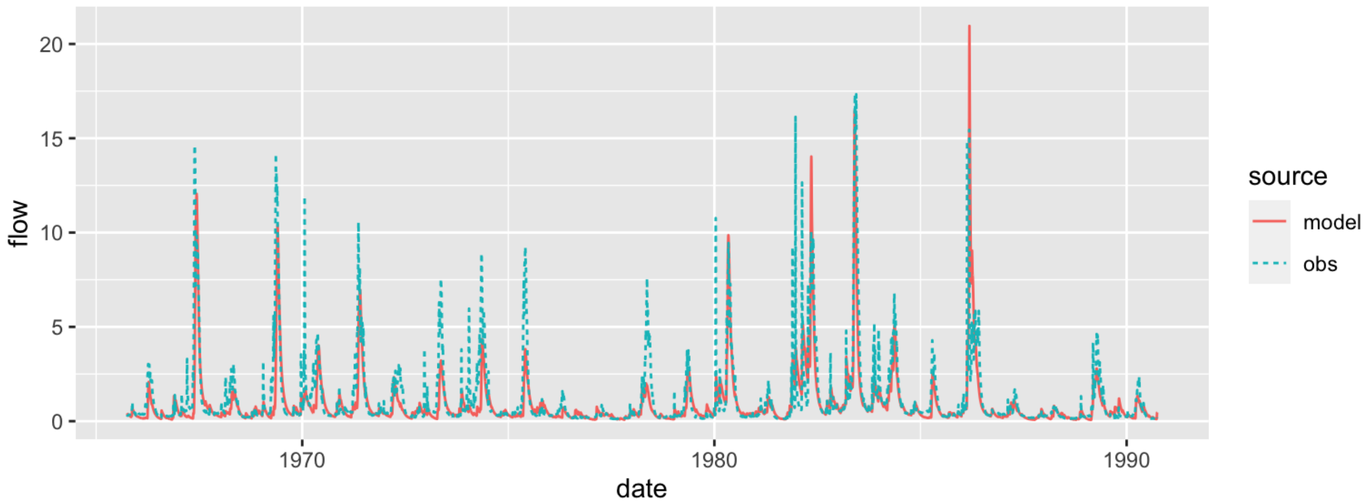
More on Calibration - Equifinality

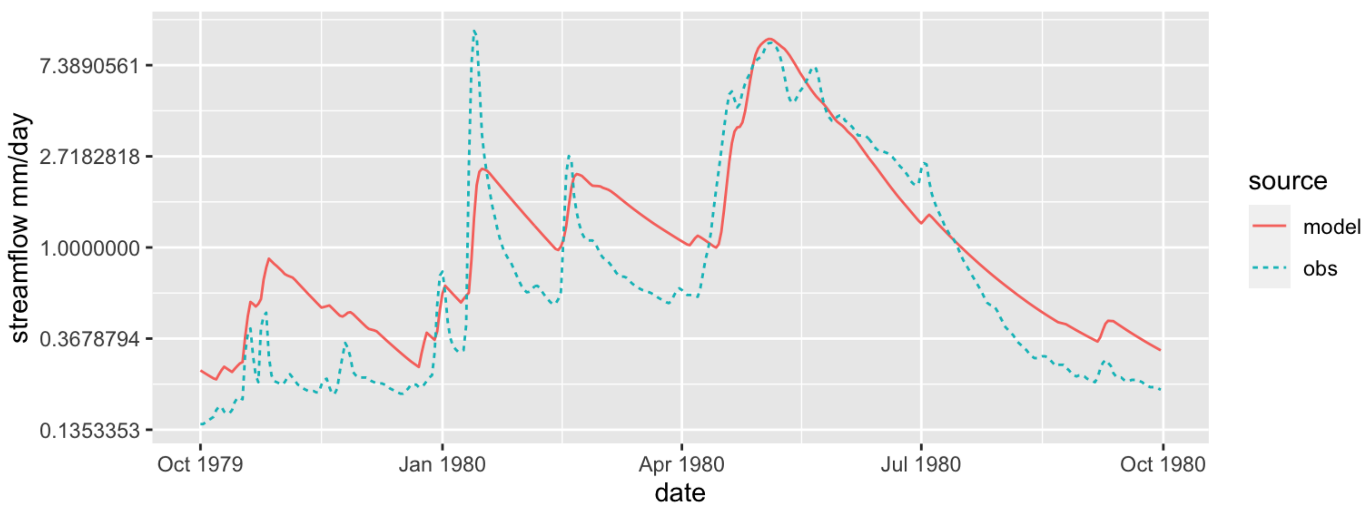
Review of Calibration / Evaluation So far

```
sager = read.table("../Data/sager.txt", header=T)
head(sager)
```

```
##       model       obs month day year   wy wyd
## 1 0.4238063 0.3358678    10   1 1965 1966   1
## 2 0.4133587 0.3208737    10   2 1965 1966   2
## 3 0.4032640 0.3058796    10   3 1965 1966   3
## 4 0.3935287 0.2968832    10   4 1965 1966   4
## 5 0.3841480 0.2968832    10   5 1965 1966   5
## 6 0.3751000 0.2968832    10   6 1965 1966   6
```

```
# add date
sager = sager %>% mutate(date = paste(day,month,year, sep="/"))
sager$date = as.Date(sager$date,"%d/%m/%Y")

# plot
sagerl = sager %>% pivot_longer(cols=c("model","obs"), names_to="source",
                                  values_to="flow")

# basic plot
p1=ggplot(sagerl, aes(date, flow, col=source, linetype=source))+geom_line()

p2=ggplot(subset(sagerl, wy==1980), aes(date, flow, col=source,
        linetype=source))+geom_line()+scale_y_continuous(trans="log")+labs(y="streamflow
        mm/day")

# look at it another way with 1:1 line
ggarrange(p1,p2, ncol=1)
```


flow
20
15
10
5
0
1970
1980
1990
date
source
model
obs


streamflow mm/day
7.3890561
2.7182818
1.0000000
0.3678794
0.1353353
Oct 1979
Jan 1980
Apr 1980
Jul 1980
Oct 1980
date
source
model
obs

Calibration

Calibration is picking parameter sets based on performance evaluation

Apply metrics over multiple outputs (generated by running across many parameters sets) - like we've done in our sensitivity analysis work

Example - a dataset where each column is a different model run for Sagehen Creek (using different parameters) - don't worry about the parameters for now

- sagerm.txt

Split-sample: split time period into * calibration time period (used to pick parameter sets) * validation time period (used to see how well chose paramter sets perform)

Read in data plot results for an example year (1974)

```
# multiple results - lets say we've run the model for multiple years,
#each column  is streamflow for a different parameter set
msage = read.table("../Data/sagerm.txt", header=T)

# keep track of number of simulations (e.g results for each parameter set)
# use as a column names
nsim = ncol(msage)
snames = sprintf("S%d",seq(from=1, to=nsim))
colnames(msage)=snames


# lets say we know the start date from our earlier output
msage$date = sager$date
msage$month = sager$month
msage$year = sager$year
msage$day = sager$day
msage$wy = sager$wy

# lets add observed
msage = left_join(msage, sager[,c("obs","date")], by=c("date"))

head(msage)
```

```
##            S1        S2         S3        S4         S5
S6        S7
## 1 0.07191767 0.3316747 0.04331200 0.1875757 0.07469700
0.2454343 0.1347037
## 2 0.06689267 0.3179167 0.04020500 0.1819137 0.06790767
0.2412470 0.1286780
## 3 0.06221900 0.3047440 0.03732067 0.1764227 0.06173567
0.2371983 0.1229220
## 4 0.05787167 0.2921237 0.03464333 0.1710973 0.05612433
0.2332663 0.1174237
## 5 0.05382833 0.2800427 0.03215800 0.1659330 0.05102333
0.2294617 0.1121710
## 6 0.05006733 0.2684613 0.02985100 0.1609243 0.04638600
0.2257630 0.1071530
##              S8        S9         S10       S11       S12
S13          S14
## 1 0.0003533333 0.2383413 0.003331333 0.2431933 0.3644930
0.05328633 0.005250000
```

```
## 2 0.0003400000 0.2321840 0.003039333 0.2355610 0.3583200
0.05014967 0.004755333
## 3 0.0003273333 0.2261857 0.002773000 0.2281683 0.3522187
0.04719767 0.004307333
## 4 0.0003150000 0.2203423 0.002530000 0.2210077 0.3463190
0.04441933 0.003901333
## 5 0.0003033333 0.2146500 0.002308333 0.2140717 0.3404873
0.04180433 0.003533667
## 6 0.0002920000 0.2091047 0.002106333 0.2073533 0.3347960
0.03934333 0.003200667
##         S15         S16       S17         S18         S19
S20       S21
## 1 0.5948570 0.012760333 0.2362903 0.01888033 0.12594367
0.4374097 0.2176843
## 2 0.5860857 0.011643667 0.2341553 0.01800533 0.11671333
0.4312180 0.2053780
## 3 0.5774453 0.010624667 0.2320393 0.01717100 0.10815933
0.4251140 0.1937673
## 4 0.5689357 0.009695000 0.2299423 0.01637500 0.10023233
0.4190963 0.1828130
## 5 0.5605520 0.008846667 0.2278643 0.01561600 0.09288633
0.4131640 0.1724780
## 6 0.5522937 0.008072333 0.2258053 0.01489200 0.08607867
0.4073157 0.1627270
##          S22        S23       S24         S25         S26
S27        S28
## 1 0.03378267 0.06285833 0.1675450 0.01840800 0.07664567
0.08750367 0.06550033
## 2 0.03198167 0.05886167 0.1607863 0.01818167 0.07178267
0.07925833 0.06094633
## 3 0.03027667 0.05511900 0.1543007 0.01795833 0.06722800
0.07178967 0.05670900
## 4 0.02866267 0.05161433 0.1480763 0.01773767 0.06296233
0.06502500 0.05276633
## 5 0.02713500 0.04833233 0.1421033 0.01752000 0.05896733
0.05889767 0.04909767
## 6 0.02568833 0.04525933 0.1363710 0.01730467 0.05522600
0.05334767 0.04568400
##         S29       S30       S31       S32       S33       S34
S35
## 1 0.4238063 0.1451923 0.2529733 0.5392687 0.2826070 0.3202217
0.09478400
## 2 0.4133587 0.1420453 0.2425717 0.5297423 0.2725720 0.3132013
0.08795600
## 3 0.4032640 0.1389667 0.2325977 0.5207750 0.2628933 0.3063350
0.08161967
## 4 0.3935287 0.1359547 0.2230337 0.5123903 0.2535583 0.2996190
0.07573967
## 5 0.3841480 0.1330080 0.2138630 0.5044643 0.2445547 0.2930503
0.07028333
## 6 0.3751000 0.1301250 0.2050693 0.4969153 0.2358707 0.2866257
```

```
0.06522000
##          S36        S37        S38        S39      S40
S41        S42
## 1 0.06635500 0.11842967 0.06669433 0.04664267 0.300477
0.2028417 0.012289333
## 2 0.06367833 0.11037967 0.06533933 0.04223633 0.294672
0.1982920 0.011173667
## 3 0.06110933 0.10287700 0.06401167 0.03824633 0.289076
0.1938443 0.010159667
## 4 0.05864433 0.09588400 0.06271100 0.03463333 0.283719
0.1894963 0.009237667
## 5 0.05627867 0.08936667 0.06143667 0.03136167 0.278557
0.1852460 0.008399333
## 6 0.05400833 0.08329200 0.06018833 0.02839900 0.273602
0.1810907 0.007637000
##          S43        S44       S45       S46       S47
S48       S49
## 1 0.06128400 0.02764267 0.1804390 0.2829493 0.1520090
0.2241143 0.7156417
## 2 0.06053600 0.02508200 0.1691530 0.2743833 0.1437337
0.2130743 0.7082513
## 3 0.05979700 0.02275867 0.1585730 0.2660767 0.1359090
0.2025780 0.7009373
## 4 0.05906700 0.02065067 0.1486547 0.2580213 0.1285100
0.1925987 0.6936990
## 5 0.05834567 0.01873767 0.1393567 0.2502097 0.1215140
0.1831110 0.6865357
## 6 0.05763333 0.01700167 0.1306403 0.2426347 0.1148987
0.1740907 0.6794463
##         S50       S51        S52        S53        S54
S55        S56
## 1 0.2459190 0.2593303 0.04046233 0.10185033 0.06195833
0.10997067 0.009269667
## 2 0.2405390 0.2468773 0.03690200 0.09695700 0.05648833
0.10079000 0.008794000
## 3 0.2352767 0.2350223 0.03365500 0.09229867 0.05150133
0.09237700 0.008343000
## 4 0.2301293 0.2237367 0.03069367 0.08786433 0.04695433
0.08466767 0.007915000
## 5 0.2250950 0.2129927 0.02799267 0.08364300 0.04280867
0.07760267 0.007509000
## 6 0.2201707 0.2027647 0.02552933 0.07962467 0.03902933
0.07112800 0.007123667
##          S57        S58       S59        S60       S61
S62        S63
## 1 0.08622433 0.10054867 0.2285157 0.08376633 0.5664663
0.10368200 0.06505233
## 2 0.07895133 0.09925867 0.2167053 0.07812267 0.5552560
0.09547367 0.06421500
## 3 0.07229167 0.09798533 0.2055057 0.07285900 0.5442673
0.08791533 0.06338833
```

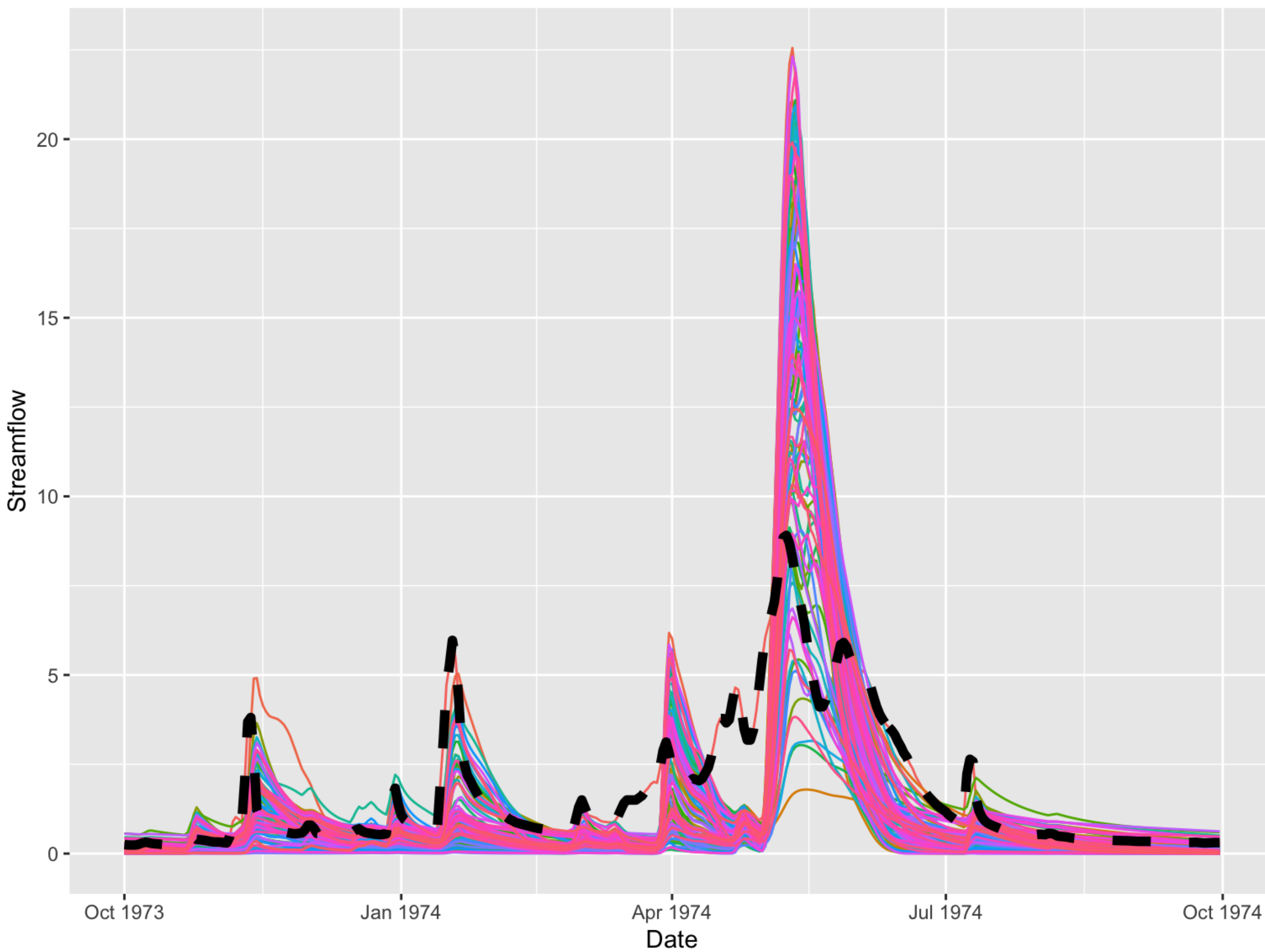
```
## 4 0.06619400 0.09672833 0.1948847 0.06795000 0.5334960
0.08095500 0.06257267
## 5 0.06061067 0.09548733 0.1848123 0.06337167 0.5229380
0.07454600 0.06176733
## 6 0.05549833 0.09426233 0.1752607 0.05910200 0.5125890
0.06864433 0.06097233
##          S64       S65         S66        S67        S68
S69        S70
## 1 0.03208967 0.1484727 0.02082133 0.1788070 0.2103860
0.05299600 0.08575100
## 2 0.02934900 0.1428527 0.01943867 0.1768543 0.2058670
0.05246267 0.08295733
## 3 0.02684233 0.1374453 0.01814767 0.1749230 0.2015403
0.05194533 0.08025467
## 4 0.02454967 0.1322427 0.01694233 0.1730127 0.1974167
0.05144067 0.07764000
## 5 0.02245300 0.1272373 0.01581733 0.1711233 0.1934500
0.05095233 0.07511067
## 6 0.02053500 0.1224213 0.01476667 0.1692543 0.1896473
0.05047133 0.07266367
##          S71          S72        S73         S74        S75
S76       S77
## 1 0.08208500 0.0007126667 0.3321513 0.08189933 0.3378253
0.1432480 0.7430853
## 2 0.07795867 0.0006753333 0.3250353 0.07565067 0.3255447
0.1332823 0.7382633
## 3 0.07404000 0.0006400000 0.3180720 0.06987867 0.3137103
0.1240100 0.7334727
## 4 0.07031800 0.0006063333 0.3112577 0.06454733 0.3023063
0.1153827 0.7287130
## 5 0.06678333 0.0005746667 0.3045897 0.05962267 0.2913170
0.1073557 0.7239843
## 6 0.06342633 0.0005446667 0.2980643 0.05507367 0.2807270
0.0998870 0.7192863
##         S78       S79         S80        S81        S82
S83       S84
## 1 0.1609307 0.1326143 0.08507667 0.5321190 0.6998950
0.06295467 0.4064717
## 2 0.1496117 0.1302127 0.07844300 0.5224367 0.6909930
0.05740367 0.4009937
## 3 0.1390887 0.1278543 0.07232633 0.5129303 0.6822040
0.05234233 0.3955893
## 4 0.1293057 0.1255390 0.06668667 0.5035970 0.6735270
0.04772733 0.3902580
## 5 0.1202110 0.1232657 0.06148667 0.4944333 0.6649603
0.04351900 0.3849987
## 6 0.1117560 0.1210333 0.05669233 0.4854367 0.6565027
0.03968167 0.3798100
##         S85          S86        S87         S88        S89
S90       S91
## 1 0.1612057 0.011333000 0.5693913 0.10873833 0.3803070
```

```
0.5337300 0.1945403
## 2 0.1501753 0.010880000 0.5595980 0.10389400 0.3671423
0.5310793 0.1823263
## 3 0.1398997 0.010444667 0.5499730 0.09926567 0.3544333
0.5284417 0.1708793
## 4 0.1303273 0.010027000 0.5405137 0.09484367 0.3421643
0.5258170 0.1601510
## 5 0.1214097 0.009626000 0.5312170 0.09061867 0.3303200
0.5232057 0.1500963
## 6 0.1131023 0.009241333 0.5220803 0.08658167 0.3188857
0.5206070 0.1406727
##          S92       S93       S94       S95        S96
S97       S98
## 1 0.02710667 0.1718877 0.2836493 0.1334437 0.07881167
0.2935460 0.2200570
## 2 0.02649667 0.1624967 0.2761773 0.1266033 0.07252633
0.2823550 0.2093427
## 3 0.02590033 0.1536187 0.2689023 0.1201153 0.06674233
0.2715907 0.1991500
## 4 0.02531767 0.1452257 0.2618187 0.1139563 0.06141933
0.2612367 0.1894533
## 5 0.02474800 0.1372913 0.2549220 0.1081093 0.05652100
0.2512777 0.1802290
## 6 0.02419133 0.1297903 0.2482067 0.1025590 0.05201333
0.2416983 0.1714537
##           S99       S100       S101       date month year day
wy       obs
## 1 0.011247667 0.07537933 0.04625600 1965-10-01    10 1965   1
1966 0.3358678
## 2 0.010750333 0.07278433 0.04515367 1965-10-02    10 1965   2
1966 0.3208737
## 3 0.010282667 0.07027900 0.04407767 1965-10-03    10 1965   3
1966 0.3058796
## 4 0.009823000 0.06785967 0.04302733 1965-10-04    10 1965   4
1966 0.2968832
## 5 0.009406333 0.06552400 0.04200200 1965-10-05    10 1965   5
1966 0.2968832
## 6 0.008985333 0.06326867 0.04100100 1965-10-06    10 1965   6
1966 0.2968832
```

```
# how can we plot all results - lets plot water year 1970 otherwise its hard to see
msagel = msage %>% pivot_longer(cols=!c(date, month, year, day,wy), names_to="run",
        values_to="flow")


p1=ggplot(subset(msagel, wy ==1974), aes(as.Date(date), flow,
        col=run))+geom_line()+theme(legend.position = "none")
# lets add observed streamflow
p1=p1+geom_line(data=subset(sager, wy == 1974), aes(as.Date(date), obs), size=2,
        col="black", linetype=2)+labs(y="Streamflow", x="Date")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in
ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this
warning was
## generated.
```

```
p1
```



Calibration

apply your function

```
source("../R/nse.R")
source("../R/relerr.R")
# subset for split sample calibration
short_msage = subset(msage, wy < 1975)

# compute performance measures for output from all parameters
res = short_msage %>% select(!c("date","month","year","day","wy","obs")) %>%
      map_dbl(nse, short_msage$obs) # purrr function here! map_dbl will apply the
        function nse() to each column in our data frame against the observed and
        returns a vector


head(res)
```

```
##          S1         S2         S3         S4         S5
S6
## -0.4724480  0.5330579  0.3692951  0.2700532  0.3606149
0.4573876
```

```
# another example using our low flow statistics
# use apply to compute for all the data
source("../R/compute_lowflowmetrics_all.R")
res = short_msage %>% select(-date, -month, -day, -year, -wy, -obs ) %>%
  map_df(compute_lowflowmetrics_all, o=short_msage$obs, month=short_msage$month,
        day=short_msage$day, year=short_msage$year, wy=short_msage$wy)
```

```
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
```

```
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
```

```
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
```

```
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
```

```
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
```

```
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
```

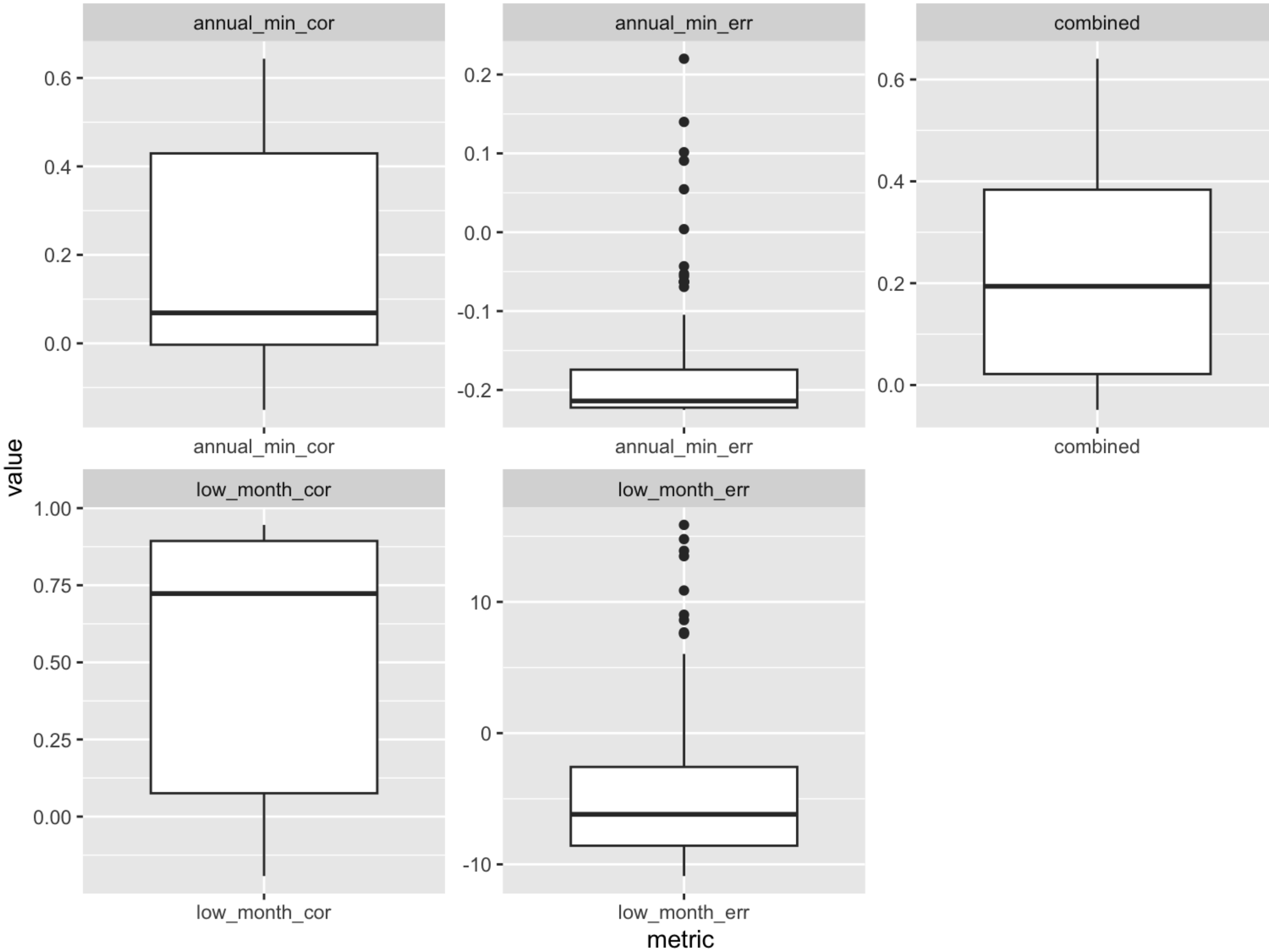
```
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
```

```
# note here we use map_df to get a dataframe back



# interesting to look at range of metrics - could use this to decide on
# acceptable values
summary(res)
```

```
##  annual_min_err    annual_min_cor      low_month_err
low_month_cor
##  Min.   :-0.2252   Min.   :-0.150353   Min.   :-10.890   Min.
:-0.19268
##  1st Qu.:-0.2223   1st Qu.:-0.003188   1st Qu.: -8.579   1st
Qu.: 0.07569
##  Median :-0.2140   Median : 0.068846   Median : -6.188
Median : 0.72298
```

```
##  Mean   :-0.1756   Mean   : 0.176563   Mean   : -4.368   Mean
: 0.52731
##  3rd Qu.:-0.1743   3rd Qu.: 0.429427   3rd Qu.: -2.576   3rd
Qu.: 0.89349
##  Max.   : 0.2200   Max.   : 0.643432   Max.   : 15.866   Max.
: 0.94551
##     combined
##  Min.   :-0.04867
##  1st Qu.: 0.02162
##  Median : 0.19392
##  Mean   : 0.22239
##  3rd Qu.: 0.38361
##  Max.   : 0.64072
```

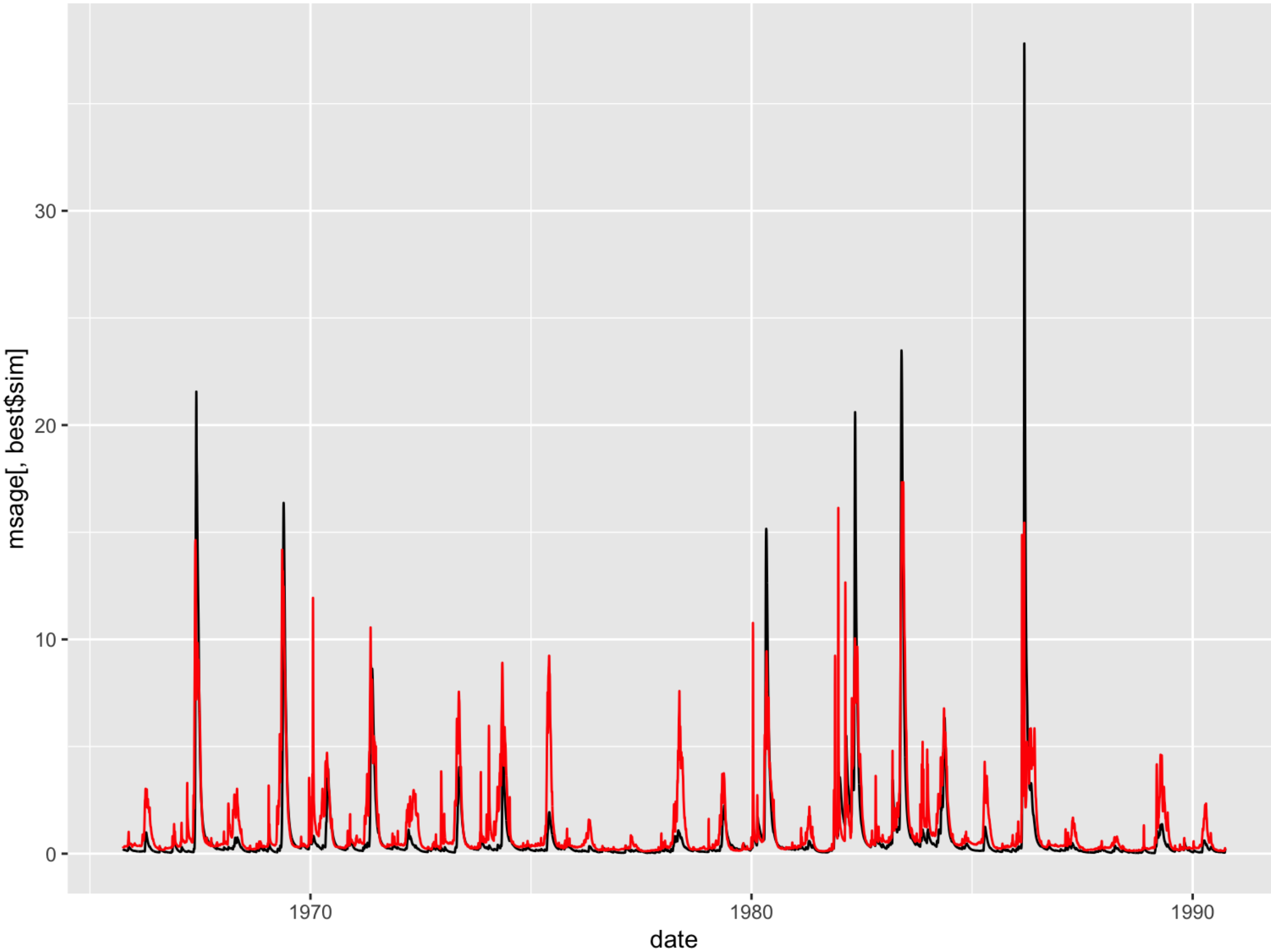
```
# we can add a row that links with simulation number
res$sim = snames

# graph range of performance measures
resl = res %>% pivot_longer(-sim, names_to="metric", values_to="value")

ggplot(resl, aes(metric, value))+geom_boxplot()+facet_wrap(~metric, scales="free")
```


annual_min_cor
annual_min_err
combined
low_month_cor
low_month_err
value
metric
0.6
0.4
0.2
0.0
0.2
0.1
0.0
-0.1
-0.2
0.6
0.4
0.2
0.0
1.00
0.75
0.50
0.25
0.00
10
0
-10
annual_min_cor
annual_min_err
combined
low_month_cor
low_month_err

Explore how ‘good’ parameters compare with “bad” parameters

Extract out the “best” parameter set

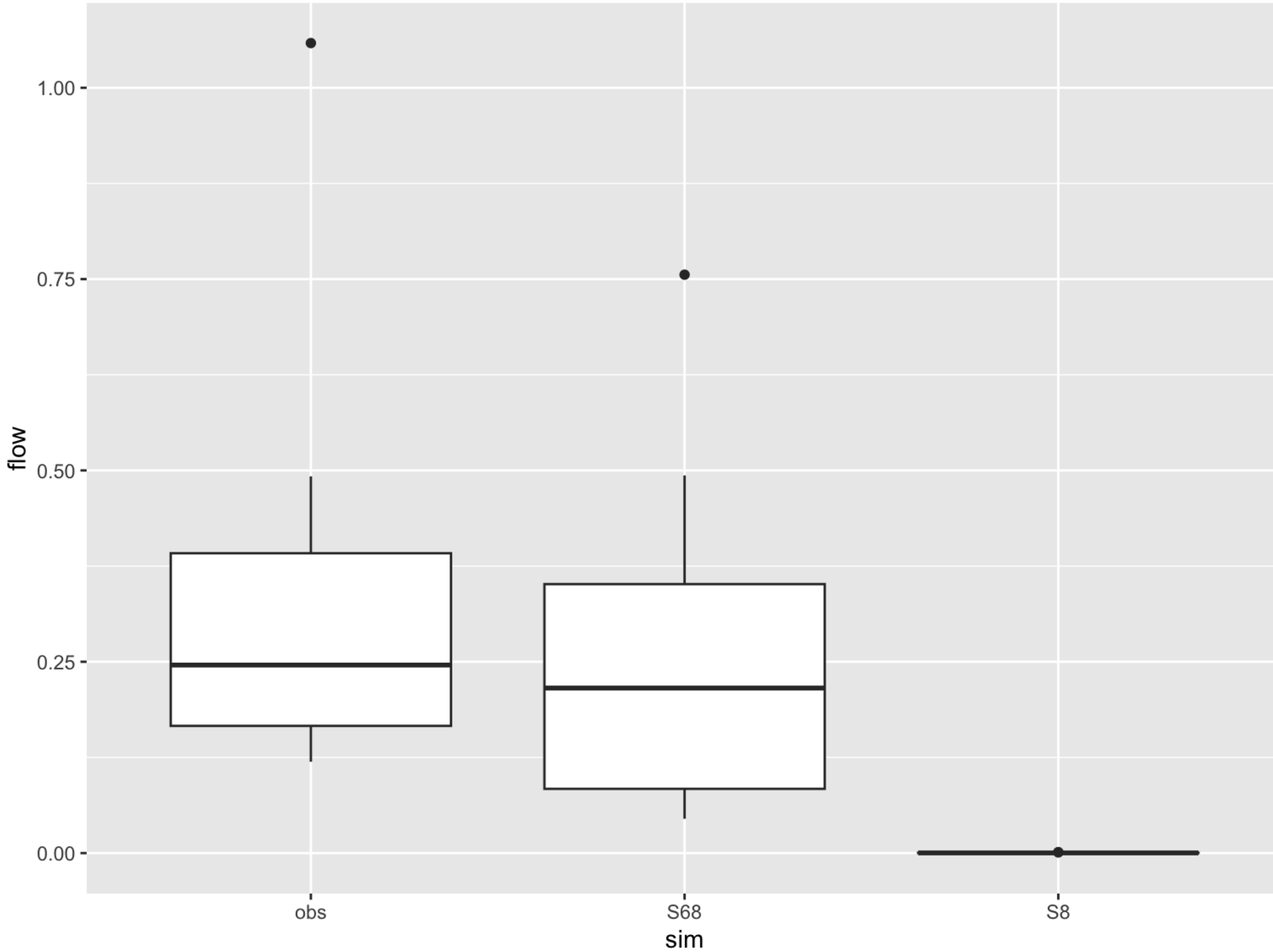
```
# select the best one based on the combined metric
best = res[which.max(res$combined),]

# running the model forward
# so we can look at the full time series

# lets start with streamflow estimates from best performing parameter set
 ggplot(msage, aes(date, msage[,best$sim])) + geom_line()+geom_line(aes(date, obs),
        col="red")
```




```
# for comparison lets consider how worst and best parameters perform for subsequent
        simulations
# focusing specifically on August streamflow
 worst = res[which.min(res$combined),]

 compruns = msage %>% select(best$sim, worst$sim, date, obs, month, day, year, wy)
 compruns = subset(compruns, wy > 1970)
 compruns_mwy = compruns %>% select(-c(day,date, year)) %>% group_by(month, wy) %>%
        summarize(across(everything(), mean))
```

```
## `summarise()` has grouped output by 'month'. You can override
using the
## `.groups` argument.
```

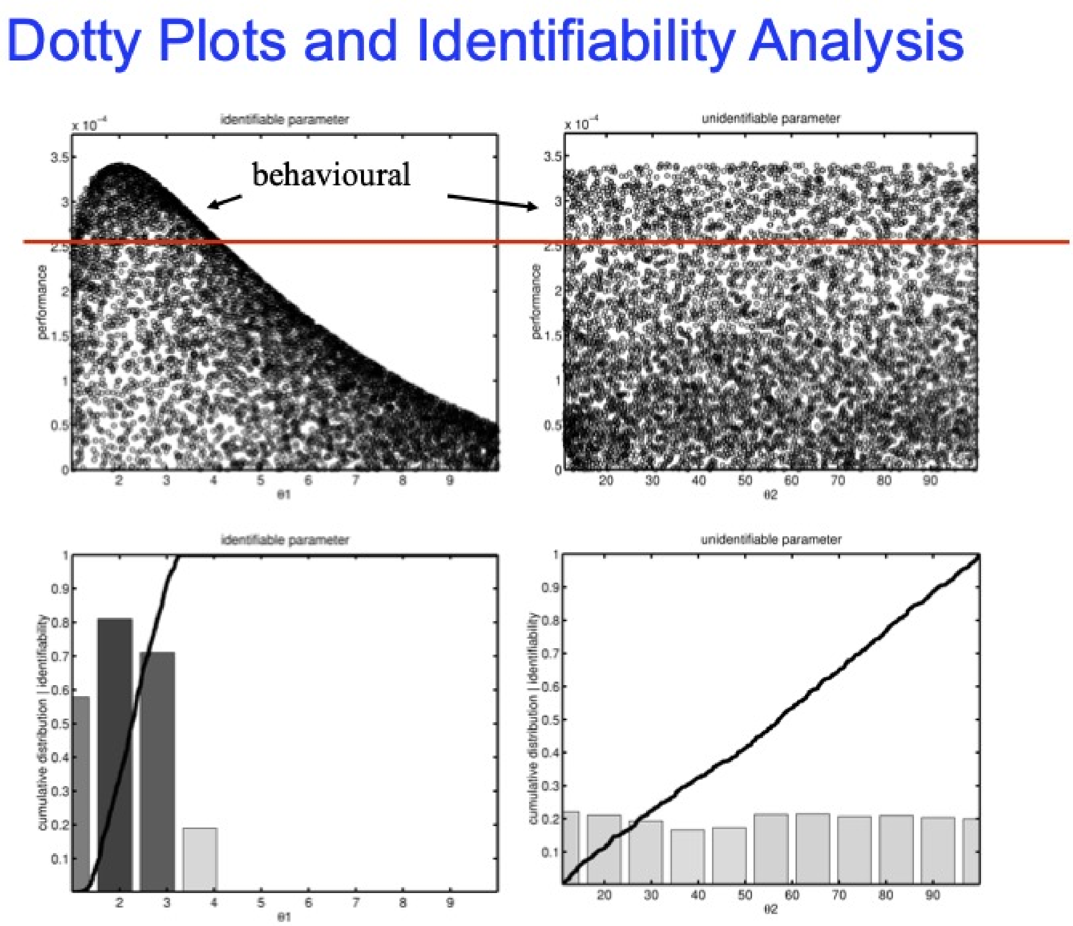
```
 compruns_mwyl = compruns_mwy %>% pivot_longer(cols=!c(month,wy), names_to="sim",
        values_to="flow")
 compruns_mwyl %>% subset(month==8) %>% ggplot(aes(sim,flow ))+geom_boxplot()
```


1.00
0.75
0.50
0.25
0.00
flow
obs
S68
S8
sim

More on calibration - some complications

Equifinality.

Many parameter values can give equal performance

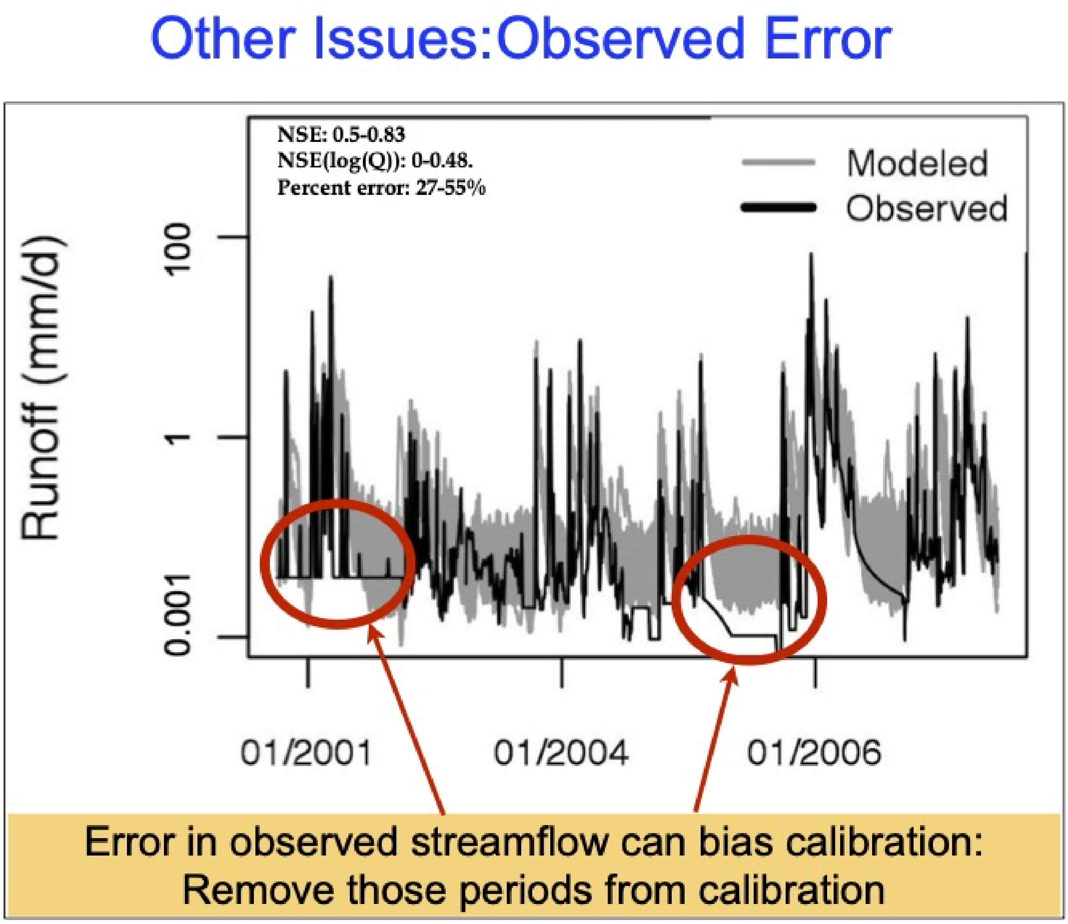


Equifinality

Other Issues

- comparison with observed data is limited

*errors in observed data



Equifinality

why picking the “best” parameter set may not be ideal

Parameter optimization/evaluation: will not be robust

- calibration period is limited
- different performance measure give different “best” parameters
- input/measurement errors
- equifinality
- overfitting in time and space

possible approach

Bayesian, or simply use all "acceptible" parameter sets

- assess the likelihood of different models + parameters being good predictors of the system of interest
- reject (give zero likelihood) those models that are clearly not good predictors of calibration data
- Can be done with different model structures as well as different parameter sets

Steps

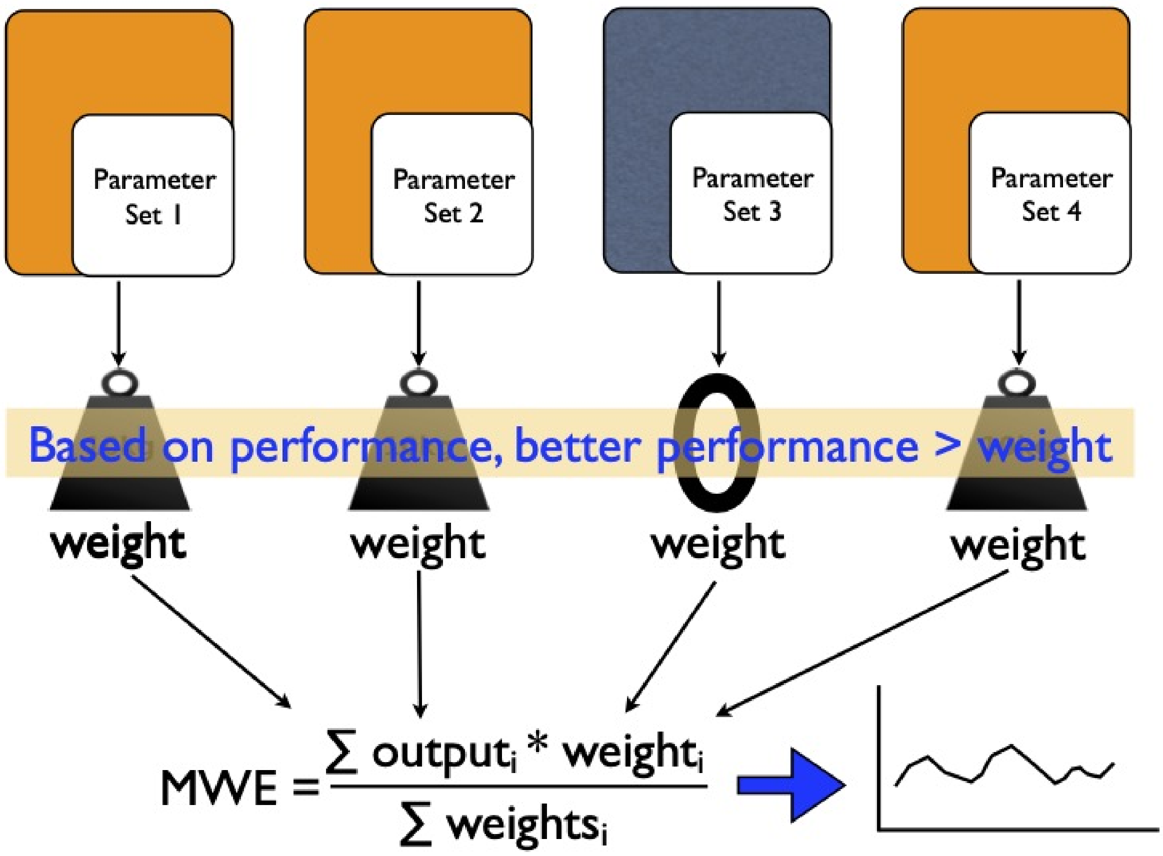
Keep all parameter sets that are acceptable

- acceptable: above some threshold of performance
- always run the model for those parameter sets and use range of model output to define uncertainty bound

if you need a single model es5mate:

- combine results from all acceptable parameters
- average »weight by performance

Steps 2



Equifinality

Generalized Likelihood Uncertainty Estimate< (GLUE)

Generalized Likelihood Uncertainty Estimate.

Beven, JH, 2006, Manifesto for the Equifinality Thesi

How to do a maximum likelihood estimate of model results in R

Glue - generalized uncertainty analysis

What if we wanted to keep all of the ‘good’ parameters

- we could just keep them all as equally likely
- we could weight them by performance

Either way we can graph and come up with ‘best’ prediction accounting for uncertainty

Calibration with GLUE

Create a single measure of accuracy - above we used *compute_lowlowmetrics_all* to compute an accuracy measure based on

- relative error in annual minimum flow estimate
- relative error in monthly flow during low flow period
- correlation between observed and modelled annual minimum flow
- correlation between observed and modelled flow during the low flow period

We weighted all 4 the same

Use the accuracy measure

We can use the combined accuracy measure to define behavioural (acceptable) parameter set (**res_acc**) - two options

- define a threshold for acceptability (we will use 30%)
- take top 50 performing parameter sets

(we go with the latter but code could be commented to go with threshold approach)

Define behavioral / acceptable parameter set

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
## -0.04867  0.02162  0.19392  0.22239  0.38361  0.64072
```

```
## # A tibble: 6 × 6
##   annual_min_err annual_min_cor low_month_err low_month_cor
combined sim
##            <dbl>          <dbl>         <dbl>         <dbl>
<dbl> <chr>
## 1       -0.190         -0.00169        -0.483         0.786
0.424 S2
## 2       -0.108          0.447          -0.908         0.931
0.563 S6
## 3       -0.172          0.228          -3.09          0.902
0.391 S9
## 4       -0.184          0.109          -3.41          0.870
0.338 S11
## 5       -0.105          0.580           4.48          0.929
0.440 S12
## 6        0.00389        0.577          10.9           0.893
0.609 S15
```

```
## # A tibble: 6 × 6
##   annual_min_err annual_min_cor low_month_err low_month_cor
combined sim
##            <dbl>          <dbl>         <dbl>         <dbl>
<dbl> <chr>
## 1       -0.130            0.638       -0.0671         0.938
0.641 S68
## 2        0.00389          0.577       10.9            0.893
0.609 S15
## 3       -0.0693           0.611        3.28           0.893
0.572 S84
## 4       -0.141            0.364       -0.0261         0.908
0.567 S34
## 5       -0.108            0.447       -0.908          0.931
0.563 S6
## 6       -0.137            0.399        0.417          0.909
0.558 S73
```

Defining weights (likelihood) for parameter sets

Now define "weights" (likelihood) based on parameter performance for the acceptable or behaviorial parameters

We want the sum of the weights to equal 1

- accuracy measure defined above will define weight
- we divide by the sum of all accuracy measures to get fractions that add to 1
- note we now only work with behavioural parameter sets (in ** res_acc ** versus ** res **)

```
## # A tibble: 6 × 7
##   annual_min_err annual_min_cor low_month_err low_month_cor
combined sim
##            <dbl>          <dbl>         <dbl>         <dbl>
<dbl> <chr>
## 1       -0.130            0.638       -0.0671         0.938
0.641 S68
## 2        0.00389          0.577       10.9            0.893
0.609 S15
## 3       -0.0693           0.611        3.28           0.893
0.572 S84
## 4       -0.141            0.364       -0.0261         0.908
0.567 S34
## 5       -0.108            0.447       -0.908          0.931
0.563 S6
## 6       -0.137            0.399        0.417          0.909
0.558 S73
## # ℹ 1 more variable: wt_acc <dbl>
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.009744 0.016850 0.019442 0.020000 0.023189 0.032167
```

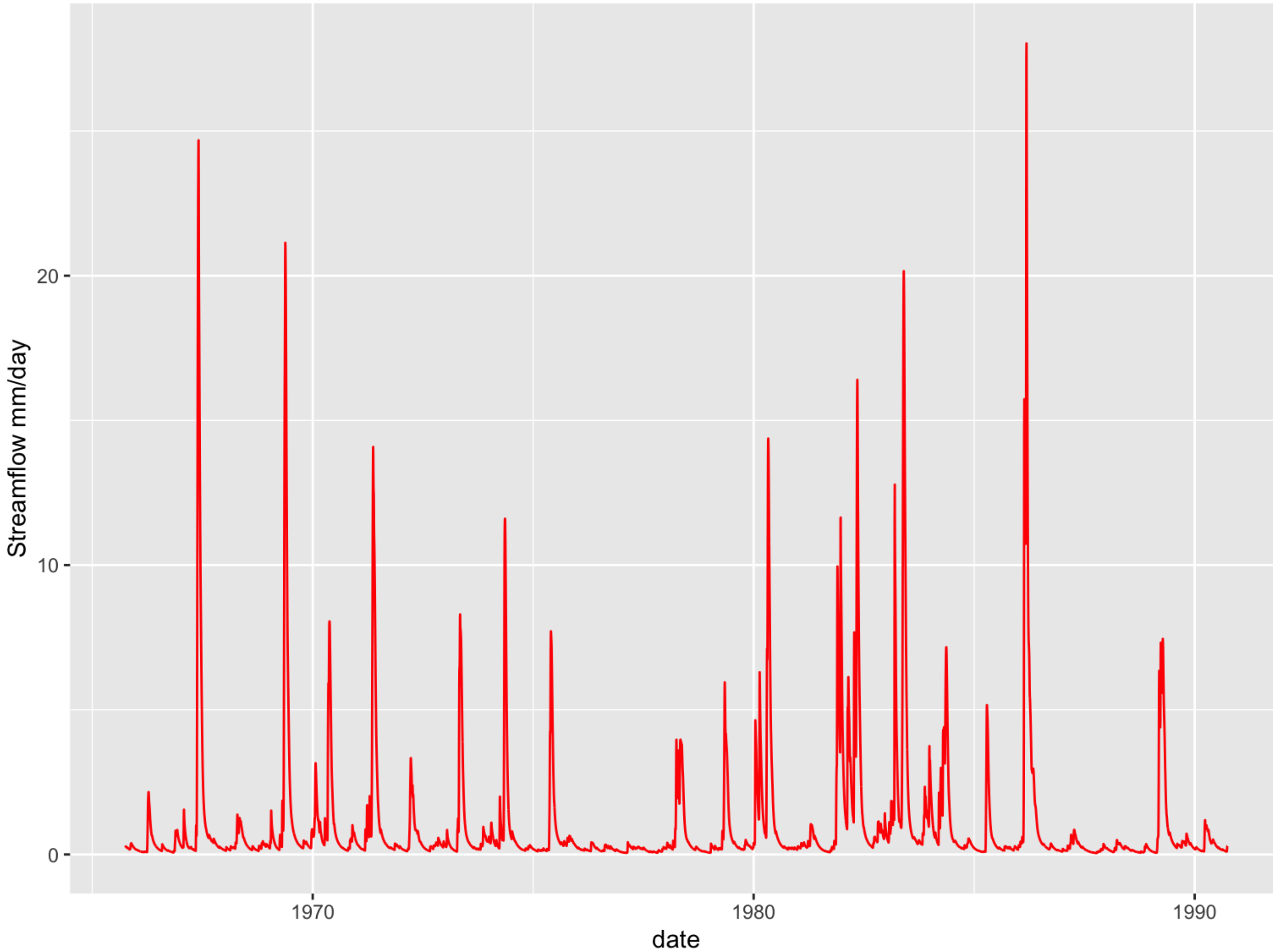
```
## [1] 1
```

```
## [1] 50
```

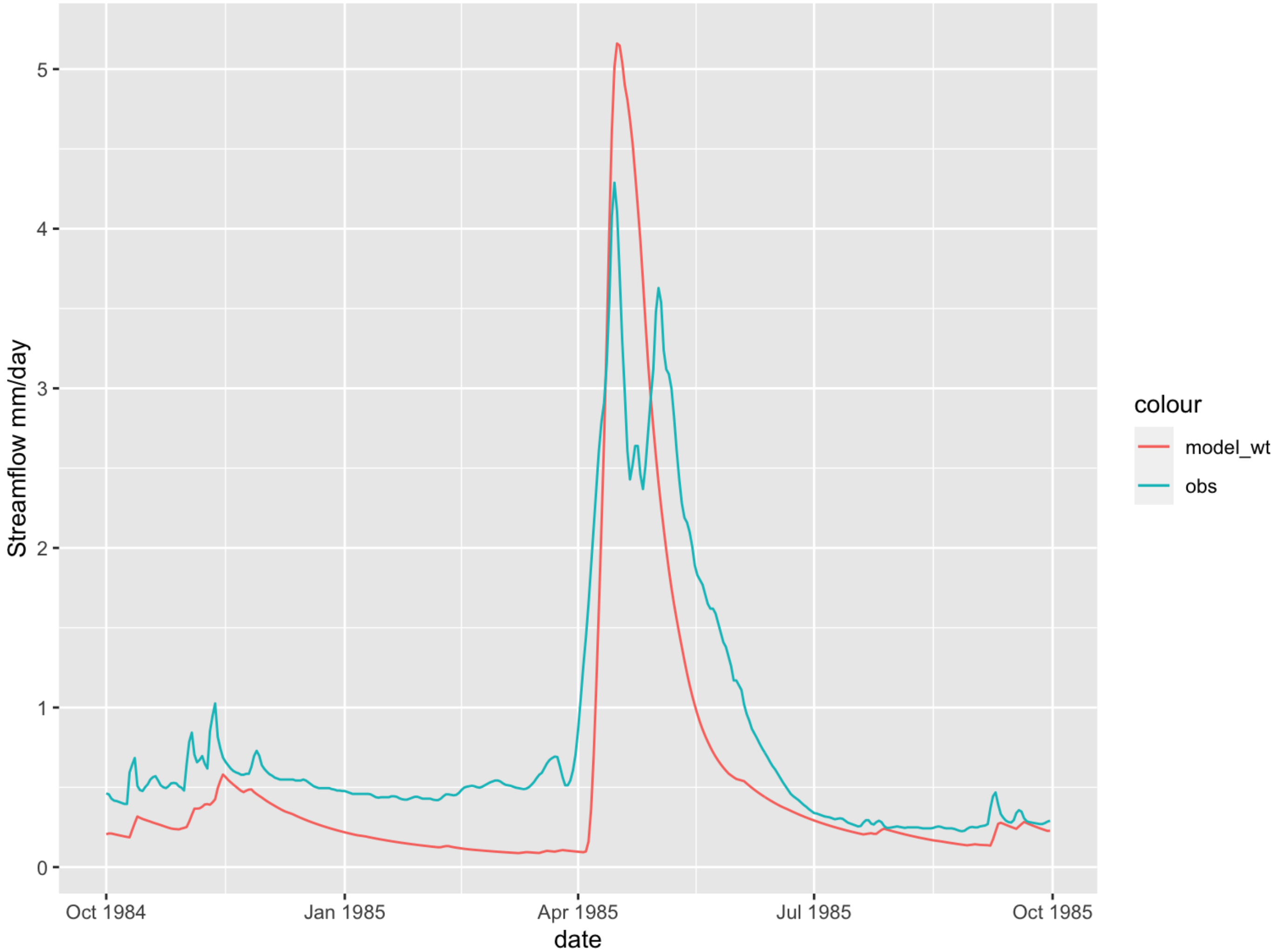
Using weights

One way to use weights is to define a maximum likelihood estimate by averaging (weighted by accuracy) streamflow from all behavioural simulations

```
## # A tibble: 6 × 14
##   date       month  year   day    wy   obs sim    flow
annual_min_err
##   <date>     <int> <int> <int> <int> <dbl> <chr> <dbl>
<dbl>
## 1 1965-10-01    10  1965     1  1966 0.336 S2    0.332
-0.190
## 2 1965-10-01    10  1965     1  1966 0.336 S4    0.188
-0.191
## 3 1965-10-01    10  1965     1  1966 0.336 S6    0.245
-0.108
## 4 1965-10-01    10  1965     1  1966 0.336 S9    0.238
-0.172
## 5 1965-10-01    10  1965     1  1966 0.336 S11   0.243
-0.184
## 6 1965-10-01    10  1965     1  1966 0.336 S12   0.364
-0.105
## # ℹ 5 more variables: annual_min_cor <dbl>, low_month_err
<dbl>,
## #   low_month_cor <dbl>, combined <dbl>, wt_acc <dbl>
```



Streamflow mm/day
0
10
20
1970
1980
1990
date



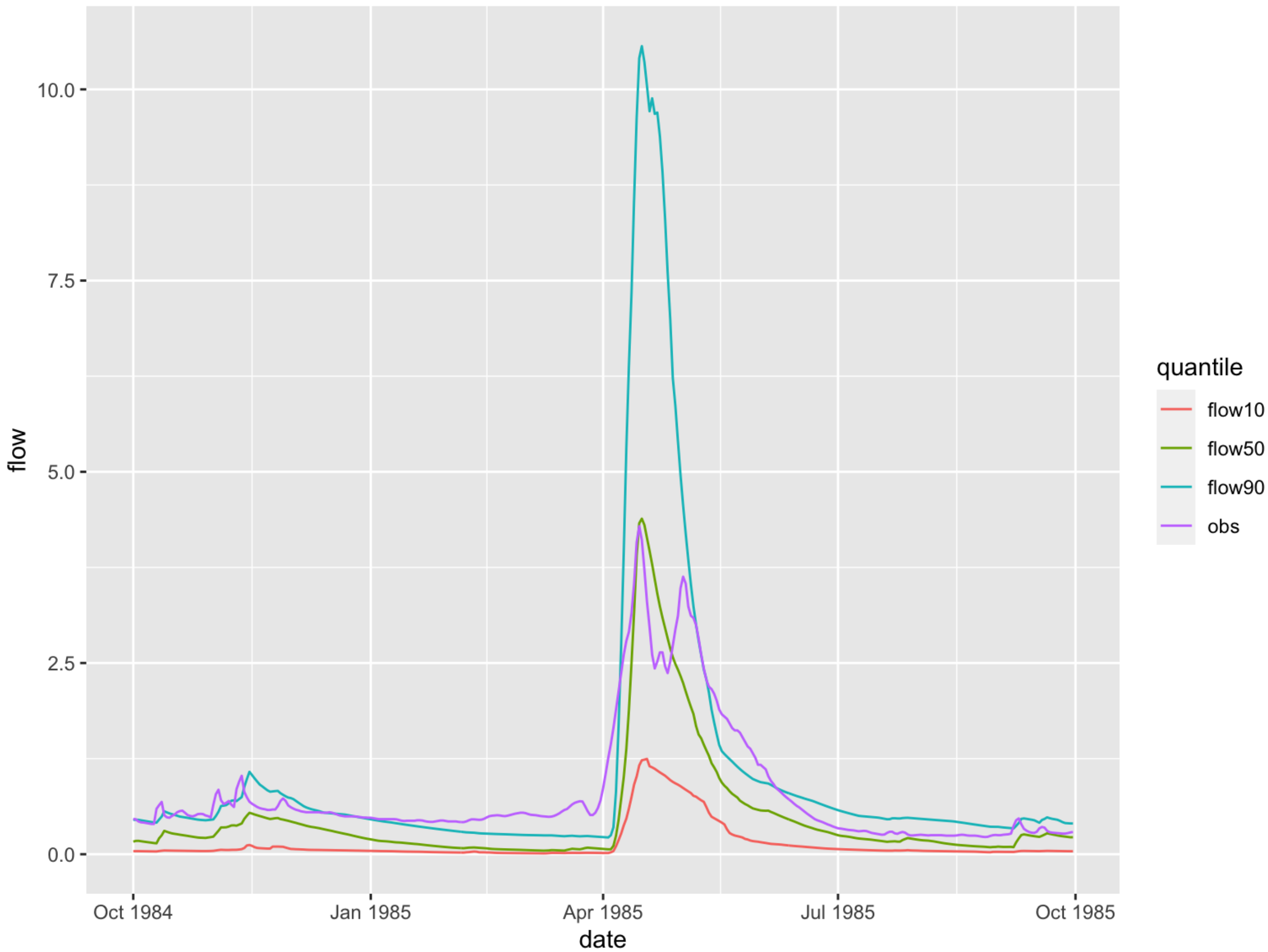
Streamflow mm/day
0
1
2
3
4
5
Oct 1984
Jan 1985
Apr 1985
Jul 1985
Oct 1985
date
colour
model_wt
obs

Final step of GLUE

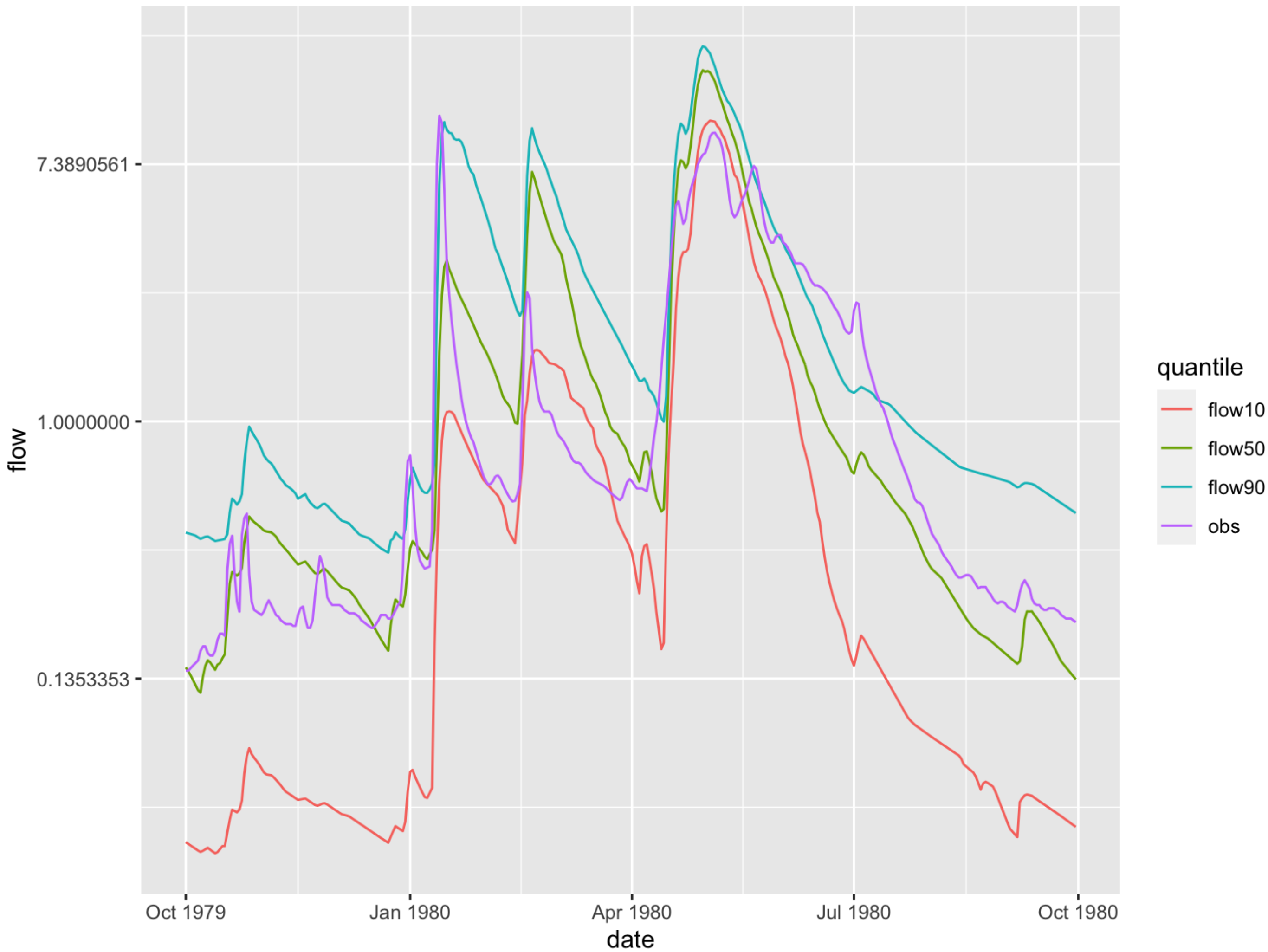
We could also compute quantiles rather than just mean

We can use the `wtd.quantile()` function in the `reldist` package to do this - it computes quantiles accounting for different weights on each observation

```
# compute quantiles based on performance weights
quant_flow = msagel_acc %>% group_by(date) %>% dplyr::summarize(
  flow10=wtd.quantile(x=flow, weight=wt_acc, q=0.1),
  flow50=wtd.quantile(x=flow, weight=wt_acc, q=0.5),
  flow90=wtd.quantile(x=flow, weight=wt_acc, q=0.9)
)


# ad observed back
quant_flow_obs = left_join(quant_flow, msage[,c("date","month","year",
        "day","wy","obs")],
                       by=c("date"))

# format for plotting
quant_flowl = quant_flow_obs %>% pivot_longer(col=c(flow10, flow50, flow90, obs),
                                          values_to="flow", names_to="quantile")

# plot
ggplot(subset(quant_flowl, wy==1985), aes(date, flow, col=quantile))+geom_line()
```




```
# to see low flows, transform y-axis
ggplot(subset(quant_flowl, wy==1980), aes(date, flow, col=quantile))+
  geom_line()+scale_y_continuous(trans="log")
```


flow
7.3890561
1.0000000
0.1353353
date
Oct 1979
Jan 1980
Apr 1980
Jul 1980
Oct 1980
quantile
flow10
flow50
flow90
obs

Extra Credit

Final piece will be to produce a graph of maximum likelihood estimate given you acceptable parameters!

To hand in - an Rmarkdown and R function.

- Part 3 (OPTIONAL)

  1. *Use the performance measure to select "acceptable" outcomes from parameter sets*
  2. *Compute the range of the performance measure using only the "acceptable" outcomes over the post-calibration period (part that you didn't use for calibration in step 1)*
  3. *Graph the range of outcomes for acceptable parameters (e.g post-calibration parameter uncertainty); you can choose what output is most interesting for you*
  4. *Compute and graph the maximum likelihood estimate of your output of interest (e.g minimum summer streamflow each year) for the post-calibration period (see #16 or #17 in contents)*

- Extra Credit! up to 15pts

  - *your metrics are used to select 'acceptable' parameter set outcomes (5)*
  - *metrics are computed for post-calibration data of accepted parameter set outcomes (5)*

- *maximum likelihood estimate is computed for post-calibration data (5)*