pestpp-glm_part1

July 1, 2019

1 PESTPP-GLM Part 1

In this notebook, we will run PESTPP-GLM to generate a jco matrix and stop - this is to support data worth testing

```
In [1]: %matplotlib inline
        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_csiro/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	upper bound \
pp_prsity2	pp_prsity2	log	32	0	0.176091
gr_sy3	gr_sy3	log	705	0	0.243038
pp_rech0	pp_rech0	log	32	0	0.0413927
cn_hk6	cn_hk6	log	1	0	1
cn_sy7	cn_sy7	log	1	0	0.243038
gr_ss4	gr_ss4	log	705	0	1
cn_sy6	cn_sy6	log	1	0	0.243038
cn_hk8	cn_hk8	log	1	0	1

pp_vka1	pp_vka1	log	32	0	1
pp_sy0	pp_sy0	log	32	0	0.243038
cn_prsity8	cn_prsity8	log	1	0	0.176091
cn_ss6	cn_ss6	log	1	0	1
welflux_k02	welflux_k02	log	6	0	1
cn_rech5	cn_rech5	log	1	0	0.0413927
cn_vka8	cn_vka8	log	1	0	1
cn_rech4	cn_rech4	log	1	0	0.0413927
cn_hk7	cn_hk7	log	1	0	1
gr_hk5	gr_hk5	log	705	0	1
gr_hk4	gr_hk4	log	705	0	1
pp_hk0	pp_hk0	log	32	0	1
gr_vka4	gr_vka4	log	705	0	1
pp_strt1	pp_strt1	log	32	0	0.0211893
cn_sy8	cn_sy8	log	1	0	0.243038
gr_vka5	gr_vka5	log	705	0	1
gr_prsity3	gr_prsity3	log	705	0	0.176091
gr_prsity4	gr_prsity4	log	705	0	0.176091
cn_vka7	cn_vka7	log	1	0	1
gr_rech2	gr_rech2	log	705	0	0.0413927
cn_ss7	cn_ss7	log	1	0	1
pp_hk2	pp_hk2	log	32	0	1
...	...	...	...	...	...
pp_strt0	pp_strt0	log	32	0	0.0211893
gr_hk3	gr_hk3	log	705	0	1
pp_ss0	pp_ss0	log	32	0	1
cn_prsity6	cn_prsity6	log	1	0	0.176091
gr_ss5	gr_ss5	log	705	0	1
gr_sy5	gr_sy5	log	705	0	0.243038
gr_strt3	gr_strt3	log	705	0	0.0211893
cn_ss8	cn_ss8	log	1	0	1
cn_strt6	cn_strt6	log	1	0	0.0211893
pp_ss2	pp_ss2	log	32	0	1
cn_strt8	cn_strt8	log	1	0	0.0211893
flow	flow	log	1	0	0.09691
gr_vka3	gr_vka3	log	705	0	1
gr_rech3	gr_rech3	log	705	0	0.0413927
gr_prsity5	gr_prsity5	log	705	0	0.176091
gr_strt5	gr_strt5	log	705	0	0.0211893
pp_prsity0	pp_prsity0	log	32	0	0.176091
pp_vka0	pp_vka0	log	32	0	1
pp_sy1	pp_sy1	log	32	0	0.243038
pp_hk1	pp_hk1	log	32	0	1
cn_prsity7	cn_prsity7	log	1	0	0.176091
cn_strt7	cn_strt7	log	1	0	0.0211893
gr_ss3	gr_ss3	log	705	0	1
pp_sy2	pp_sy2	log	32	0	0.243038
welflux	welflux	log	2	0	1

pp_prsity1	pp_prsity1	log	32	0	0.176091
drncond_k00	drncond_k00	log	10	0	1
pp_strt2	pp_strt2	log	32	0	0.0211893
pp_vka2	pp_vka2	log	32	0	1
gr_sy4	gr_sy4	log	705	0	0.243038

	lower bound	standard deviation
pp_prsity2	-0.30103	0.11928
gr_sy3	-0.60206	0.211275
pp_rech0	-0.0457575	0.0217875
cn_hk6	-1	0.5
cn_sy7	-0.60206	0.211275
gr_ss4	-1	0.5
cn_sy6	-0.60206	0.211275
cn_hk8	-1	0.5
pp_vka1	-1	0.5
pp_sy0	-0.60206	0.211275
cn_prsity8	-0.30103	0.11928
cn_ss6	-1	0.5
welflux_k02	-1	0.5
cn_rech5	-0.0457575	0.0217875
cn_vka8	-1	0.5
cn_rech4	-0.0457575	0.0217875
cn_hk7	-1	0.5
gr_hk5	-1	0.5
gr_hk4	-1	0.5
pp_hk0	-1	0.5
gr_vka4	-1	0.5
pp_strt1	-0.0222764	0.0108664
cn_sy8	-0.60206	0.211275
gr_vka5	-1	0.5
gr_prsity3	-0.30103	0.11928
gr_prsity4	-0.30103	0.11928
cn_vka7	-1	0.5
gr_rech2	-0.0457575	0.0217875
cn_ss7	-1	0.5
pp_hk2	-1	0.5
...	...	...
pp_strt0	-0.0222764	0.0108664
gr_hk3	-1	0.5
pp_ss0	-1	0.5
cn_prsity6	-0.30103	0.11928
gr_ss5	-1	0.5
gr_sy5	-0.60206	0.211275
gr_strt3	-0.0222764	0.0108664
cn_ss8	-1	0.5
cn_strt6	-0.0222764	0.0108664
pp_ss2	-1	0.5

```
cn_strt8     -0.0222764          0.0108664
flow          -0.124939          0.0554622
gr_vka3              -1                0.5
gr_rech3     -0.0457575          0.0217875
gr_prsity5     -0.30103            0.11928
gr_strt5     -0.0222764          0.0108664
pp_prsity0     -0.30103            0.11928
pp_vka0              -1                0.5
pp_sy1         -0.60206           0.211275
pp_hk1               -1                0.5
cn_prsity7     -0.30103            0.11928
cn_strt7     -0.0222764          0.0108664
gr_ss3               -1                0.5
pp_sy2         -0.60206           0.211275
welflux              -1                0.5
pp_prsity1     -0.30103            0.11928
drncond_k00          -1                0.5
pp_strt2     -0.0222764          0.0108664
pp_vka2              -1                0.5
gr_sy4         -0.60206           0.211275

[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]:
```
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]:
```
flow             1
cn_vka7          1
cn_strt8         1
cn_hk6           1
cn_sy6           1
cn_hk8           1
cn_prsity7       1
cn_strt7         1
cn_prsity8       1
cn_ss6           1
cn_prsity6       1
cn_vka6          1
cn_rech4         1
cn_hk7           1
cn_vka8          1
cn_strt6         1
cn_sy8           1
cn_ss8           1
cn_rech5         1
cn_ss7           1
cn_sy7           1
welflux          2
welflux_k02      6
drncond_k00     10
pp_prsity2      32
pp_sy1          32
pp_ss0          32
pp_ss2          32
pp_rech0        32
pp_vka2         32
pp_sy0          32
pp_strt0        32
pp_hk0          32
pp_strt1        32
pp_hk2          32
pp_rech1        32
pp_ss1          32
pp_prsity0      32
pp_vka1         32
pp_prsity1      32
pp_sy2          32
pp_vka0         32
pp_hk1          32
```

```
pp_strt2       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 = -1
         pst.write(os.path.join(t_d,"freyberg_pp.pst"))
```

```
noptmax:-1, npar_adj:527, nnz_obs:14
```

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

That is all we need for FOSM, so stop here and relax!