.read_parfile(os.path.join(m_d,"freyberg_opt_uu1.1.par"))
print(par_df.loc[dvg_pars,"parval1"].sum())
par_df.loc[dvg_pars,:]
```

```
10.575587155980312
```

Out[18]:

parnme	parnme	parval1	scale	offset
wf0200090016	wf0200090016	2.000000	1.0	0.0
wf0200110013	wf0200110013	2.000000	1.0	0.0
wf0200200014	wf0200200014	1.481006	1.0	0.0
wf0200260010	wf0200260010	1.094581	1.0	0.0
wf0200290006	wf0200290006	2.000000	1.0	0.0
wf0200340012	wf0200340012	2.000000	1.0	0.0

1.0.3 Opt under uncertainty part 2: ensemble-based chance constraints

PESTPP-OPT can also skip the FOSM calculations if users specify model-based constraint weights as standard deviations. These can be derived from existing ensembles (oh snap!)

```
In [19]: obs_df = pd.read_csv(os.path.join("master_prior_sweep","sweep_out.csv"),index_col=0)
         obs_df = obs_df.loc[obs_df.failed_flag==0,:]
```

```
In [20]: std = obs_df.std().loc[pst.nnz_obs_names]
         std
```

```
Out[20]: fa_hw_19791230        399.204426
         fa_hw_19801229        564.029274
         fa_tw_19791230        524.756534
         fa_tw_19801229        608.379823
         flx_wells_19791230    677.978902
         dtype: float64
```

Wait! Something is wrong here: The cumulative well flux constraint is not uncertain - it is just a summation of the specified flux. So lets give it a crazy small weight, implying it has a tiny uncertainty

```
In [21]: std["flx_wells_19791230"] = 1.0e-10
         std
```

```
Out[21]: fa_hw_19791230        3.992044e+02
         fa_hw_19801229        5.640293e+02
         fa_tw_19791230        5.247565e+02
         fa_tw_19801229        6.083798e+02
         flx_wells_19791230    1.000000e-10
         dtype: float64
```

```
In [22]: pst.observation_data.loc[pst.nnz_obs_names,"weight"] = std.loc[pst.nnz_obs_names]
         pst.pestpp_options["opt_std_weights"] = True
         pst.write(os.path.join(t_d,"freyberg_opt_uu2.pst"))
```

```
In [23]: pyemu.os_utils.start_slaves(t_d,"pestpp-opt","freyberg_opt_uu2.pst",num_slaves=10,mas
```

```
In [24]: par_df = pyemu.pst_utils.read_parfile(os.path.join(m_d,"freyberg_opt_uu2.1.par"))
         print(par_df.loc[dvg_pars,"parval1"].sum())
         par_df.loc[dvg_pars,:]
```

```
11.189845992374467
```

```
Out[24]:                     parnme   parval1  scale  offset
         parnme
         wf0200090016  wf0200090016  2.000000    1.0     0.0
         wf0200110013  wf0200110013  2.000000    1.0     0.0
         wf0200200014  wf0200200014  1.189846    1.0     0.0
         wf0200260010  wf0200260010  2.000000    1.0     0.0
         wf0200290006  wf0200290006  2.000000    1.0     0.0
         wf0200340012  wf0200340012  2.000000    1.0     0.0
```

1.0.4 Super secret mode

turns out, if the opt problem is truely linear, we can reuse results of a previous PESTPP-OPT run to modify lots of the pieces of the optimization problem and resolve without running the model even once! WAT!?

In [25]:
```
shutil.copy2(os.path.join(m_d,"freyberg_opt_uu2.1.jcb"),os.path.join(m_d,"restart.jcb
shutil.copy2(os.path.join(m_d,"freyberg_opt_uu2.jcb.1.rei"),os.path.join(m_d,"restart

pst.pestpp_options["base_jacobian"] = "restart.jcb"
pst.pestpp_options["hotstart_resfile"] = "restart.rei"
pst.pestpp_options["opt_skip_final"] = True
pst.write(os.path.join(m_d,"freyberg_opt_restart.pst"))
```

In [26]:
```
pyemu.os_utils.run("pestpp-opt freyberg_opt_restart.pst",cwd=m_d)
```

In [27]:
```
par_df = pyemu.pst_utils.read_parfile(os.path.join(m_d,"freyberg_opt_restart.1.par"))
print(par_df.loc[dvg_pars,"parval1"].sum())
par_df.loc[dvg_pars,:]
```

```
11.189845992374467
```

Out[27]:

parnme	parnme	parval1	scale	offset
wf0200090016	wf0200090016	2.000000	1.0	0.0
wf0200110013	wf0200110013	2.000000	1.0	0.0
wf0200200014	wf0200200014	1.189846	1.0	0.0
wf0200260010	wf0200260010	2.000000	1.0	0.0
wf0200290006	wf0200290006	2.000000	1.0	0.0
wf0200340012	wf0200340012	2.000000	1.0	0.0

Oh snap! that means we can do all sort of kewl optimization testing really really fast. . . .

In [28]:
```
pst.pestpp_options["opt_risk"] = 0.54
pst.write(os.path.join(m_d,"freyberg_opt_restart.pst"))
pyemu.os_utils.run("pestpp-opt freyberg_opt_restart.pst",cwd=m_d)
par_df = pyemu.pst_utils.read_parfile(os.path.join(m_d,"freyberg_opt_restart.1.par"))
print(par_df.loc[dvg_pars,"parval1"].sum())
par_df.loc[dvg_pars,:]
```

```
4.566052122715744
```

Out[28]:

parnme	parnme	parval1	scale	offset
wf0200090016	wf0200090016	2.000000	1.0	0.0
wf0200110013	wf0200110013	2.000000	1.0	0.0
wf0200200014	wf0200200014	0.566052	1.0	0.0
wf0200260010	wf0200260010	0.000000	1.0	0.0
wf0200290006	wf0200290006	0.000000	1.0	0.0
wf0200340012	wf0200340012	0.000000	1.0	0.0

Lets use the functionality to evaluate how our OUU problem changes if we use posterior standard deviations:

In [29]:
```
obs_df = pd.read_csv(os.path.join("master_ies","freyberg_ies.3.obs.csv"),index_col=0)
std = obs_df.std().loc[pst.nnz_obs_names]
std["flx_wells_19791230"] = 1.0e-10
std
```

Out[29]:
```
fa_hw_19791230        2.017401e+02
fa_hw_19801229        3.592778e+02
fa_tw_19791230        2.309447e+02
fa_tw_19801229        2.757140e+02
flx_wells_19791230    1.000000e-10
dtype: float64
```

In [30]:
```
pst.observation_data.loc[pst.nnz_obs_names,"weight"] = std.loc[pst.nnz_obs_names]
pst.observation_data.loc[pst.nnz_obs_names,"weight"]
```

Out[30]:
```
obsnme
fa_hw_19791230        2.017401e+02
fa_hw_19801229        3.592778e+02
fa_tw_19791230        2.309447e+02
fa_tw_19801229        2.757140e+02
flx_wells_19791230    1.000000e-10
Name: weight, dtype: float64
```

In [31]:
```
pst.write(os.path.join(m_d,"freyberg_opt_restart.pst"))
pyemu.os_utils.run("pestpp-opt freyberg_opt_restart.pst",cwd=m_d)
par_df = pyemu.pst_utils.read_parfile(os.path.join(m_d,"freyberg_opt_restart.1.par"))
print(par_df.loc[dvg_pars,"parval1"].sum())
par_df.loc[dvg_pars,:]
```

```
6.654694834705468
```

Out[31]:
```
                  parnme   parval1  scale  offset
parnme
wf0200090016  wf0200090016  2.000000    1.0     0.0
wf0200110013  wf0200110013  2.000000    1.0     0.0
wf0200200014  wf0200200014  2.000000    1.0     0.0
wf0200260010  wf0200260010  0.000000    1.0     0.0
wf0200290006  wf0200290006  0.000000    1.0     0.0
wf0200340012  wf0200340012  0.654695    1.0     0.0
```

In [32]:
```
res_df = pyemu.pst_utils.read_resfile(os.path.join(m_d,"freyberg_opt_restart.jcb.1.re
res_df
```

Out[32]:
```
                                   name      group  measured   modelled  \
name
```

```
fa_hw_19791230          fa_hw_19791230  less_than    -300.0 -977.23900
fa_hw_19801229          fa_hw_19801229  less_than    -300.0 -757.44600
fa_tw_19791230          fa_tw_19791230  less_than    -300.0 -453.03310
fa_tw_19801229          fa_tw_19801229  less_than    -300.0 -282.96436
flx_wells_19791230  flx_wells_19791230  less_than    -600.0 -900.00000

                     residual        weight
name
fa_hw_19791230      677.23900  2.017401e+02
fa_hw_19801229      457.44600  3.592778e+02
fa_tw_19791230      153.03310  2.309447e+02
fa_tw_19801229      -17.03564  2.757140e+02
flx_wells_19791230  300.00000  1.000000e-10
```