gate_regression

Nadya Alexander

September 10th, 2019

#Introduction. The overall goal of this model is to evaluate a law that was passed in California. The state wants farmers to improve the accuracy of their farmgates. Farmgates are gates that measure water as it flows from the delivery network ("lateral canal") onto their farm. The purpose of this model is build two gates - a lower accuracy gate and an improved gate. These gates are then put into a simulation of an irrigation district to see how their accuracy relates to the overall accuracy of water deliveries for the whole district given a range of operating conditions and uncertainty in the input variables.

What this translates to, basically, is a lot of simulations where the variances of the input parameters (gate position, upstream water level, and time) are changed and the variance of the output (flow) is observed. The type of simulations that are done are called Monte Carlo simulations, which are basically for-loops that sample variables from a uniform or normal distribution with defined standard deviations and means. And then there are lots of plots.

The code should, mostly, be running. The programmer I was working with was able to run the code through to the end and give me the outputs (attached as outputs.zip). However, I was not able to run the code all the way through. I found one definite error on line 105, there is a reference to "nformulas". Originally there were 4 different flow equations programmed, because there are different sizes of gates. We stripped those out, but I think this was left in to the nested for-loop by accident. I didn't get further than this point, so there may be other small errors, but I expect that it should be at least almost completely working.

0 Citations

```
# # cite R citation() toBibtex(citation()) # cite R studio RStudio.Version() #
# cite packages citethese <- c('nls2', 'reshape2', 'hydroGOF') for(i in
# seq_along(citethese)){ x <- citation(citethese[i]) print(x) #
# print(toBibtex(x)) } remove(x) remove(i) remove(citethese)
```

1 Data

1.1 Data Gathering– Table of Flows Through a Gate Dependent on Upstream Level (H) and Gate Openning (G)

```
# The only external flow data that is brought into the model is a flow table from
# the gate manufacturer. A flow table shows the flow (Q) for a variet of Gate
# Positions (G) and Upstream Levels (H). The flow table is created from an
# empirical equation based on lab data. This flow table data is used in the model
# below as the 'truth', or the 'observed' in a predicted versus observed plot.

# There is other data that is brought in later in the model, that data is to show
# how many lateral canals and gates there are in the system.

flowdf <- read.csv("/Users/TheVault/Desktop/Upwork/Dissertation/inputs/armco_gate18.csv",
    header = TRUE, check.names = FALSE, fileEncoding = "UTF-8-BOM")  #PM Comment: Reads in data, header
# row as column names, check.names allows for numeric column names

ngates <- 1  #length(flowdf) #<-----assign a number to ngates
```

1.2 Data Preprocessing

```
library(reshape2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:reshape2':
##
##     smiths
```

```
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
## The following objects are masked from 'package:reshape2':
##
##     dcast, melt
```

```
flowdfl <- gather(na.omit(flowdf), "G", "flow", 2:ncol(flowdf)) %>% mutate(G = as.numeric(G))
# group_by(G) mutate(G = group_indices()) %>% select(-G_val)

# PM Comment: removes NA cells from flow data frame (na.omit), takes column names
# and shifts them into rows (gather), changes G to a number instead of a
# string(mutate)

# to take a look do this
head(flowdfl)
```

```
##      H    G flow
## 1 0.08 0.17 0.49
## 2 0.10 0.17 0.55
## 3 0.13 0.17 0.59
## 4 0.15 0.17 0.63
## 5 0.17 0.17 0.67
## 6 0.19 0.17 0.71
```

```
tail(flowdfl)
```

```
##        H    G  flow
## 660 1.08 1.33  8.98
```

```
## 661 1.17 1.33  9.32
## 662 1.25 1.33  9.65
## 663 1.33 1.33  9.96
## 664 1.42 1.33 10.27
## 665 1.50 1.33 10.56
```

2 Model Fit

```
# This section is the gate equation that was created from 'flowdfl' . I could not
# get the model fit to work in R, we tried log-transforming the data, etc., but
# it never worked. So I used an outside statistical package called TableCurve3D
# to fit the data to the equation below with these coefficients.

# This equation is basically Q (flow rate) = f(G (gate position), H(upstream
# level)) Later in the model this equation will be expanded to V (Volume) = Q
# (flow) * t (time)

# vector modelcoef
a = 0.228970726
b = 8.332215198
c = -1.2246326
d = -0.275129331
e = -0.03358984
aa = -0.019840431
bb = -0.49339127
cc = 0.13387028
dd = -0.021659671

modelcoef <- list(a = 0.228970726, b = 8.332215198, c = -1.2246326, d = -0.275129331,
    e = -0.03358984, aa = -0.019840431, bb = -0.49339127, cc = 0.13387028, dd = -0.021659671)

# model formula
formulastring <- list()
formulastring[[1]] <- (a + b * flowdfl$G + c * flowdfl$G^2 + d * flowdfl$G^3 + e *
    log(flowdfl$H))/(1 + aa * flowdfl$G + bb * log(flowdfl$H) + cc * (log(flowdfl$H))^2 +
    dd * (log(flowdfl$H))^3)
```

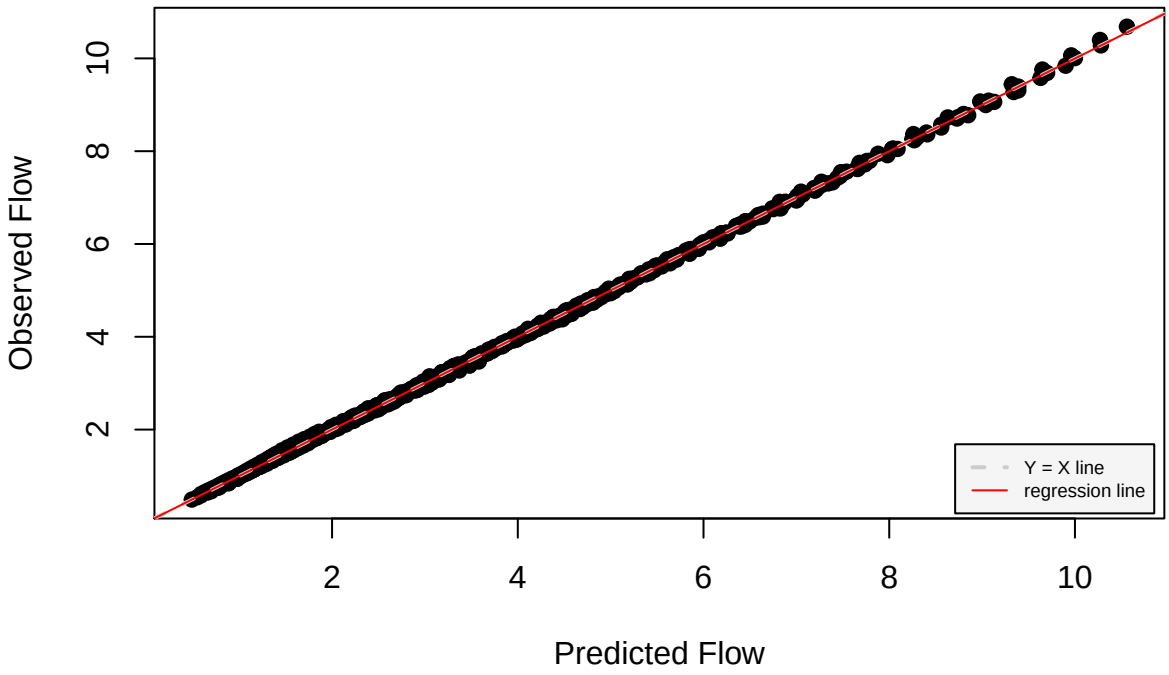
3 Gate Flow Fitting Plots

```
# This section creates plots of observed versus predicted data. This is also
# where I believe there is an error - we have set ngates = 1 above, because there
# is only 1 gate equation now. When I was trying to fit the flow equation in R
# there were multiple formulas, hence the nformulas loop. But now there is only 1
# formula, so there should be, I believe, only the outer for loop.

# for(di in 1:ngates){#As gates are only 1, just 1 is needed for di
di <- 1  #<----- set di
m <- 1  #for (m in 1:nformulas)

# PM Comment: This starts the charting call
png(paste0("outputs/obspreplots/ovp_d", di, "_m", m, ".png"), width = 3.25, height = 2.85,
    units = "in", pointsize = 8, res = 1200)
```

```
plot(flowdfl$flow, formulastring[[1]], ylab = "Observed Flow", xlab = "Predicted Flow",
    pch = 19)

# adj line for the perfect fit
abline(0, 1, col = "grey80", lty = 2, lwd = 2)

# adj line for linear fit
lmmod <- lm(formulastring[[1]] ~ flowdfl$flow)
abline(lmmod, col = "red")
legend("bottomright", horiz = FALSE, inset = c(0.01, 0.01), cex = 0.6, c("Y = X line",
    "regression line"), lty = c(2, 1), lwd = c(2, 1), col = c("grey80", "red"), bg = "grey96",
    xpd = TRUE)
# mtext(paste('Y =', round(lmmod$coefficients[2],3), 'X +',
# round(lmmod$coefficients[1],0)), side=3, line=1, cex=0.8, adj=0.1)
# mtext(paste('Model bR2:', round(t(modelstats[[di]][[m]])[18],3)), side=3,
# line=0.3, cex=0.8, adj=0.1)

dev.off()
```

```
## pdf
##   2
```

```
###### Code for R Markdown Display, Can be Deleted
plot(flowdfl$flow, formulastring[[1]], ylab = "Observed Flow", xlab = "Predicted Flow",
    pch = 19)

# adj line for the perfect fit
abline(0, 1, col = "grey80", lty = 2, lwd = 2)

# adj line for linear fit
lmmod <- lm(formulastring[[1]] ~ flowdfl$flow)
abline(lmmod, col = "red")
legend("bottomright", horiz = FALSE, inset = c(0.01, 0.01), cex = 0.6, c("Y = X line",
    "regression line"), lty = c(2, 1), lwd = c(2, 1), col = c("grey80", "red"), bg = "grey96",
    xpd = TRUE)
```




```
# mtext(paste('Y =', round(lmmod$coefficients[2],3), 'X +',
# round(lmmod$coefficients[1],0)), side=3, line=1, cex=0.8, adj=0.1)
# mtext(paste('Model bR2:', round(t(modelstats[[di]][[m]])[18],3)), side=3,
# line=0.3, cex=0.8, adj=0.1)

dev.print()
```

```
## pdf
##   2
```

```
######
```

4 Uncertainty from Flow Table Equation

```
# This section has had the code stripped, but I kept the notes below. The law
# specifies that the new, more accurate, gates must have 'less than 6% error'.
# But this of course does not specifically correlate to a standard deviation or
# variance, so we defined it as below.

# Since flow is turbulent, let's assume a normal distribution of the errors. At a
# 95% confidence interval, with a normal distribution, we report the +/-2
# standard deviation as the error.

# For simplicity, we will define the error distribution for coefficients of the
# equation to be all the same: normal distribution with the same coefficient of
# variation (standard deviation/mean).

# for +/- 6% error of Q 2SD = 0.06 * Q SD = 0.03 * Q VAR^0.5 = 0.03 * Q VAR =
# 0.0009 * Q^2
```

##4.1 Monte Carlo. Monte Carlo optimization modeling is a statistical modeling method that is useful when an analytical analysis isn't possible. In this case, I did explore a First-Order Second-Moment analysis

of the mean and variance, but since I don't have field data, and therefore don't know about covariance, my PI recommended this method instead.

Monte Carlo simulations can be used to analyze complex problems, but they are pretty simple. Basically, if you have an equation Q = G*H, you run a high number (here, 1,000) loops over that equation, and each time you sample G and H from a defined probability distribution. With a high number of loops, the mean and variance of the output (Q) should be repeatable.

The purpose of this section of code is to assign uncertainty to the equation itself. How do we represent 6% error for the new gates or 12% error for the old gates mathematically? What I decided to do is assign a coefficient of variation to the equation parameters. This section of code runs "flowsim", which calculates the Monte Carlo simulated flow, and the "flowtruth", which calculates the flow based on the mean of the parameters only (i.e., no probability distribution). Although later in the model we will be focused on playing with the variance of the input variables - gate position, level, and time - here we are not concerned with that. At this stage, we are specifically manipulating the coefficient of variance (cv) of the parameters (coefficients) only so that the equation itself has an uncertainty independent of how the gate is operated. The output of this section will determine how we differentiate between the old (baseline) and new (improved) gates.

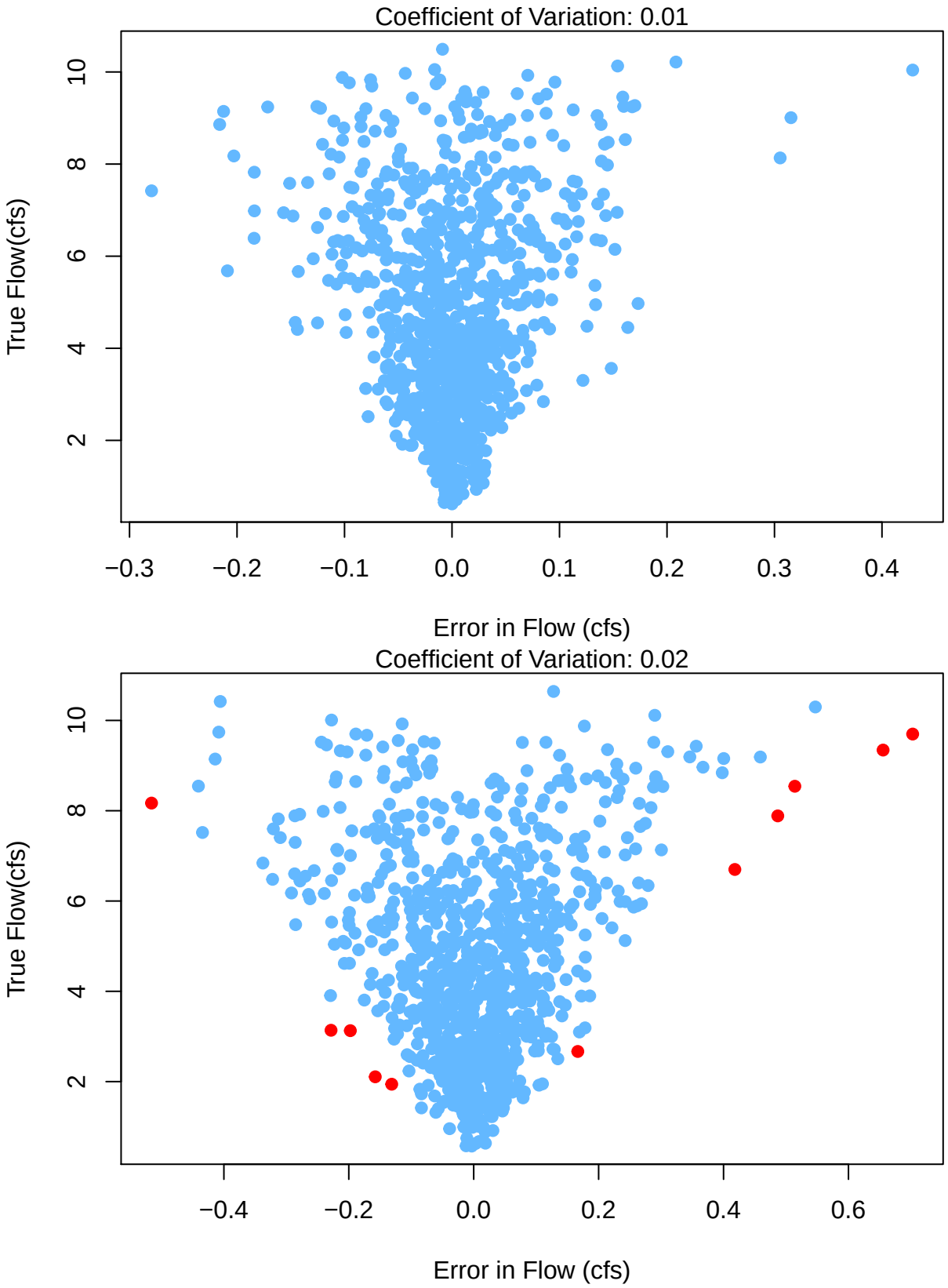
```
# the number of Monte-Carlo loops
nsim <- 1000  #<----set number of simulations
set.seed(19320928)  #<----set randomization seed

# optflowerrorp is the percent of error allowed, like 6%, 12%
equationerrorsim <- function(dataindex, coefofvar, optflowerrorp) {
    # dataindex is probably unnecessary with only one gate
    di <- dataindex

    # we can add the ability to call different functions here, but for now just
    # equation #1
    atrue <- modelcoef[[1]]  # assume the mean is the true value
    astdv <- abs(coefofvar * modelcoef[[1]])
    anorm <- rnorm(nsim, modelcoef[[1]], astdv)  # samples around true value, creating error

    btrue <- modelcoef[[2]]
    bstdv <- abs(coefofvar * modelcoef[[2]])
    bnorm <- rnorm(nsim, modelcoef[[2]], bstdv)

    ctrue <- modelcoef[[3]]
    cstdv <- abs(coefofvar * modelcoef[[3]])
    cnorm <- rnorm(nsim, modelcoef[[3]], cstdv)

    dtrue <- modelcoef[[4]]
    dstdv <- abs(coefofvar * modelcoef[[4]])
    dnorm <- rnorm(nsim, modelcoef[[4]], dstdv)

    etrue <- modelcoef[[5]]
    estdv <- abs(coefofvar * modelcoef[[5]])
    enorm <- rnorm(nsim, modelcoef[[5]], estdv)

    aatrue <- modelcoef[[6]]
    aastdv <- abs(coefofvar * modelcoef[[6]])
    aanorm <- rnorm(nsim, modelcoef[[6]], aastdv)

    bbtrue <- modelcoef[[7]]
    bbstdv <- abs(coefofvar * modelcoef[[7]])
```

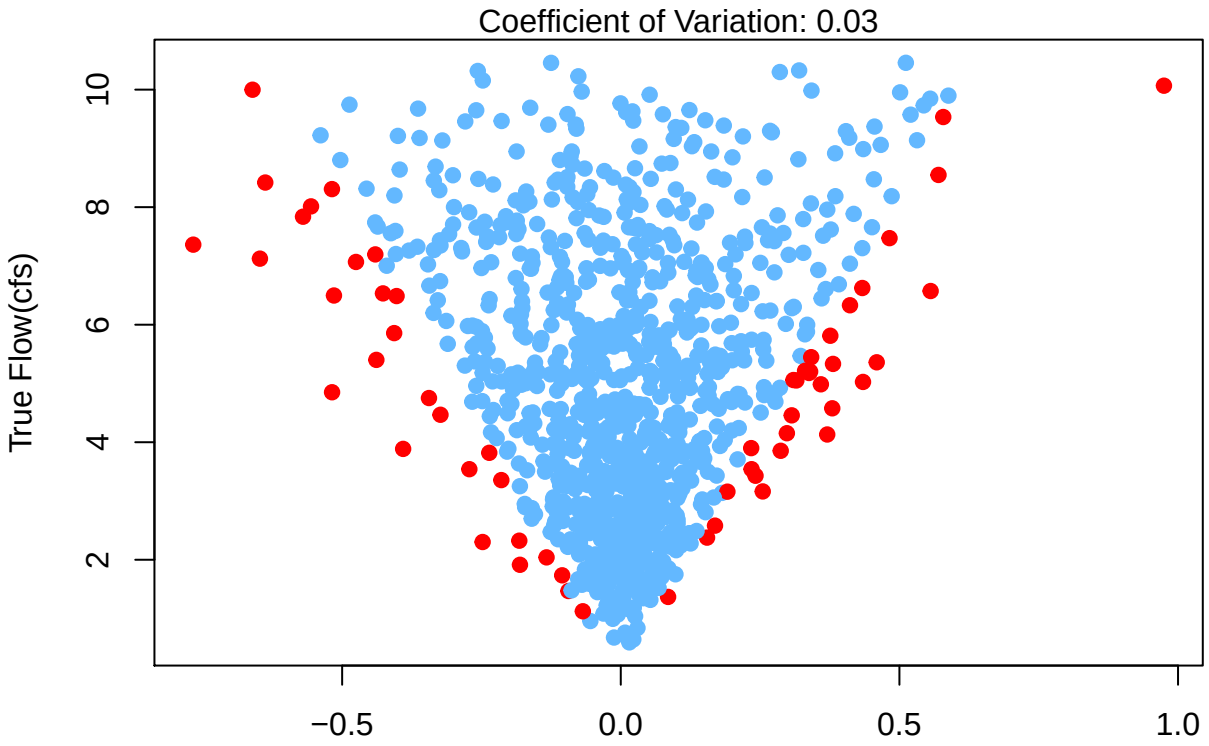
```
    bbnorm <- rnorm(nsim, modelcoef[[7]], bbstdv)

    cctrue <- modelcoef[[8]]
    ccstdv <- abs(coefofvar * modelcoef[[8]])
    ccnorm <- rnorm(nsim, modelcoef[[8]], ccstdv)

    ddtrue <- modelcoef[[9]]
    ddstdv <- abs(coefofvar * modelcoef[[9]])
    ddnorm <- rnorm(nsim, modelcoef[[9]], ddstdv)

    # note that G and H are certain here, the only source of uncertainty is in the
    # model parameters! We are sampling it from a uniform distribution, rather than
    # looping over all possible values, with a high enough nsim, this shouldn't be an
    # issue.
    guniform <- runif(nsim, min(flowdf1$G), max(flowdf1$G))
    huniform <- runif(nsim, min(flowdf1$H), max(flowdf1$H))

    flowsim <- flowtruth <- vector()
    for (i in 1:nsim) {

        # we can adjust the ability to call different functions here, but for now just
        # equation #1
        flowsim[i] <- (anorm[i] + bnorm[i] * guniform[i] + cnorm[i] * guniform[i]^2 +
            dnorm[i] * guniform[i]^3 + enorm[i] * log(huniform[i]))/(1 + aanorm[i] *
            guniform[i] + bbnorm[i] * log(huniform[i]) + ccnorm[i] * (log(huniform[i]))^2 +
            ddnorm[i] * (log(huniform[i]))^3)

        flowtruth[i] <- (atrue + btrue * guniform[i] + ctrue * guniform[i]^2 + dtrue *
            guniform[i]^3 + etrue * log(huniform[i]))/(1 + aatrue * guniform[i] +
            bbtrue * log(huniform[i]) + cctrue * (log(huniform[i]))^2 + ddtrue *
            (log(huniform[i]))^3)
    }

    flowerror <- flowtruth - flowsim

    # This code checks if all errors in flow are less than the maximum error allowed
    # (e.g., 6% of flow). This is plotted with colors to show points in and out of
    # bounds. I then choose the cv that corresponds to the results being within 6% or
    # 12% of the 'truth'. I might try to run the model with a couple different cv
    # values - one with all points in bounds, one with 98% of points in bounds, one
    # with 95% of points in bounds, etc.

    optflowerror <- optflowerrorp * flowtruth
    errorconditioncol <- ifelse(abs(flowerror) < optflowerror, 1, 0)
    # resultstomin <- nsim-sum(errorconditioncol)
    resultsdf <- cbind(FLOWTRUTH = flowtruth, FLOWSIM = flowsim, FLOWERROR = flowerror,
        OPTFLOW = optflowerror, ERRORCONDITION = errorconditioncol)
    return(resultsdf)
}

cvrange <- seq(0.01, 0.5, 0.01)
optresult <- list()
for (i in 1:length(cvrange)) {
```

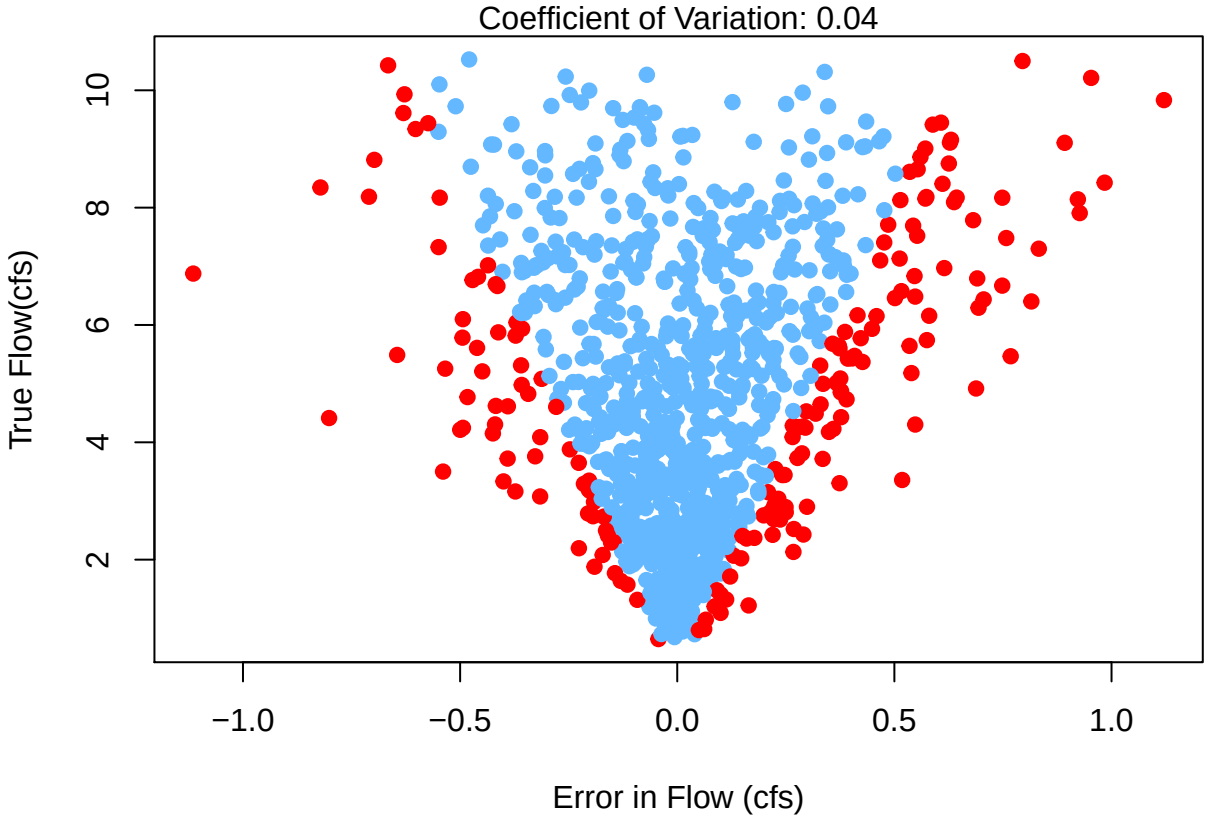
```
    cv <- cvrange[i]
    optresult[[i]] <- equationerrorsim(dataindex = di, coefofvar = cv, optflowerrorp = 0.06)
}

optresult_check <- optresult[[2]]
```

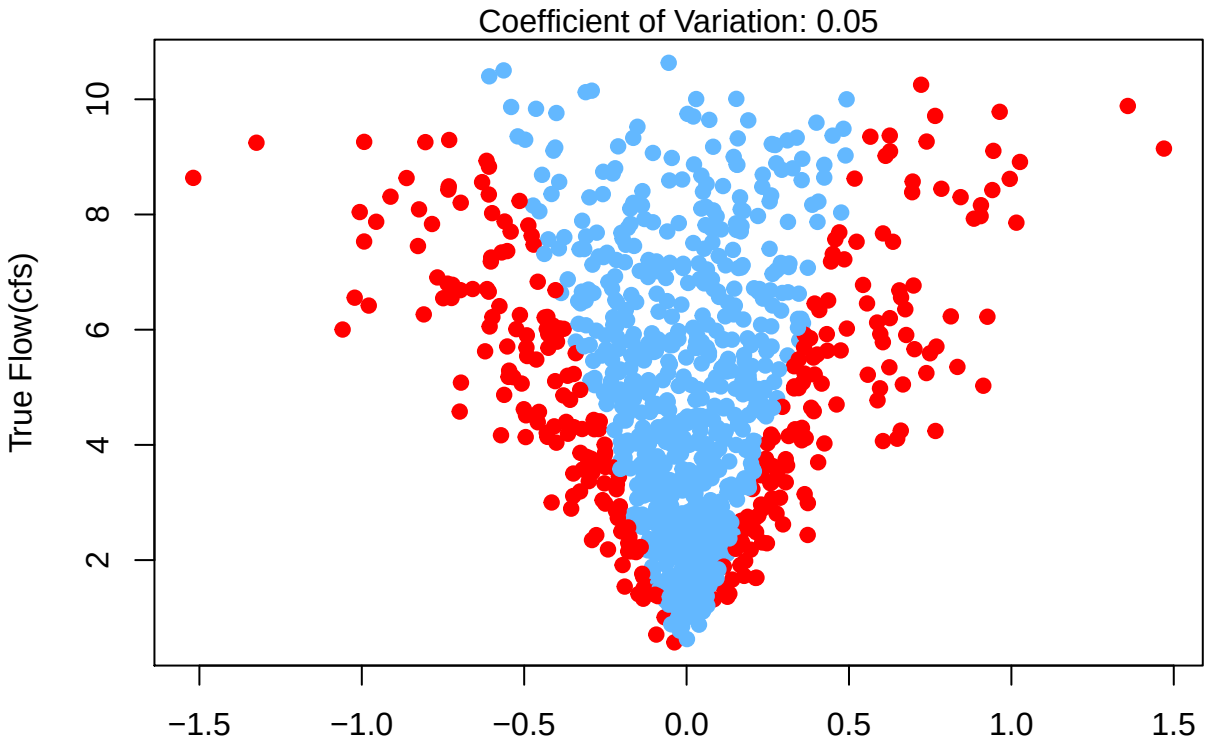
##4.2 Monte Carlo Plots. These plots show the results from the above section, with red dots for points that exceed the threshold and blue dots for points that do not exceed the threshold.

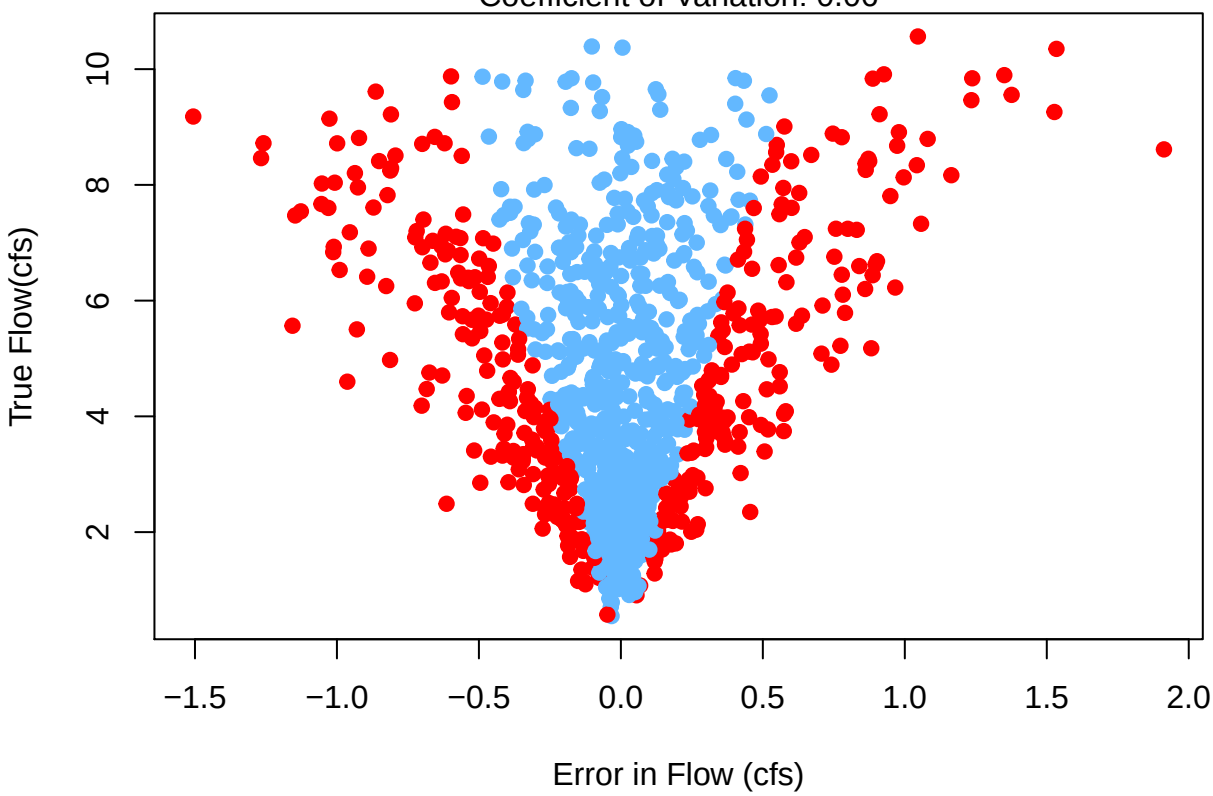
```
i <- 1
for (i in 1:length(cvrange)) {
    png(paste0("outputs/equation_error/optimization_cloud_", i, ".png"), width = 3.25,
        height = 3, units = "in", pointsize = 8, res = 1200)
    par(mar = c(4, 4, 2, 1) + 0.1)
    plot(optresult[[i]][, "FLOWERROR"], optresult[[i]][, "FLOWTRUTH"], col = c("red",
        "steelblue1")[optresult[[i]][, "ERRORCONDITION"] + 1], pch = 19, xlab = "Error in Flow (cfs)",
        ylab = "True Flow(cfs)")
    mtext(paste0("Coefficient of Variation: ", cvrange[i]), side = 3, line = 0)
    # legend('bottomleft', c('Optimal', 'Error'), pch=c(19,19), col=c('steelblue1',
    # 'red'), inset=c(0.02,0.02), cex=0.8, bg='grey96')
    dev.off()
    ###### Code for R Markdown Display, Can be Deleted
    par(mar = c(4, 4, 2, 1) + 0.1)
    plot(optresult[[i]][, "FLOWERROR"], optresult[[i]][, "FLOWTRUTH"], col = c("red",
        "steelblue1")[optresult[[i]][, "ERRORCONDITION"] + 1], pch = 19, xlab = "Error in Flow (cfs)",
        ylab = "True Flow(cfs)")
    mtext(paste0("Coefficient of Variation: ", cvrange[i]), side = 3, line = 0)
    # legend('bottomleft', c('Optimal', 'Error'), pch=c(19,19), col=c('steelblue1',
    # 'red'), inset=c(0.02,0.02), cex=0.8, bg='grey96')
    dev.print()
}
```

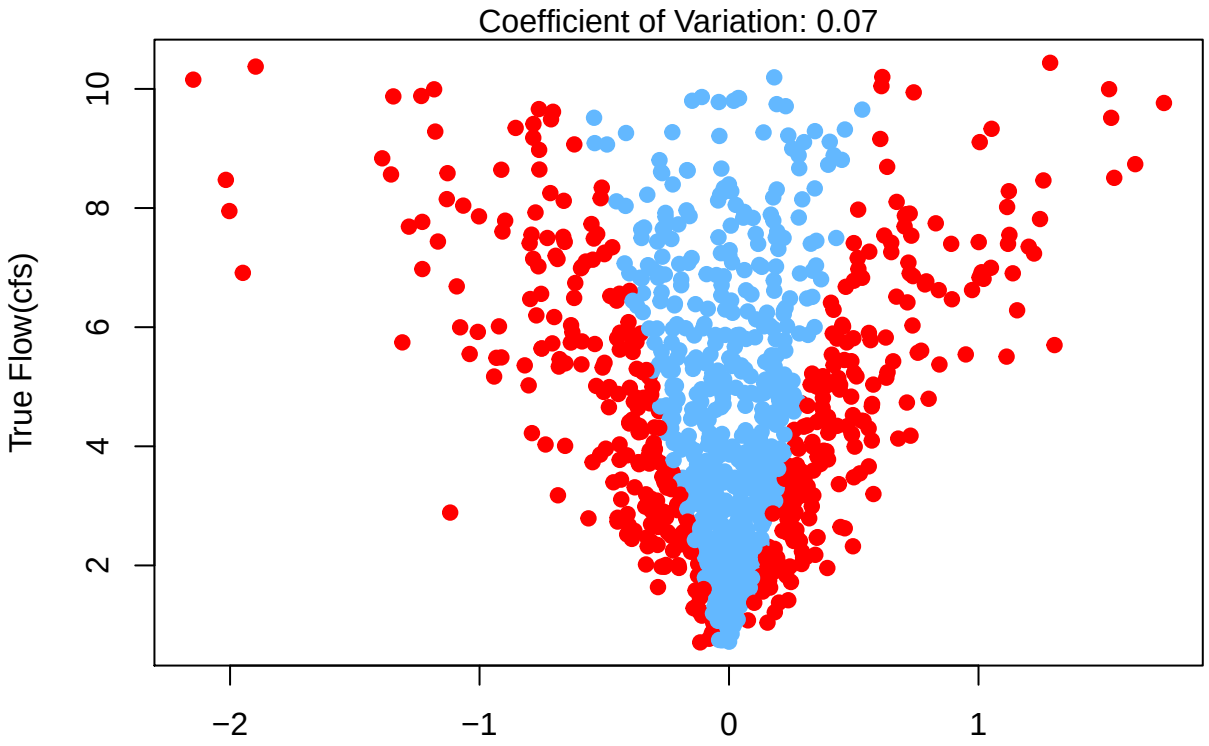

Coefficient of Variation: 0.01
True Flow(cfs)
2
4
6
8
10
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
Error in Flow (cfs)
Coefficient of Variation: 0.02
True Flow(cfs)
2
4
6
8
10
−0.4
−0.2
0.0
0.2
0.4
0.6
Error in Flow (cfs)

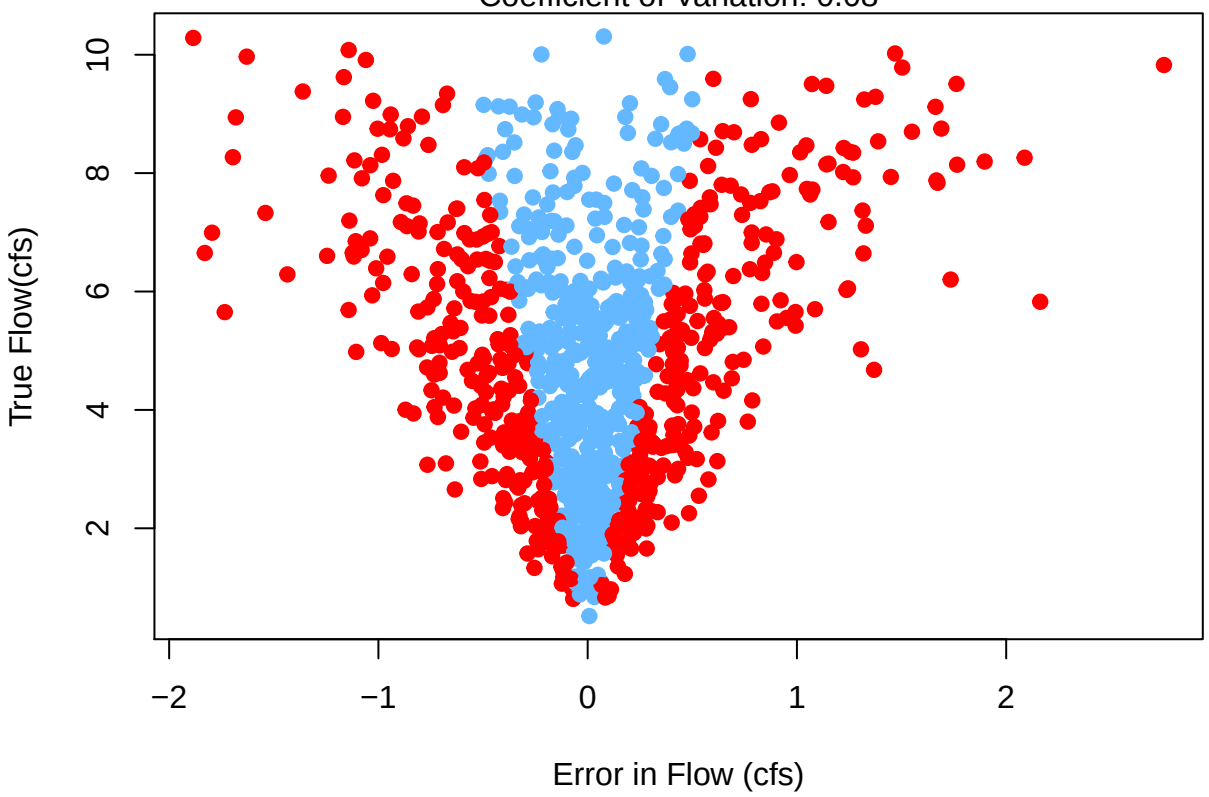

Coefficient of Variation: 0.03
True Flow(cfs)
2
4
6
8
10
−0.5
0.0
0.5
1.0
Error in Flow (cfs)

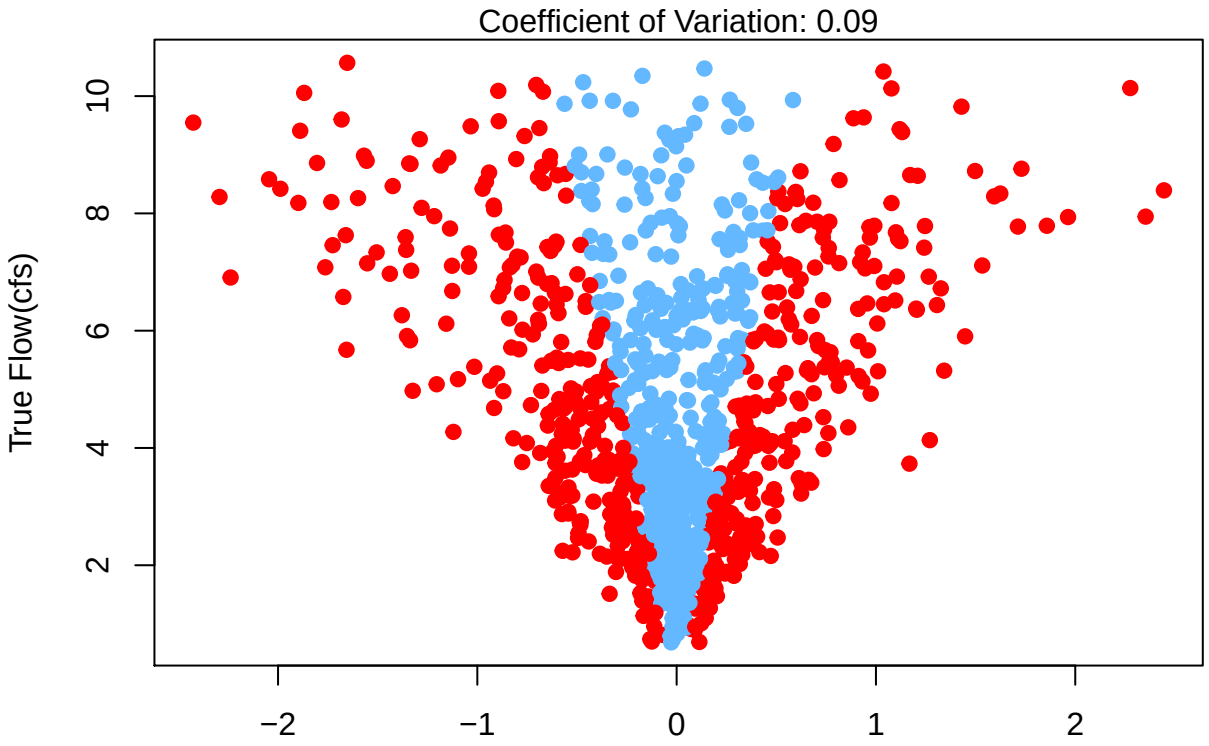

Coefficient of Variation: 0.04
True Flow(cfs)
2
4
6
8
10
−1.0
−0.5
0.0
0.5
1.0
Error in Flow (cfs)

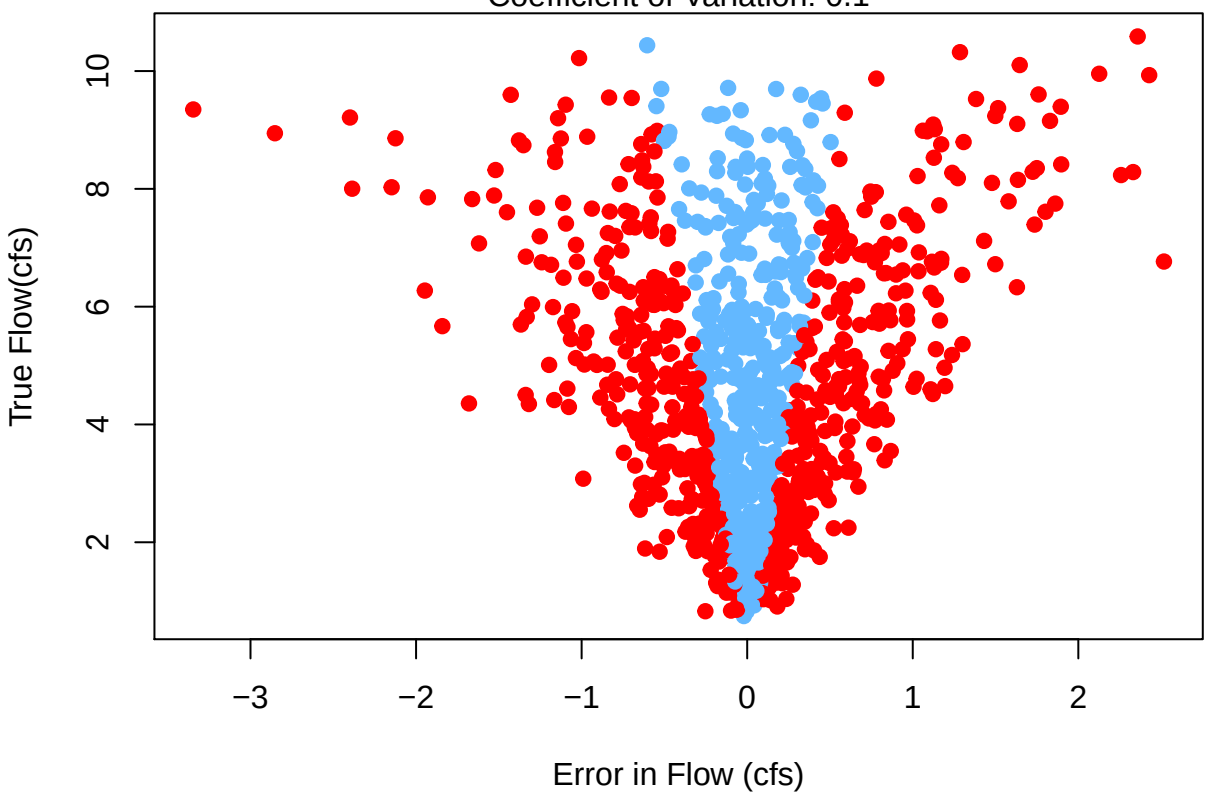

Coefficient of Variation: 0.05
True Flow(cfs)
2
4
6
8
10
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Error in Flow (cfs)

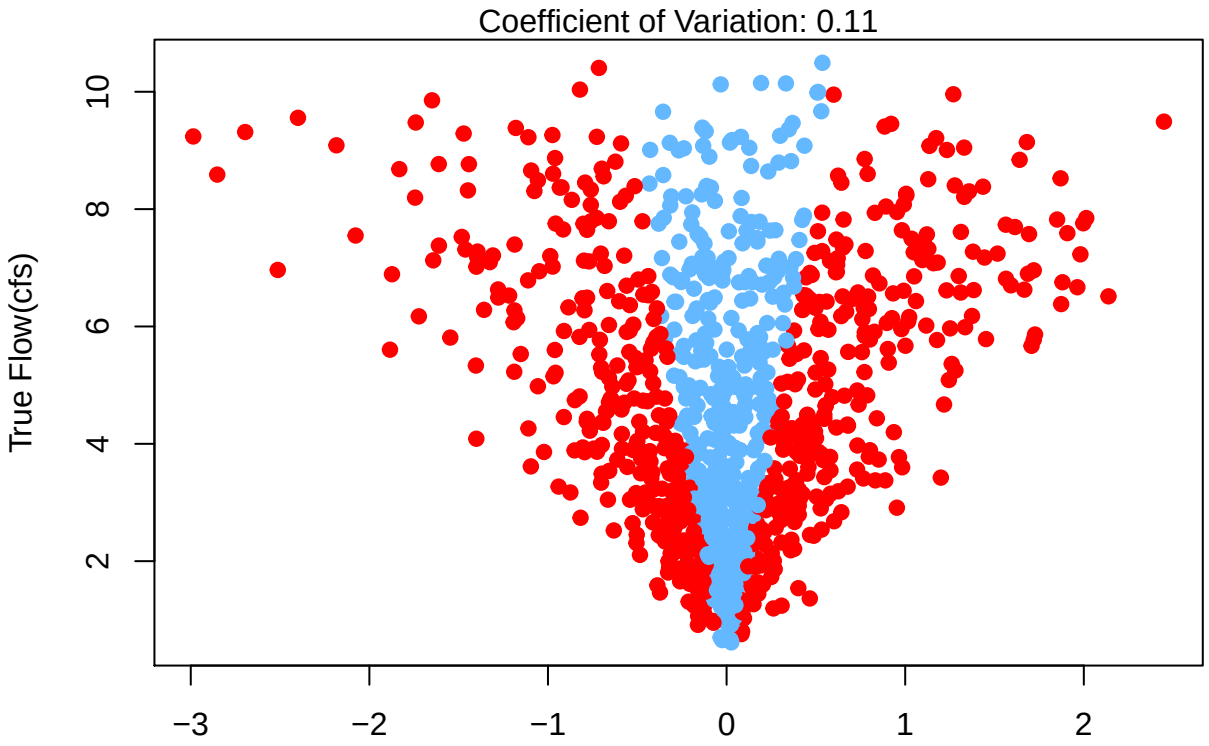

Coefficient of Variation: 0.06
True Flow(cfs)
2
4
6
8
10
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Error in Flow (cfs)

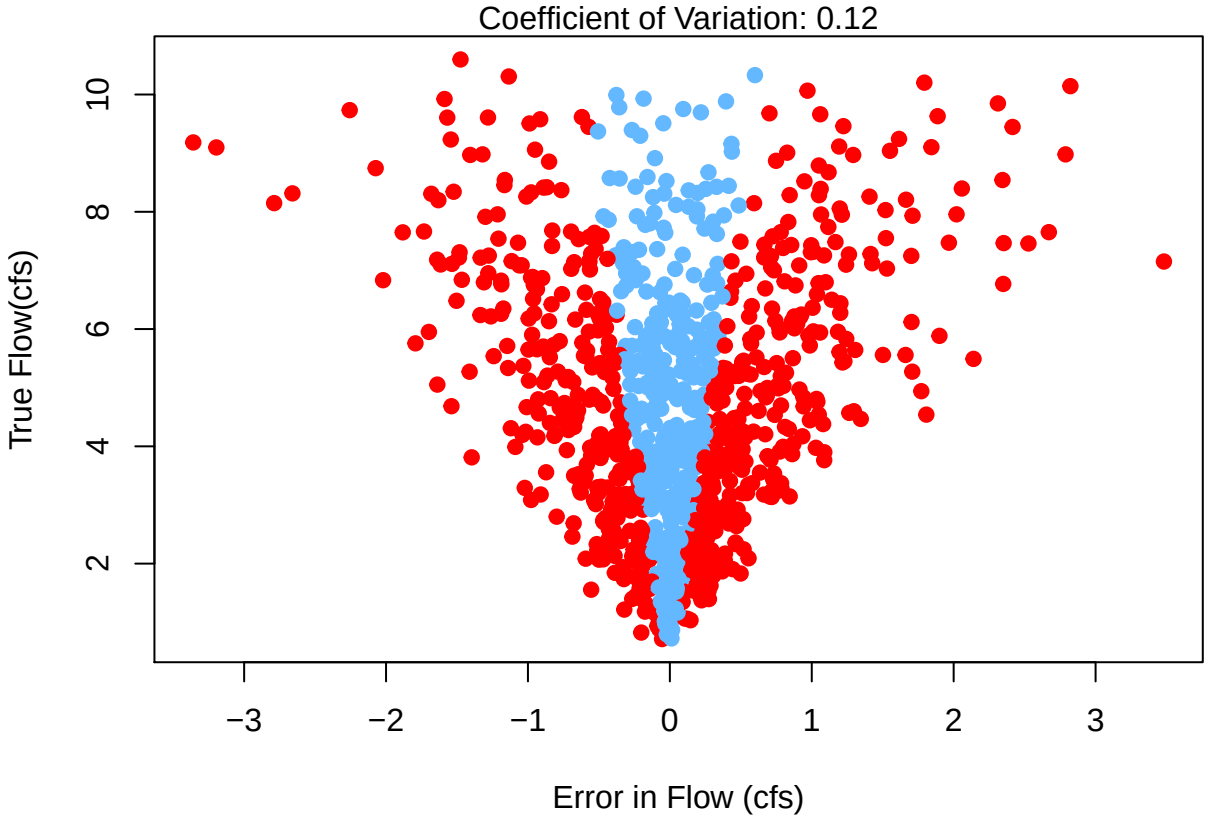

Coefficient of Variation: 0.07
True Flow(cfs)
2
4
6
8
10
−2
−1
0
1
Error in Flow (cfs)

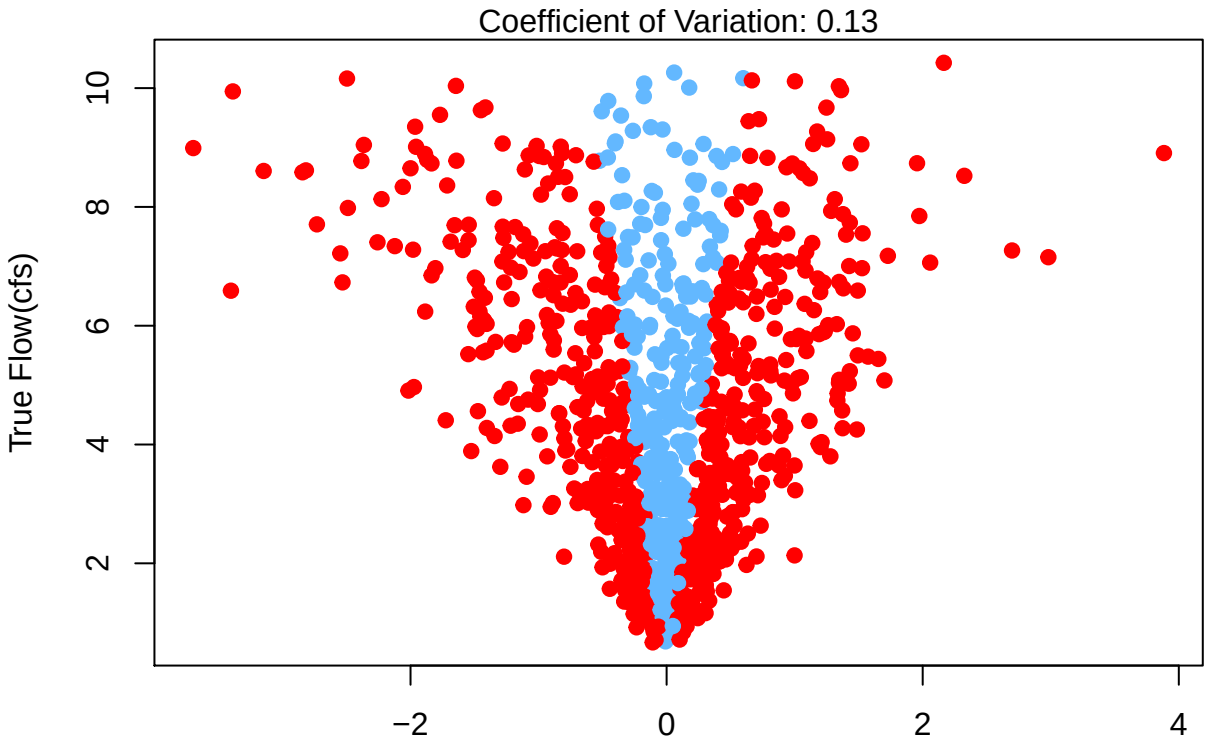

Coefficient of Variation: 0.08
True Flow(cfs)
2
4
6
8
10
−2
−1
0
1
2
Error in Flow (cfs)

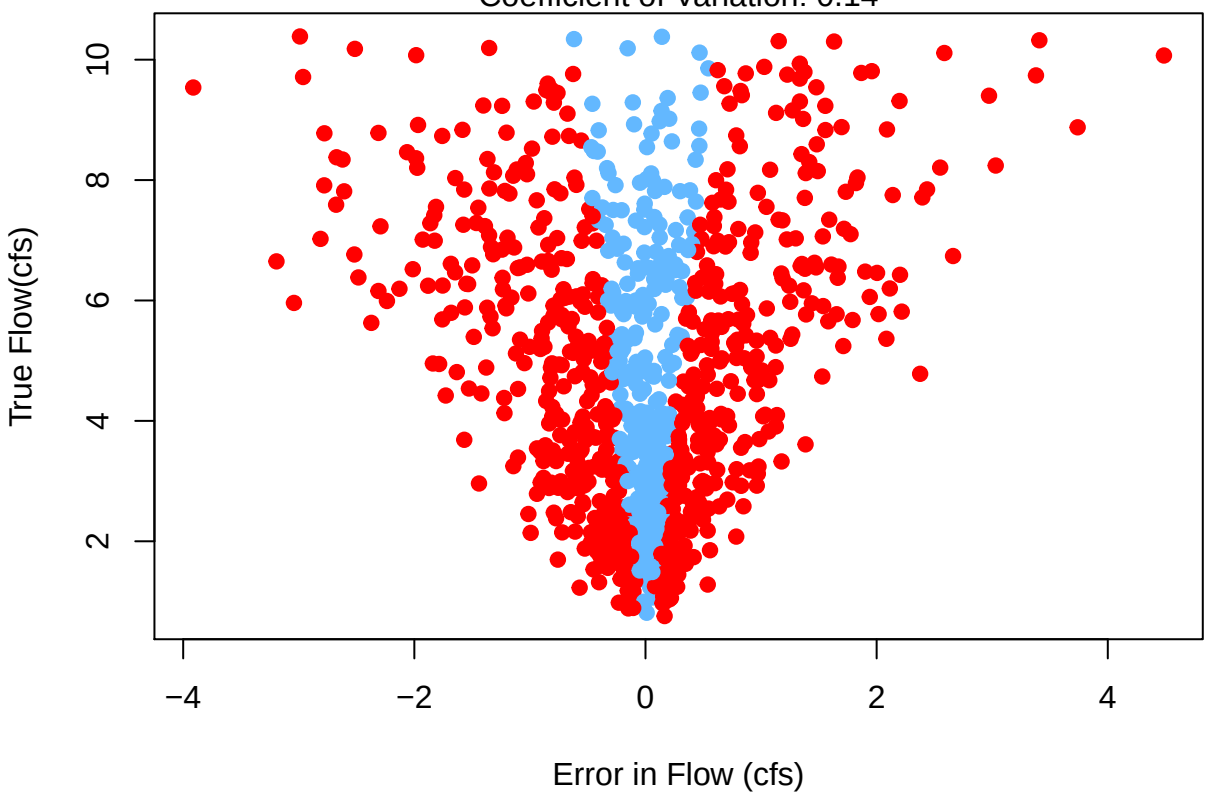

Coefficient of Variation: 0.09
True Flow(cfs)
2
4
6
8
10
−2
−1
0
1
2
Error in Flow (cfs)

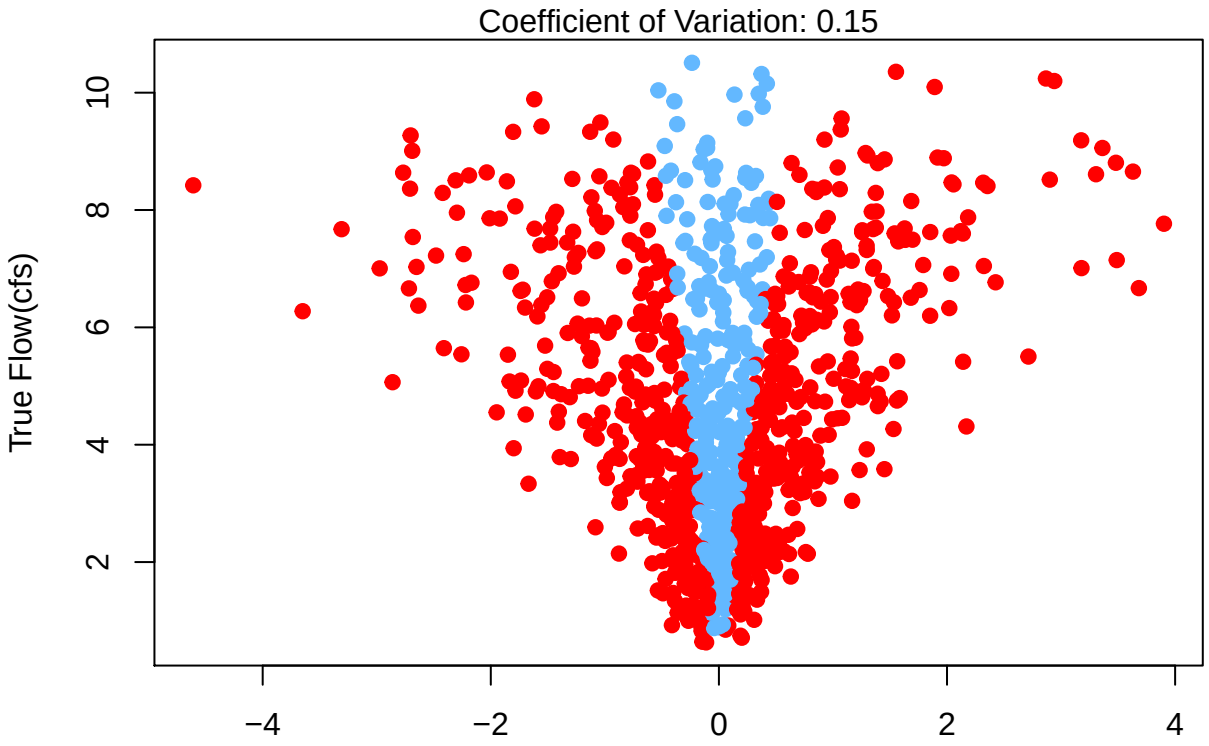

Coefficient of Variation: 0.1
True Flow(cfs)
2
4
6
8
10
−3
−2
−1
0
1
2
Error in Flow (cfs)

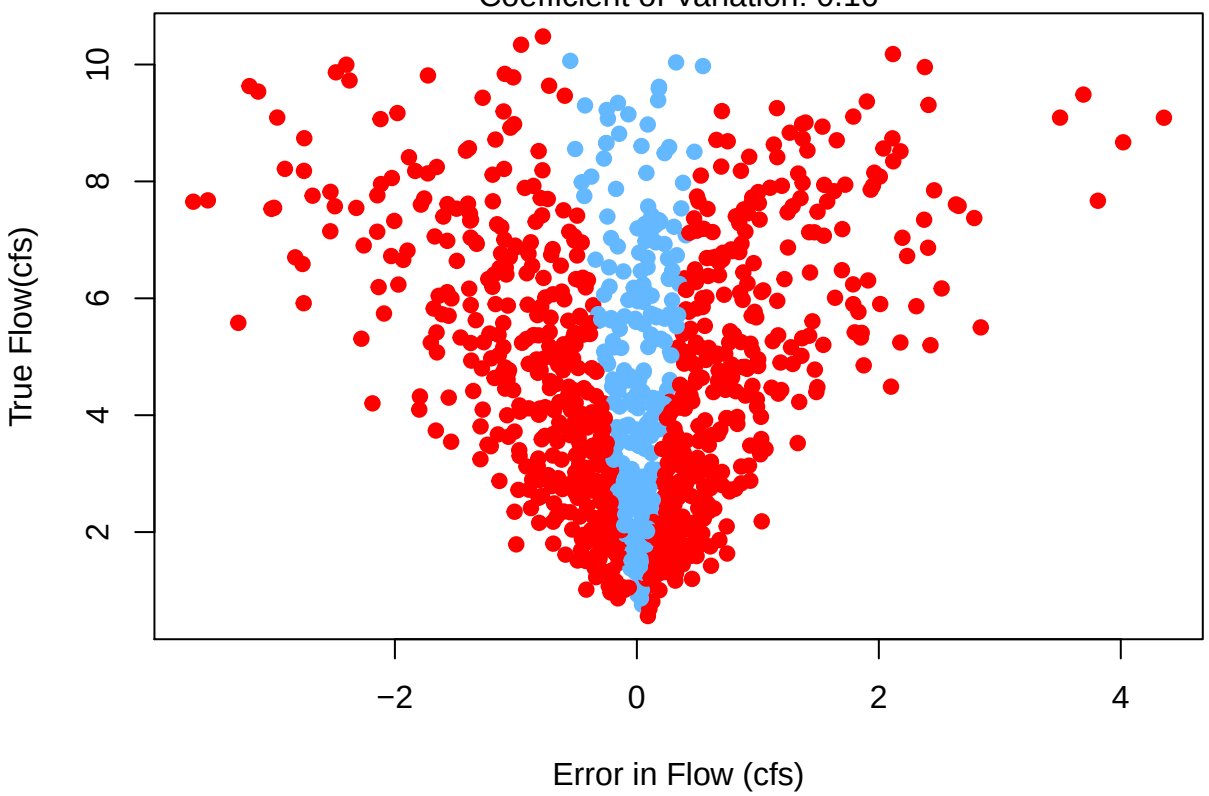

Coefficient of Variation: 0.11
True Flow(cfs)
2
4
6
8
10
−3
−2
−1
0
1
2
Error in Flow (cfs)

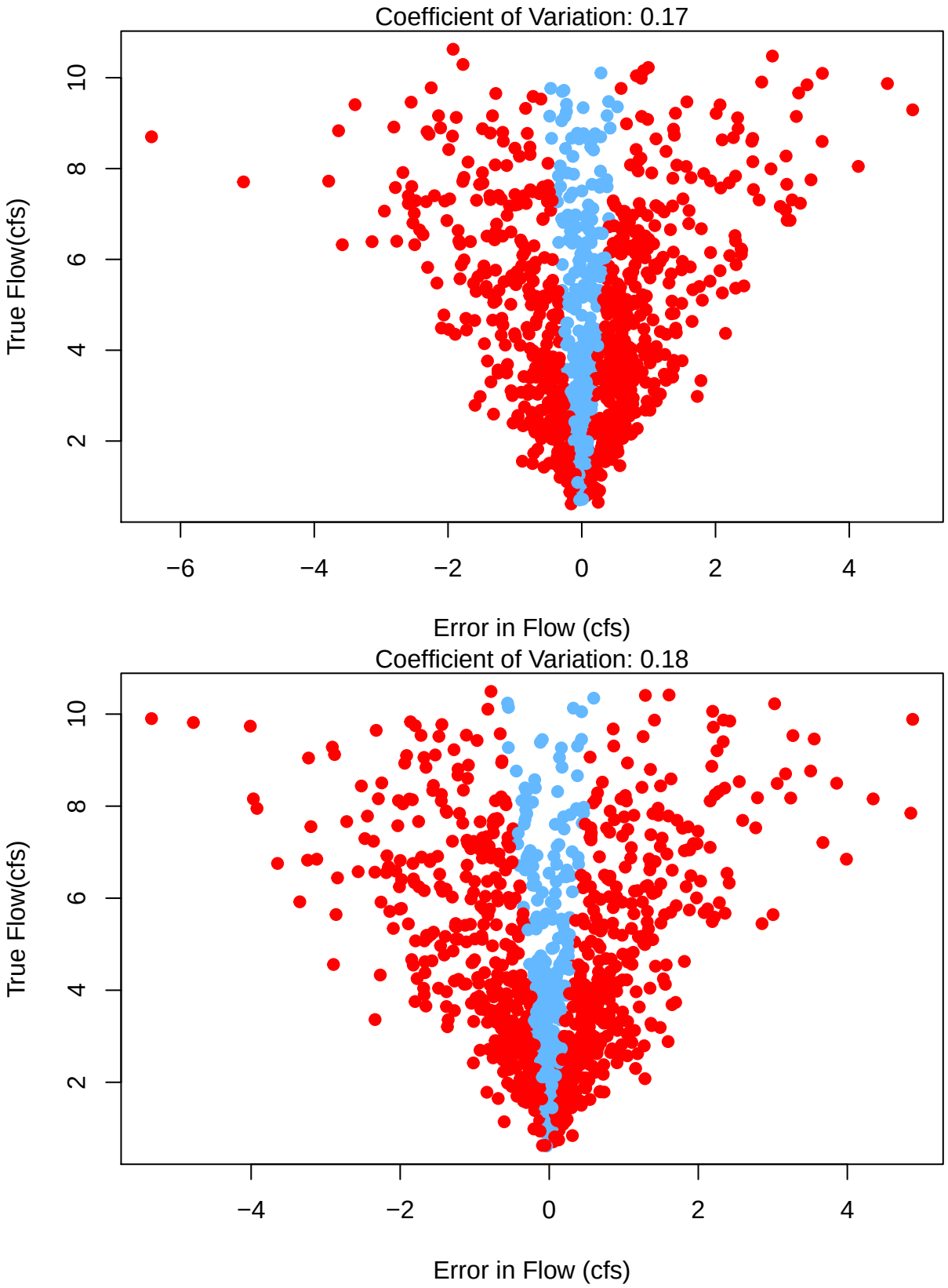

Coefficient of Variation: 0.12
True Flow(cfs)
2
4
6
8
10
−3
−2
−1
0
1
2
3
Error in Flow (cfs)

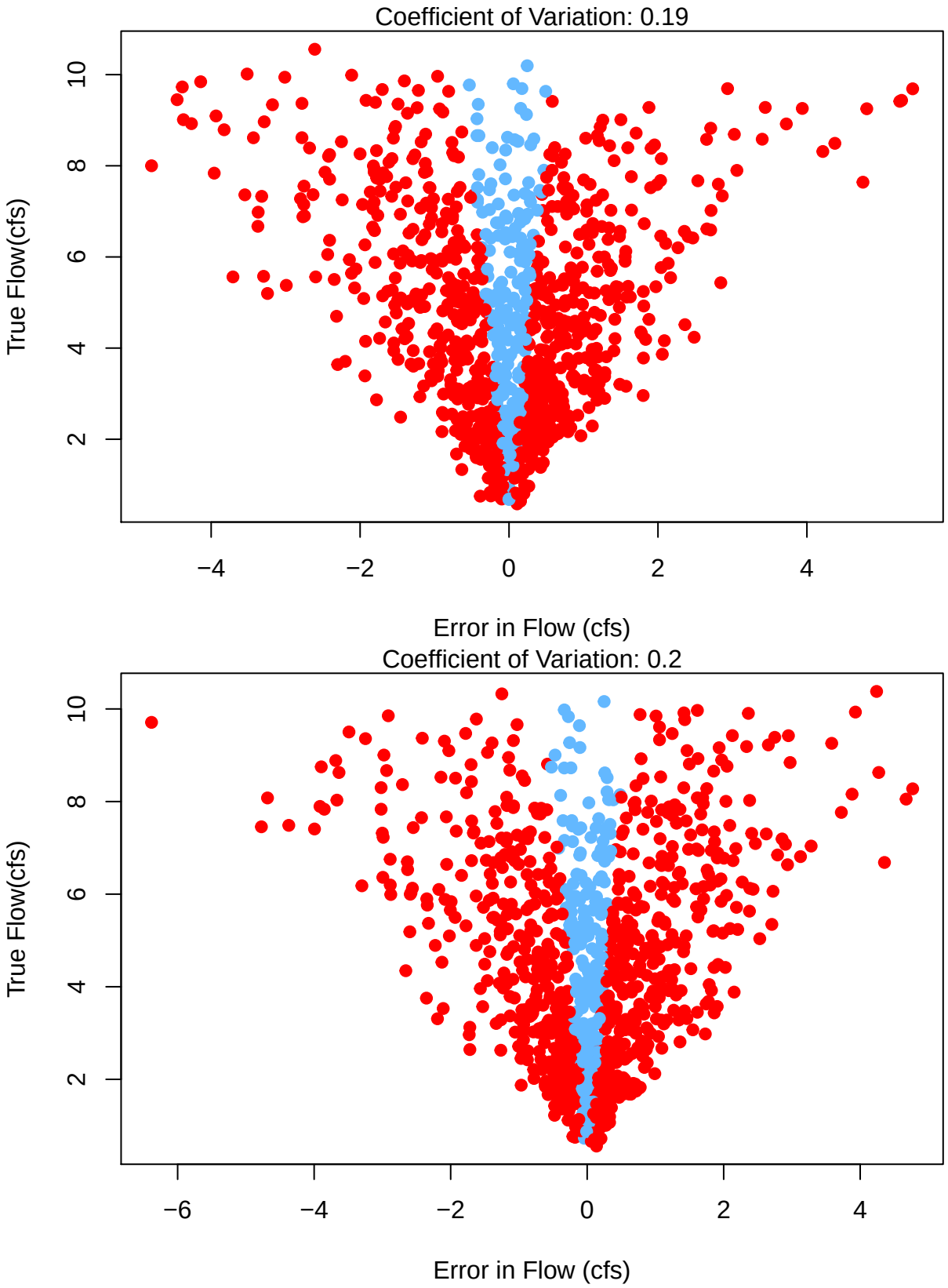

Coefficient of Variation: 0.13
True Flow(cfs)
2
4
6
8
10
−2
0
2
4
Error in Flow (cfs)

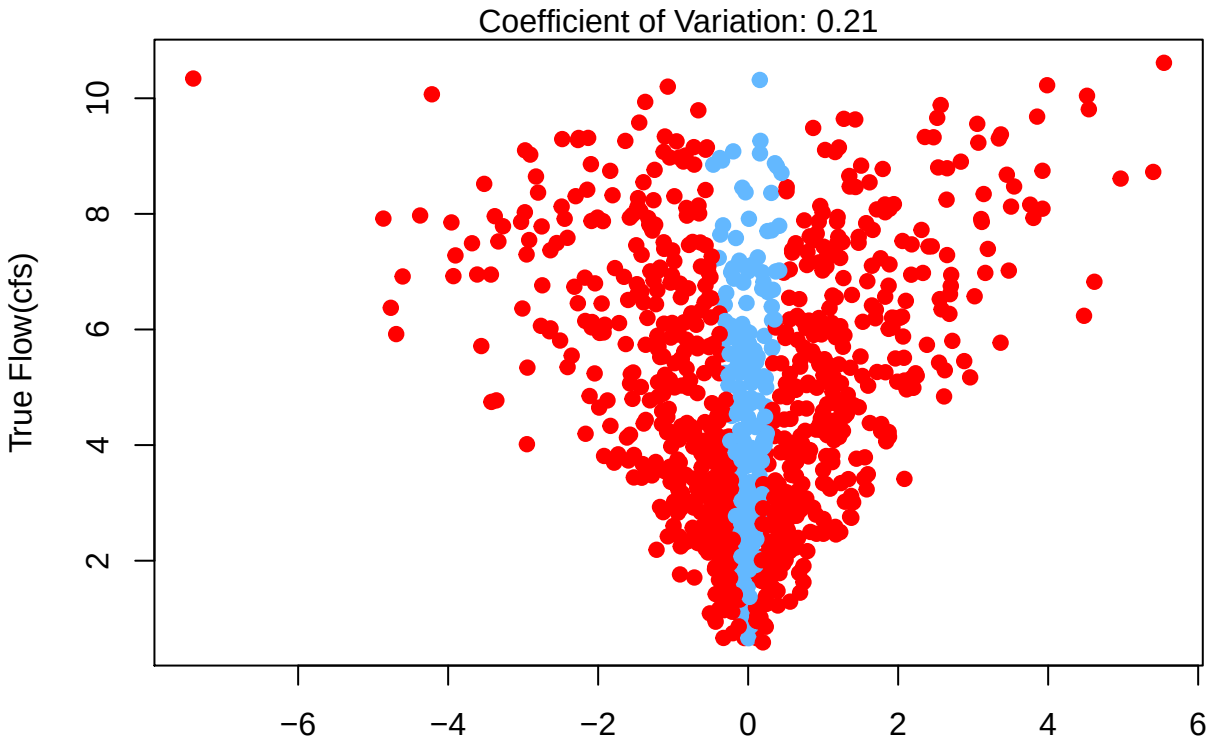

Coefficient of Variation: 0.14
True Flow(cfs)
2
4
6
8
10
−4
−2
0
2
4
Error in Flow (cfs)

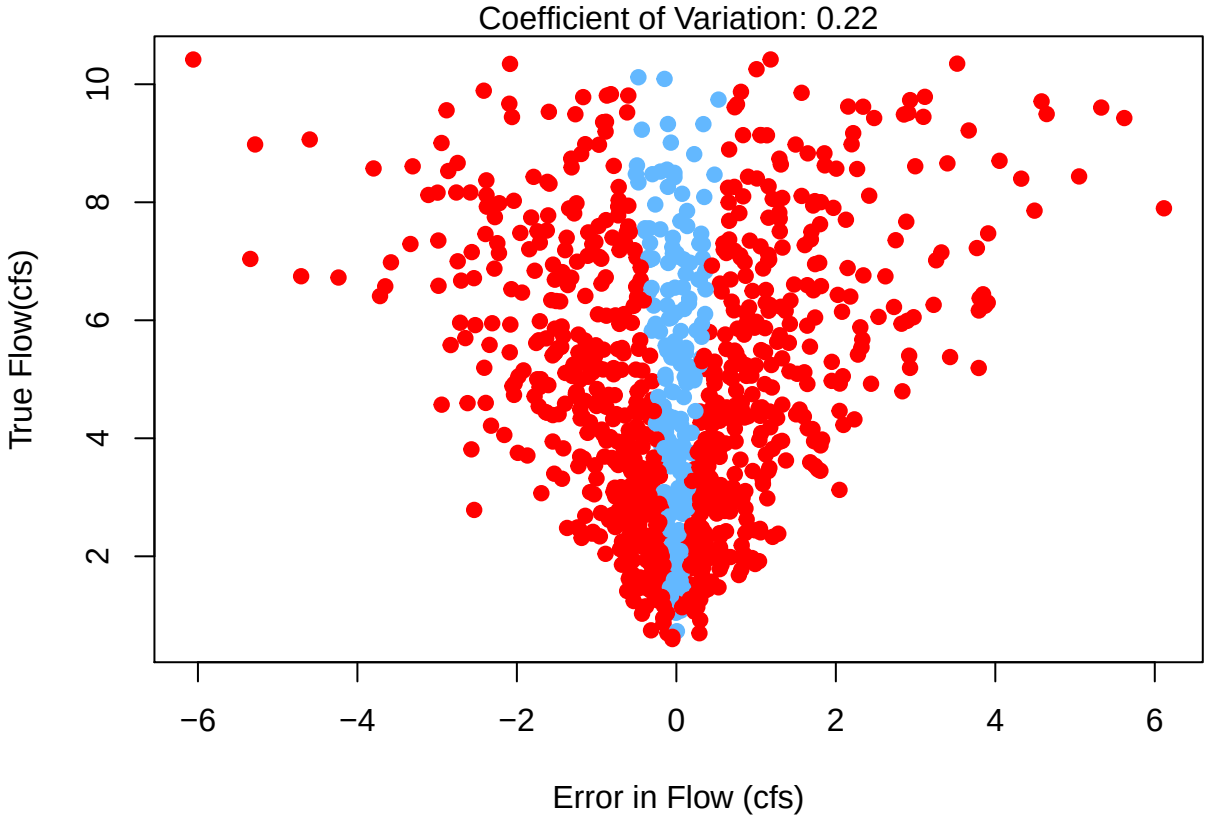

Coefficient of Variation: 0.15
True Flow(cfs)
2
4
6
8
10
−4
−2
0
2
4
Error in Flow (cfs)

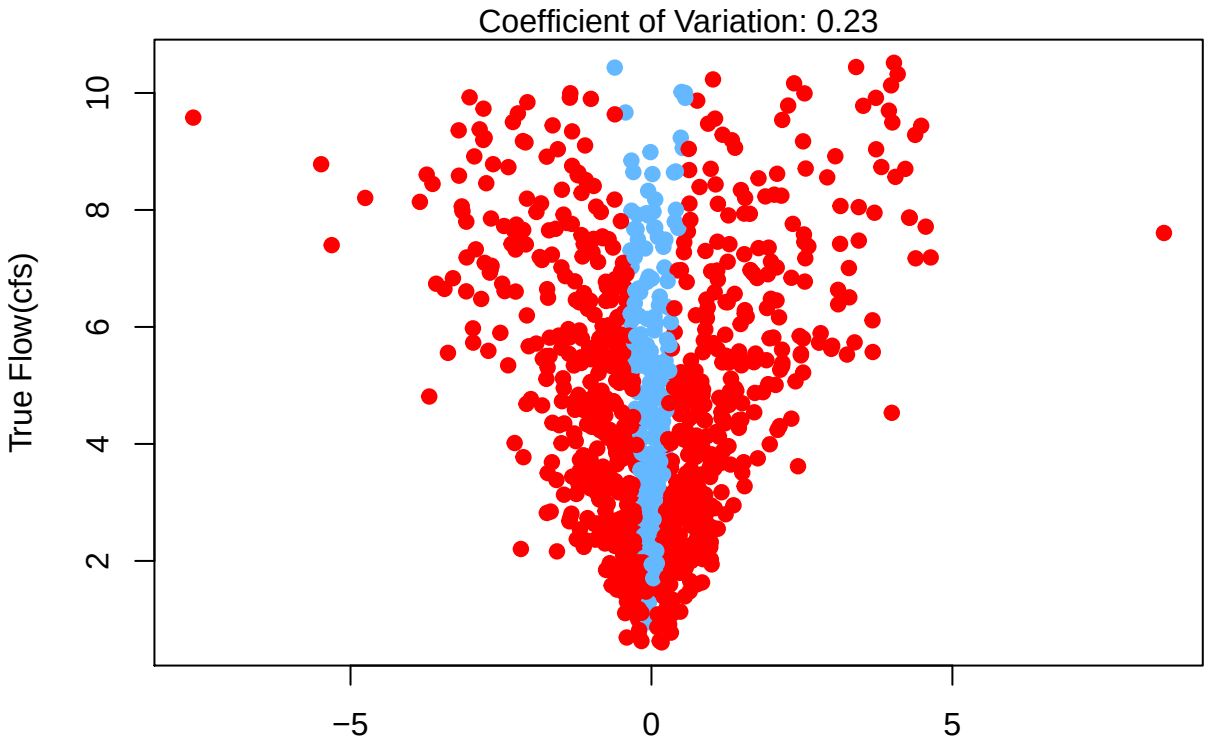

Coefficient of Variation: 0.16
True Flow(cfs)
2
4
6
8
10
−2
0
2
4
Error in Flow (cfs)

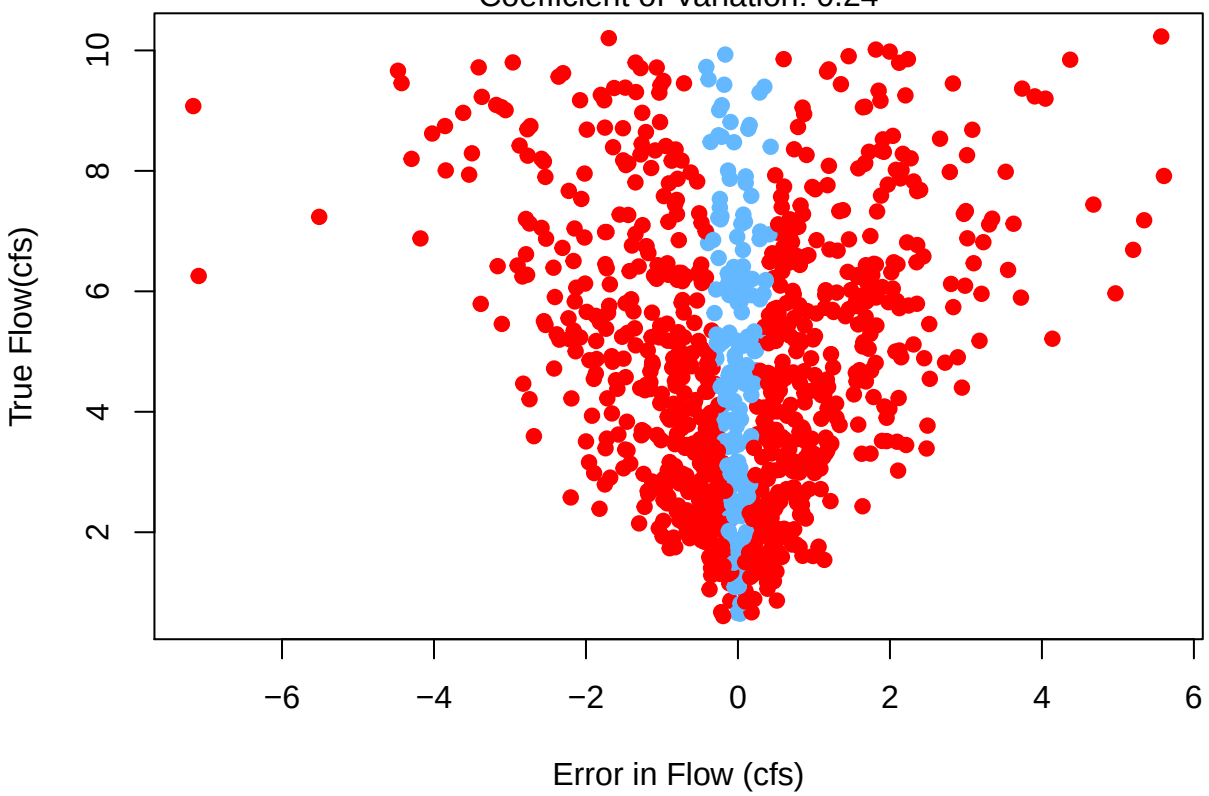

Coefficient of Variation: 0.17
True Flow(cfs)
2
4
6
8
10
−6
−4
−2
0
2
4
Error in Flow (cfs)
Coefficient of Variation: 0.18
True Flow(cfs)
2
4
6
8
10
−4
−2
0
2
4
Error in Flow (cfs)

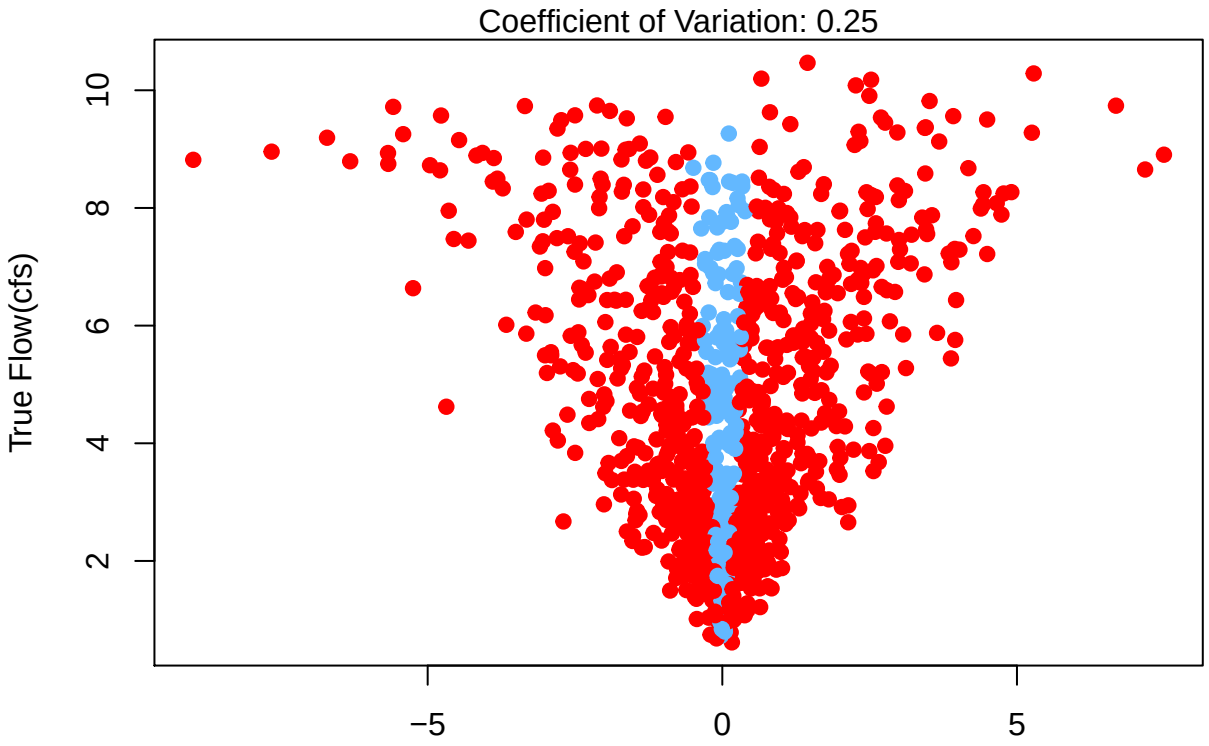

Coefficient of Variation: 0.19
True Flow(cfs)
2
4
6
8
10
−4
−2
0
2
4
Error in Flow (cfs)
Coefficient of Variation: 0.2
True Flow(cfs)
2
4
6
8
10
−6
−4
−2
0
2
4
Error in Flow (cfs)

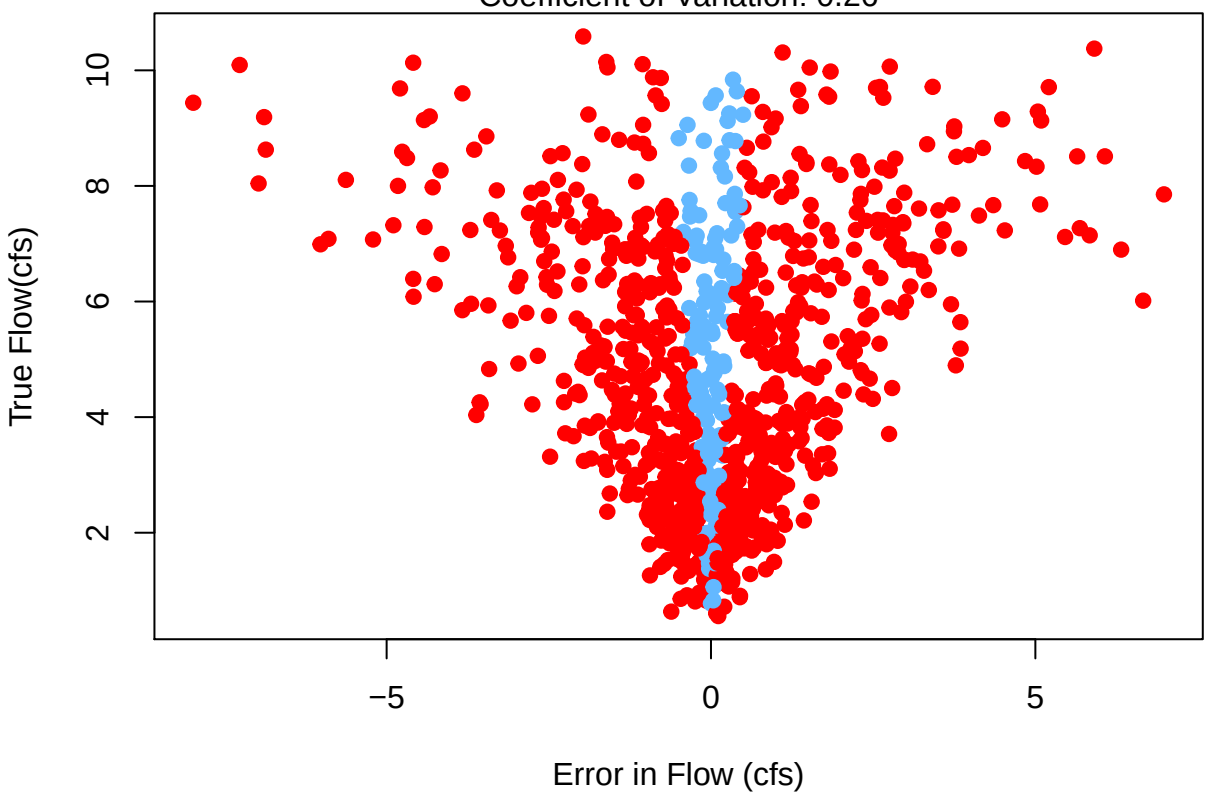

Coefficient of Variation: 0.21
True Flow(cfs)
2
4
6
8
10
−6
−4
−2
0
2
4
6
Error in Flow (cfs)

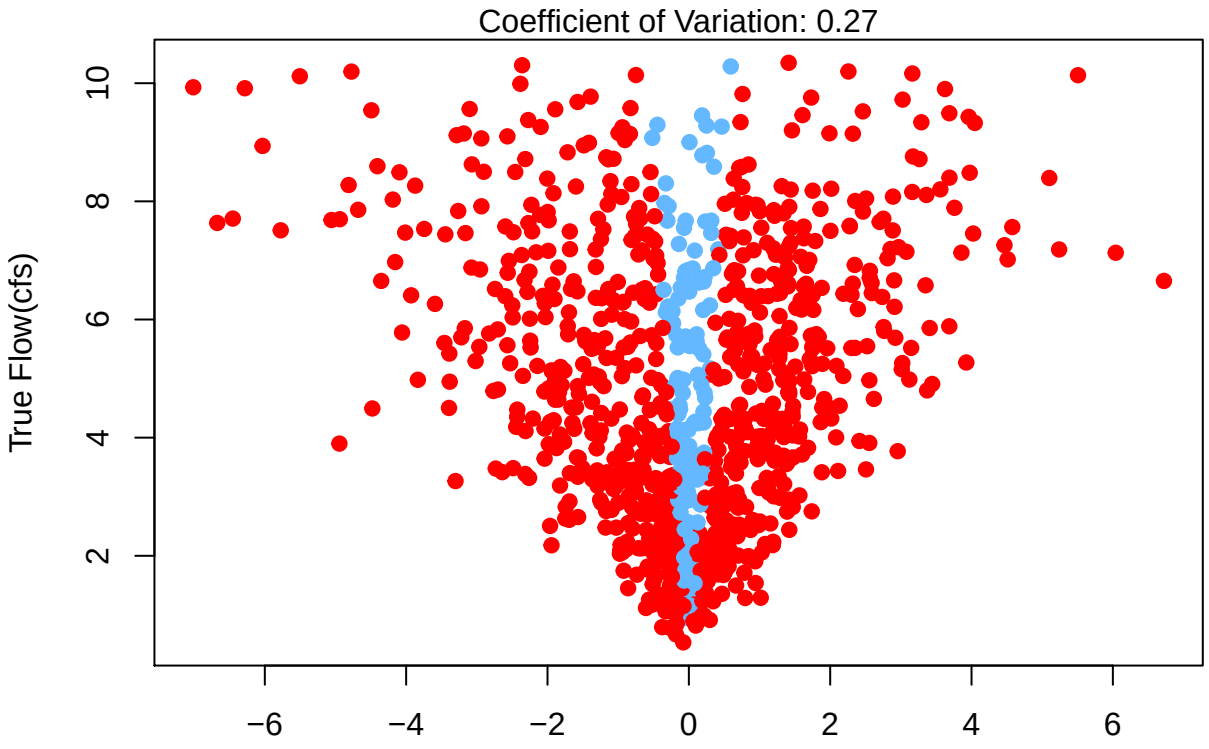

Coefficient of Variation: 0.22
True Flow(cfs)
2
4
6
8
10
−6
−4
−2
0
2
4
6
Error in Flow (cfs)

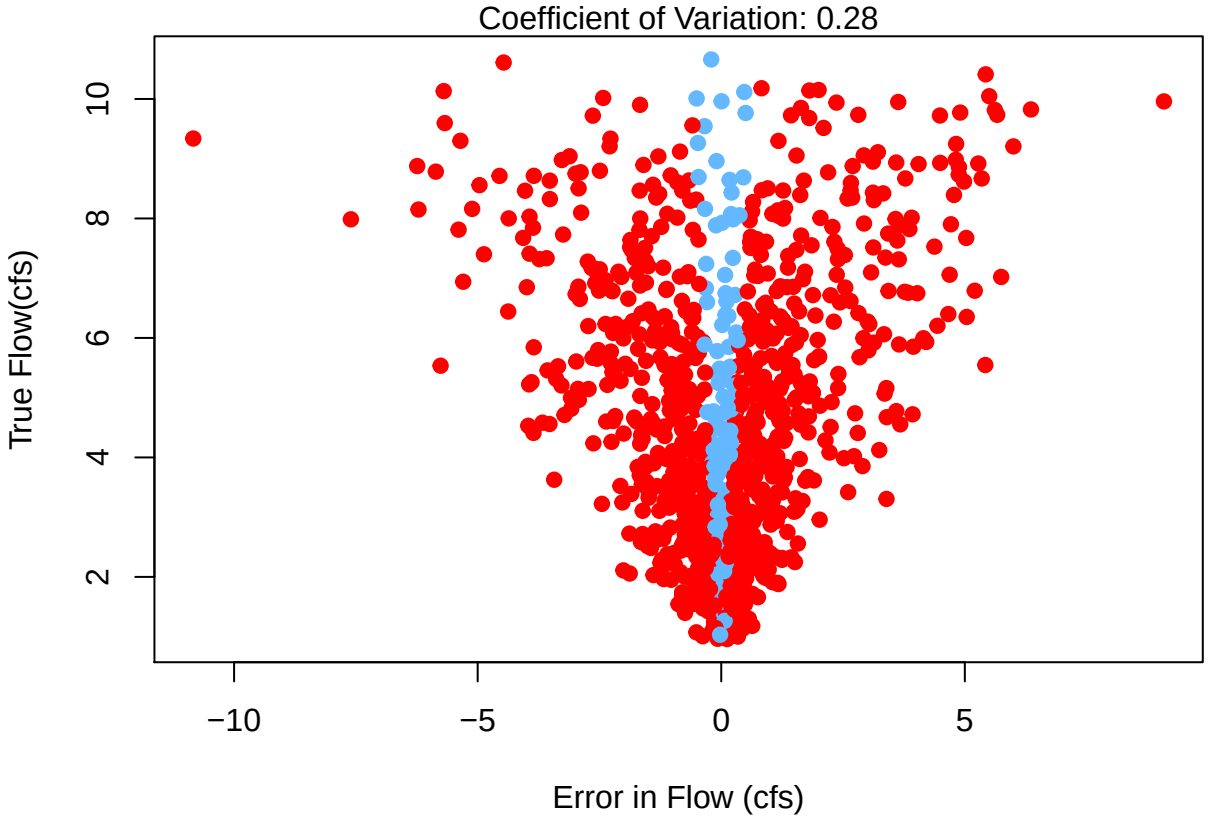

Coefficient of Variation: 0.23
True Flow(cfs)
2
4
6
8
10
−5
0
5
Error in Flow (cfs)

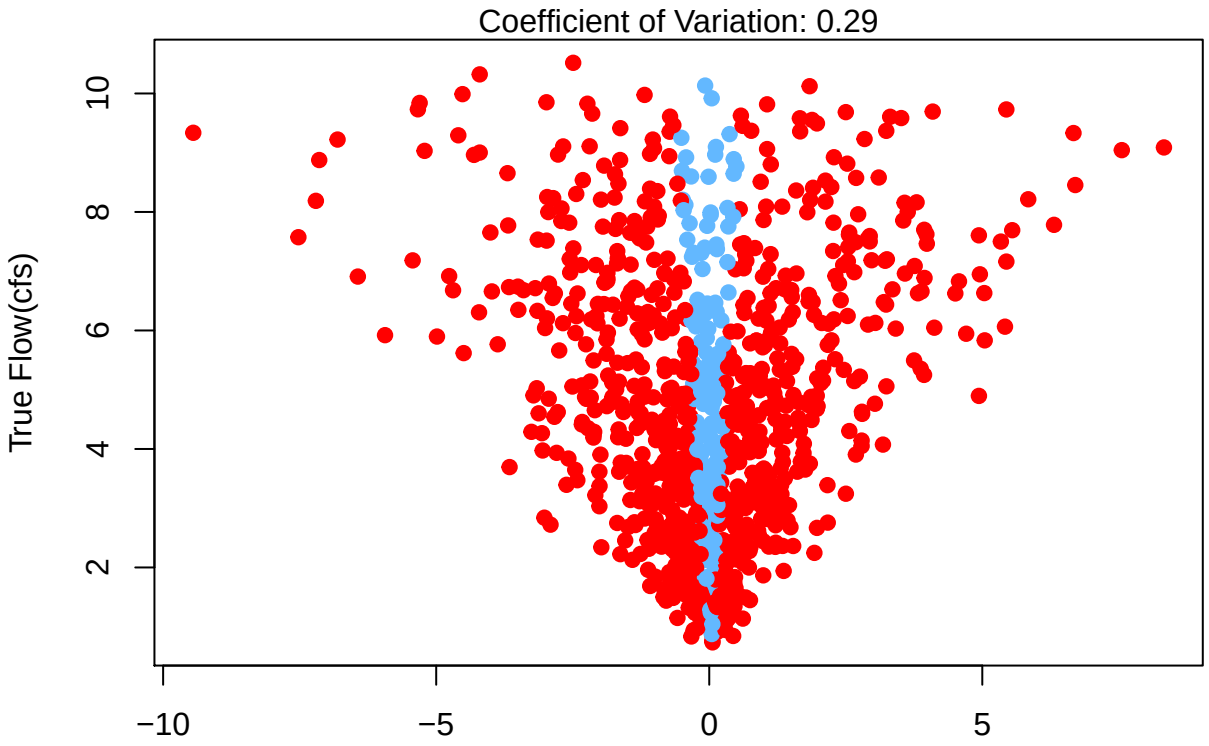

Coefficient of Variation: 0.24
True Flow(cfs)
2
4
6
8
10
−6
−4
−2
0
2
4
6
Error in Flow (cfs)

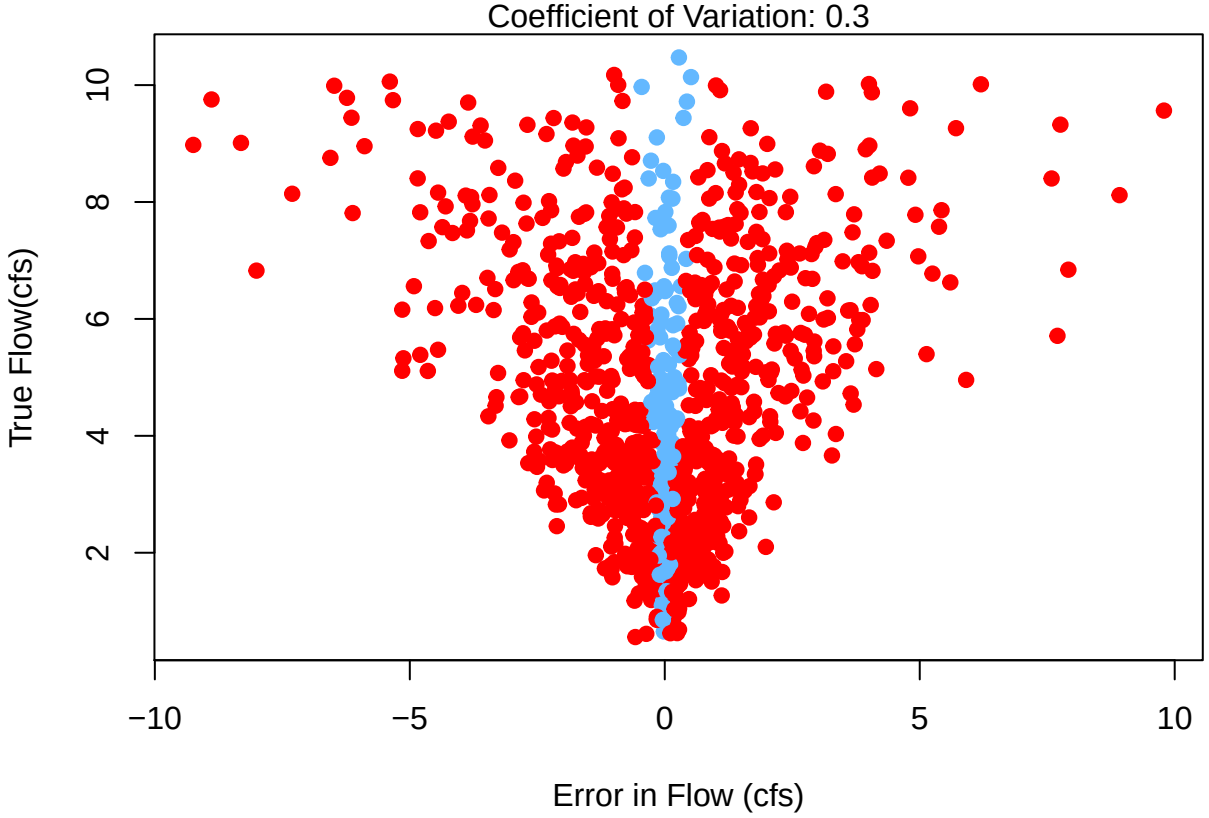

Coefficient of Variation: 0.25
True Flow(cfs)
2
4
6
8
10
−5
0
5
Error in Flow (cfs)

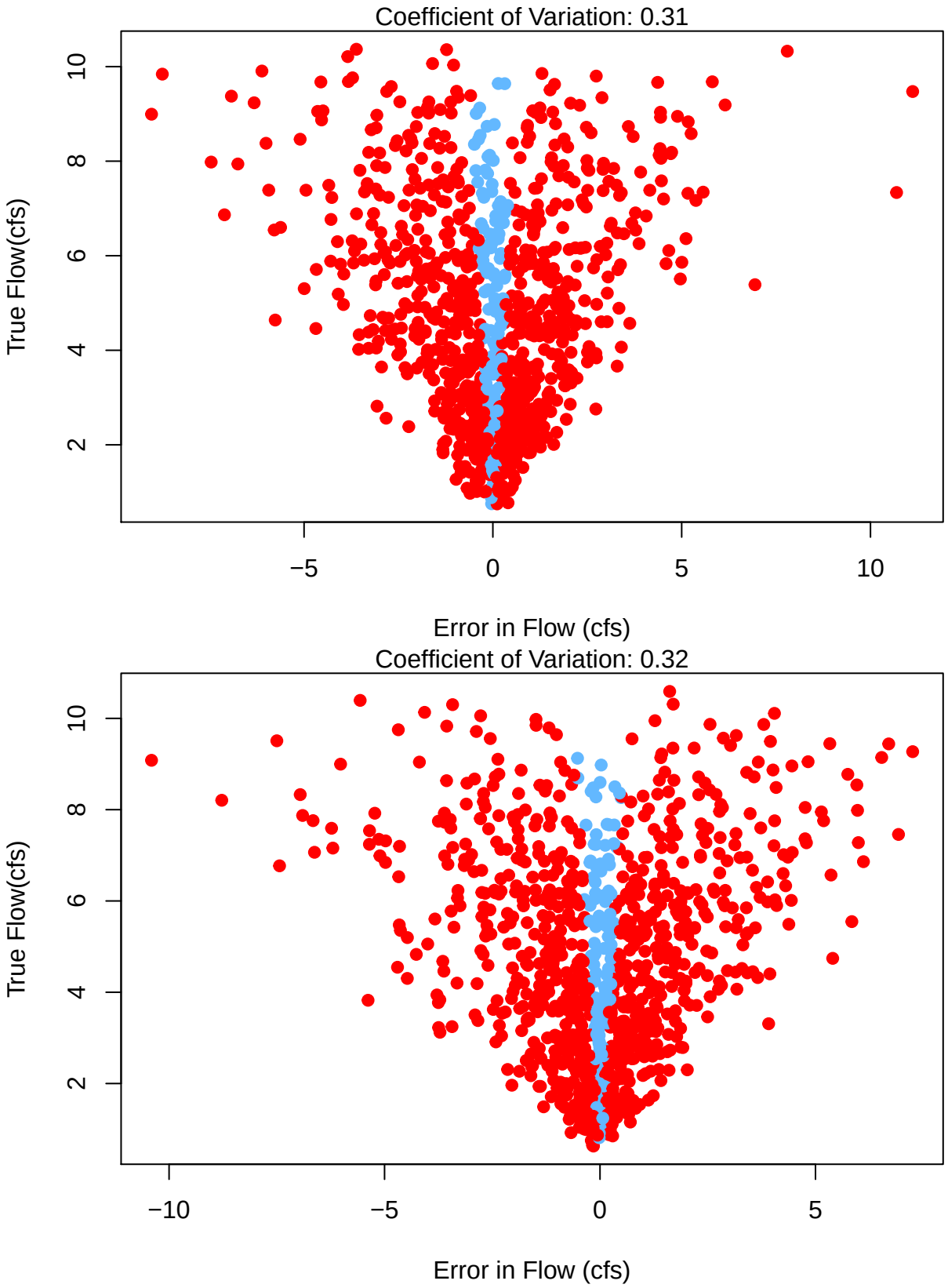

Coefficient of Variation: 0.26
True Flow(cfs)
2
4
6
8
10
−5
0
5
Error in Flow (cfs)

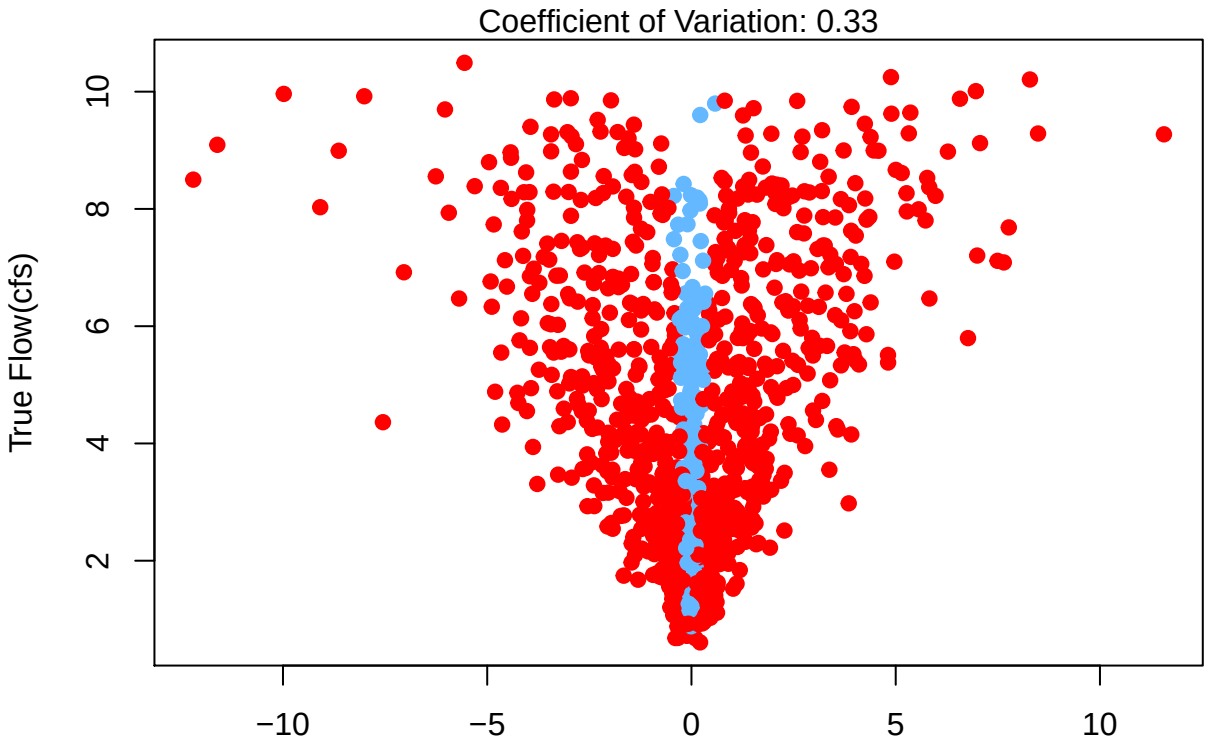

Coefficient of Variation: 0.27
True Flow(cfs)
2
4
6
8
10
−6
−4
−2
0
2
4
6
Error in Flow (cfs)

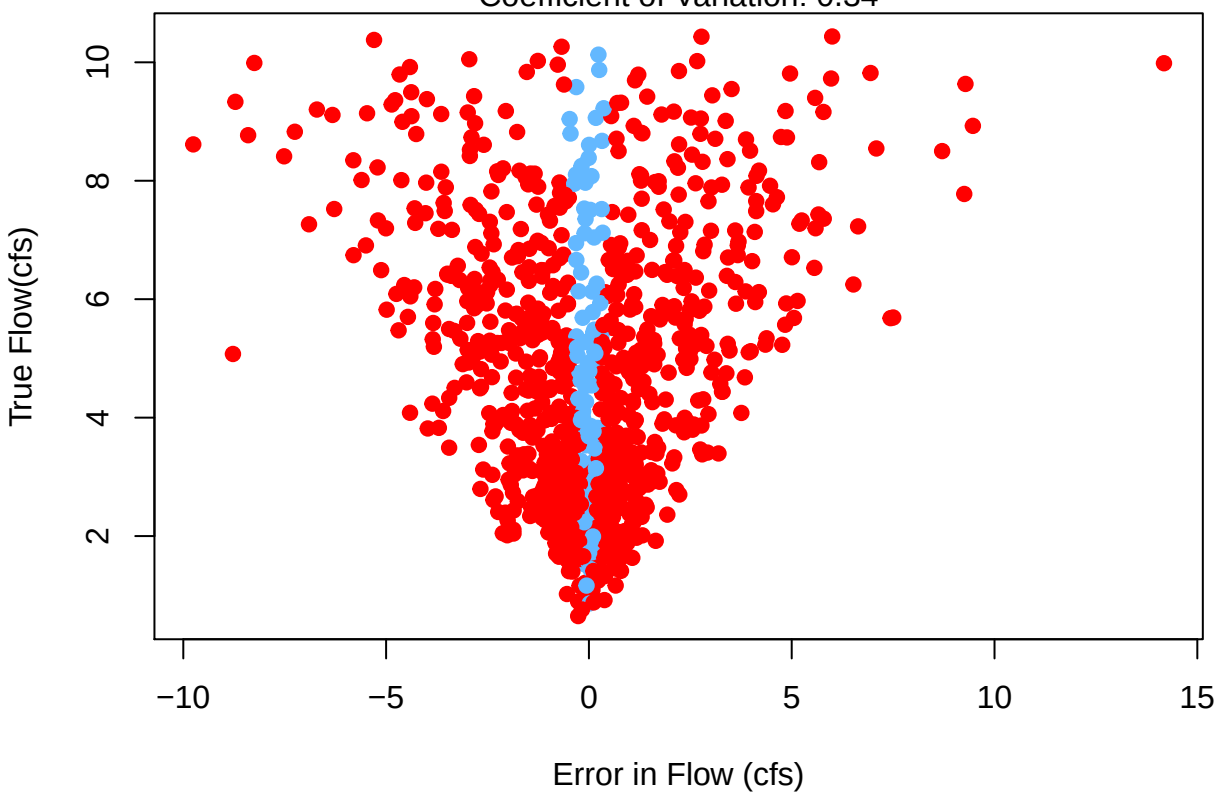

Coefficient of Variation: 0.28
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
Error in Flow (cfs)

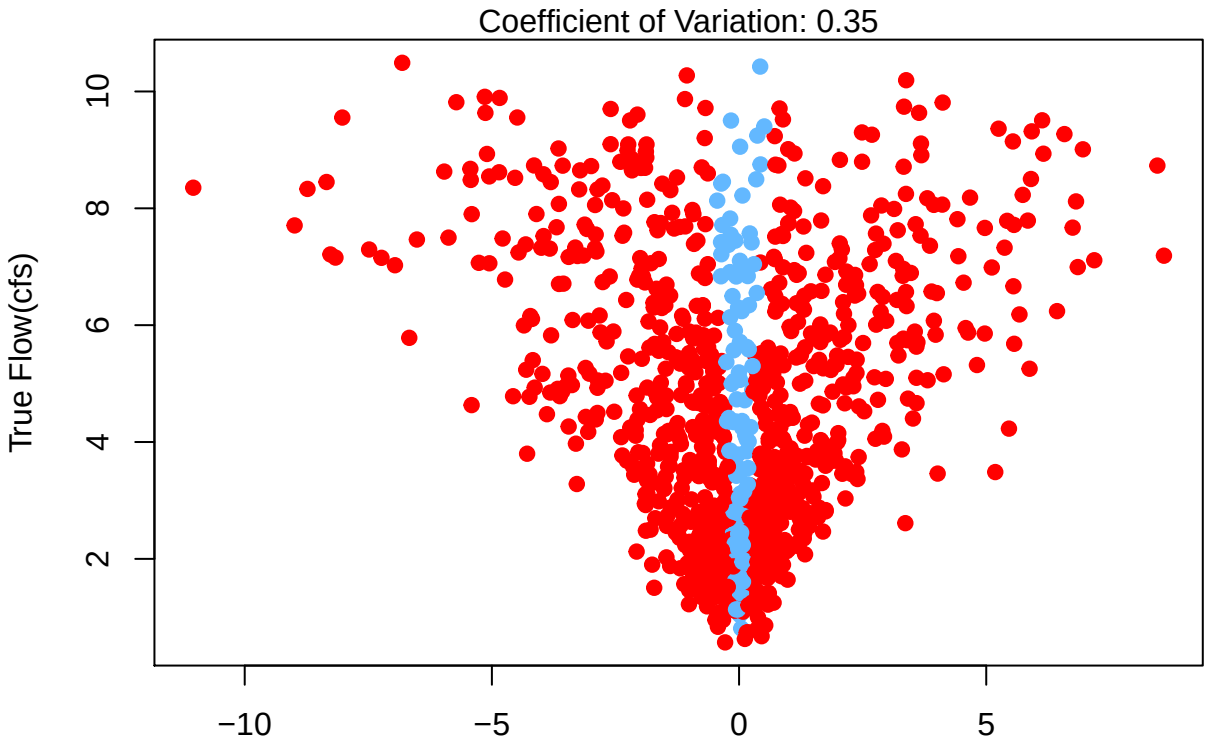

Coefficient of Variation: 0.29
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
Error in Flow (cfs)

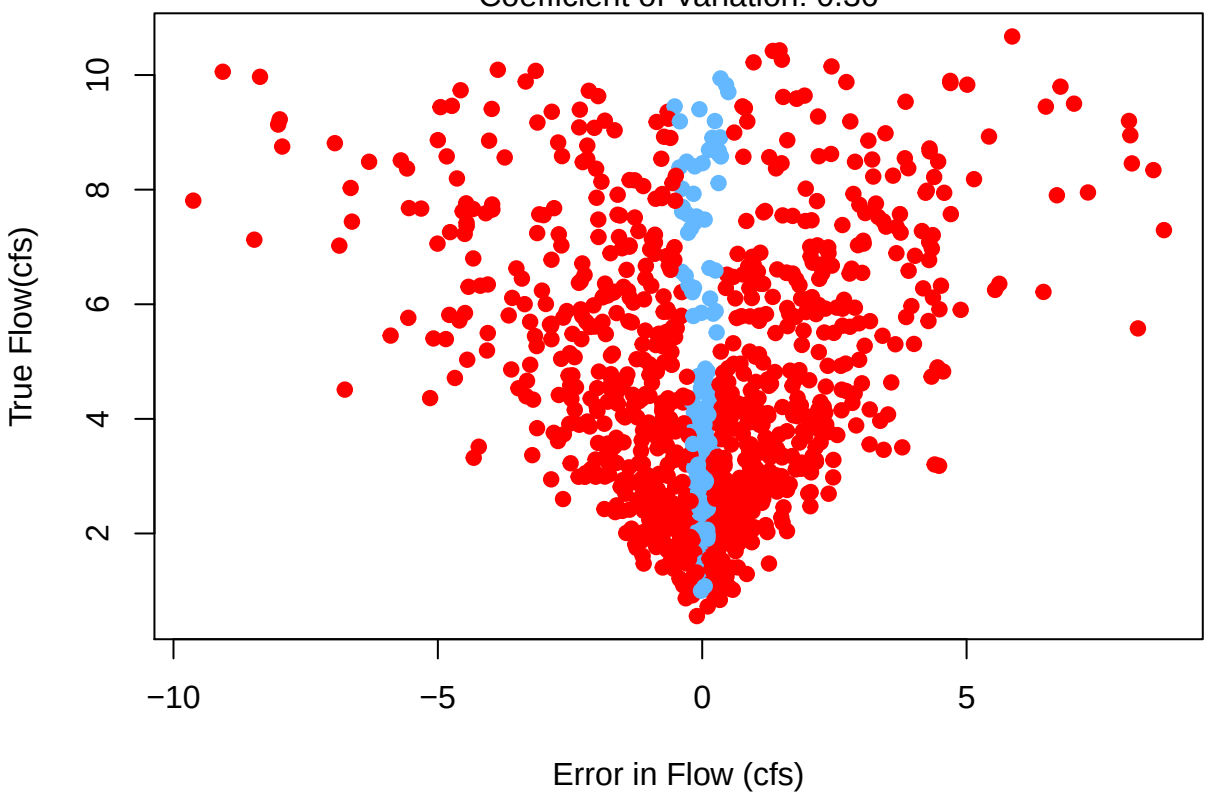

Coefficient of Variation: 0.3
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
10
Error in Flow (cfs)

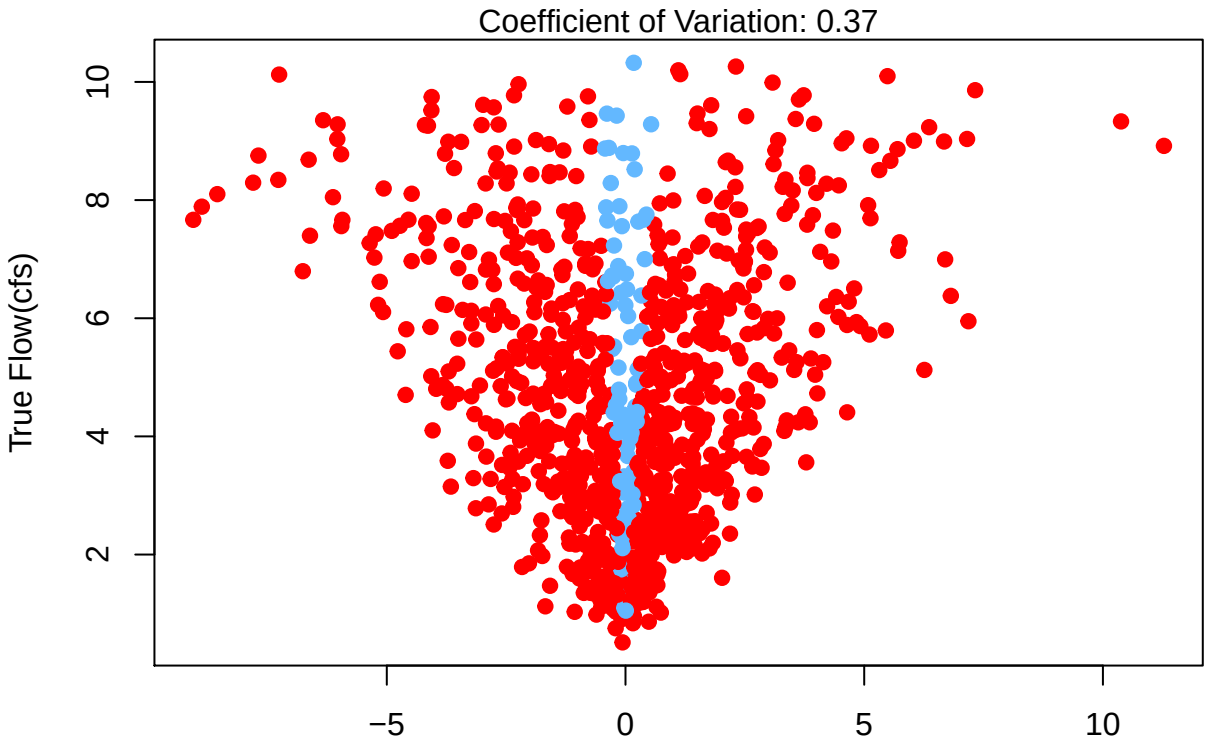

Coefficient of Variation: 0.31
True Flow(cfs)
2
4
6
8
10
−5
0
5
10
Error in Flow (cfs)
Coefficient of Variation: 0.32
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
Error in Flow (cfs)

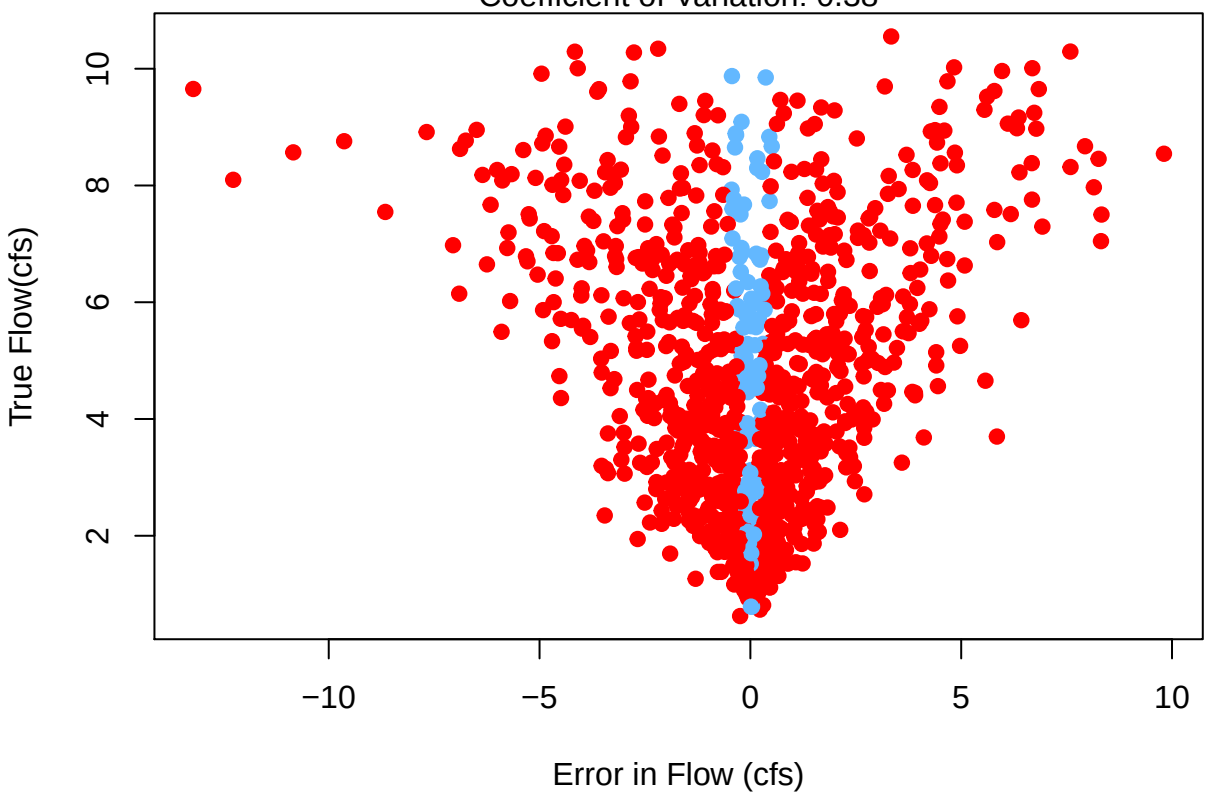

Coefficient of Variation: 0.33
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
10
Error in Flow (cfs)

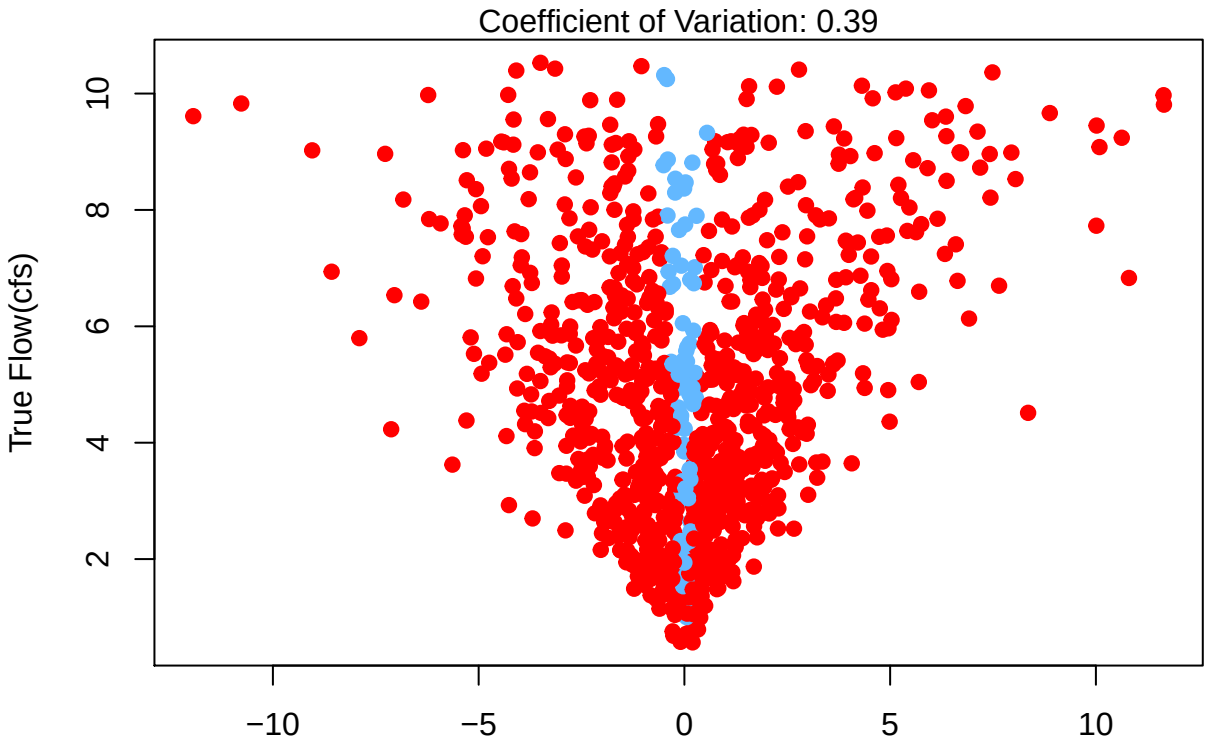

Coefficient of Variation: 0.34
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
10
15
Error in Flow (cfs)

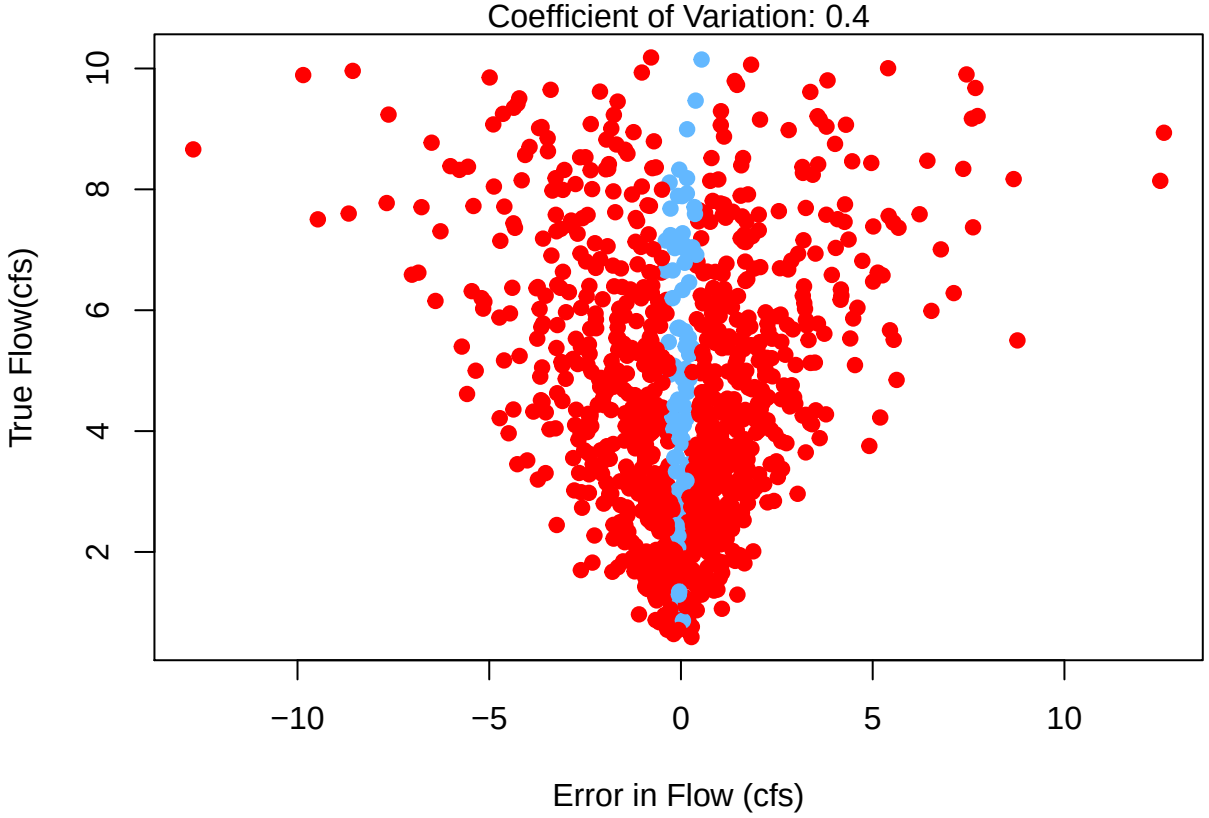

Coefficient of Variation: 0.35
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
Error in Flow (cfs)

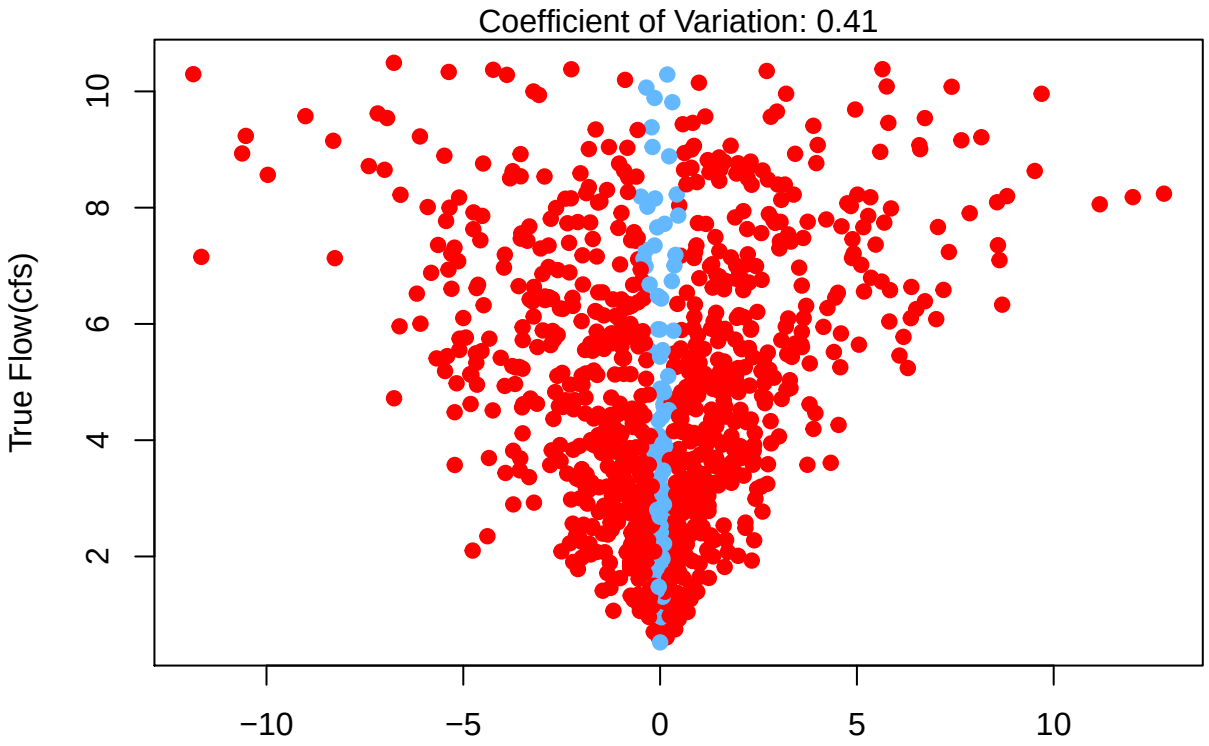

Coefficient of Variation: 0.36
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
Error in Flow (cfs)

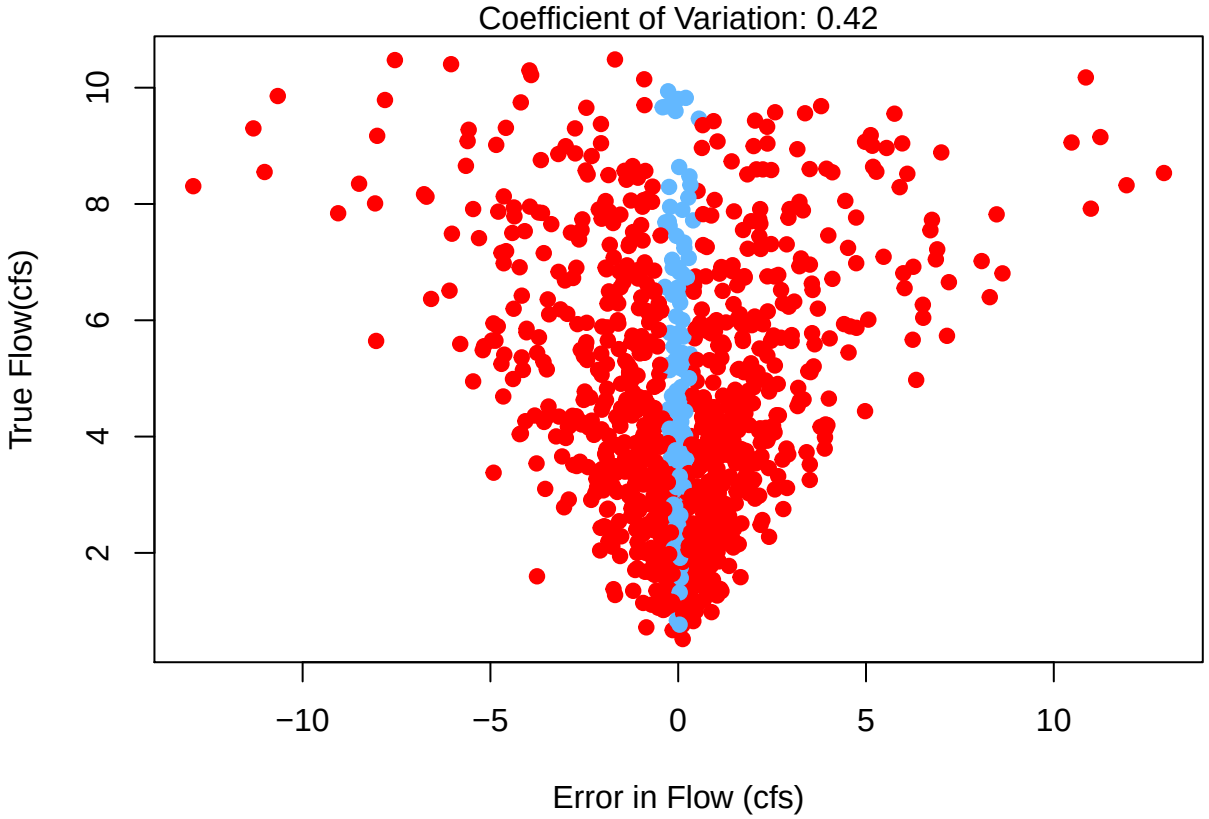

Coefficient of Variation: 0.37
True Flow(cfs)
2
4
6
8
10
−5
0
5
10
Error in Flow (cfs)

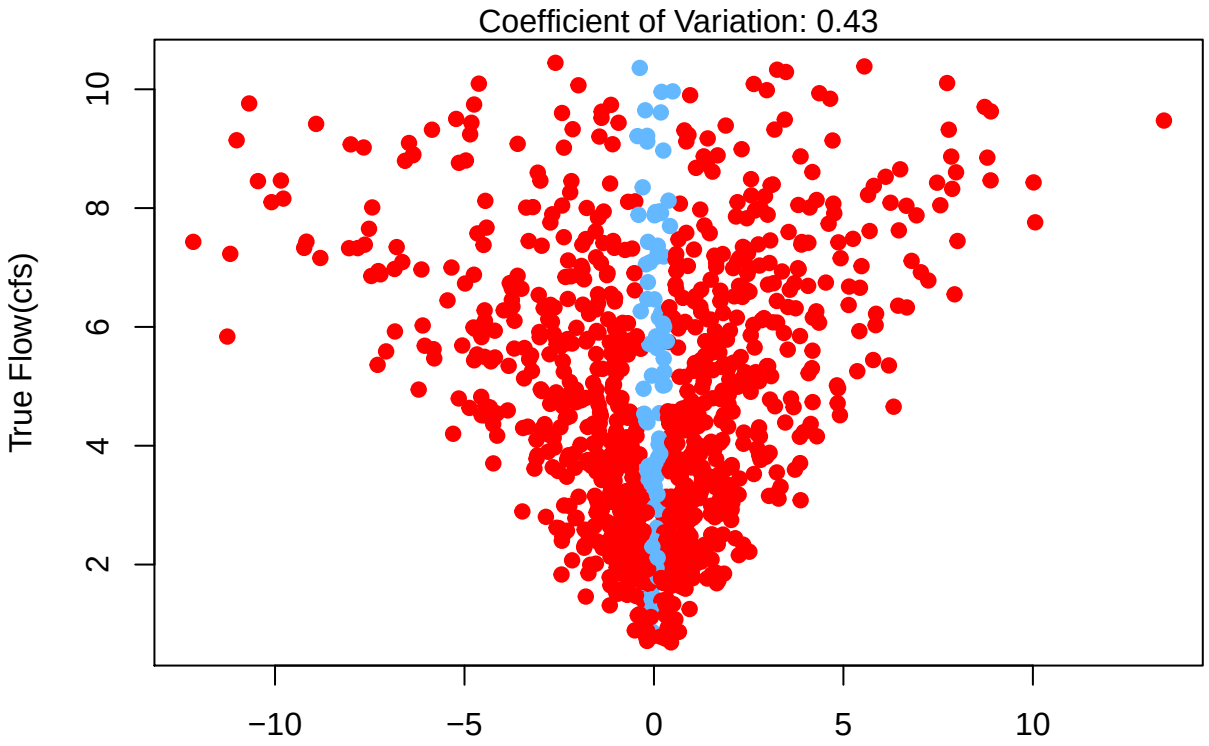

Coefficient of Variation: 0.38
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
10
Error in Flow (cfs)

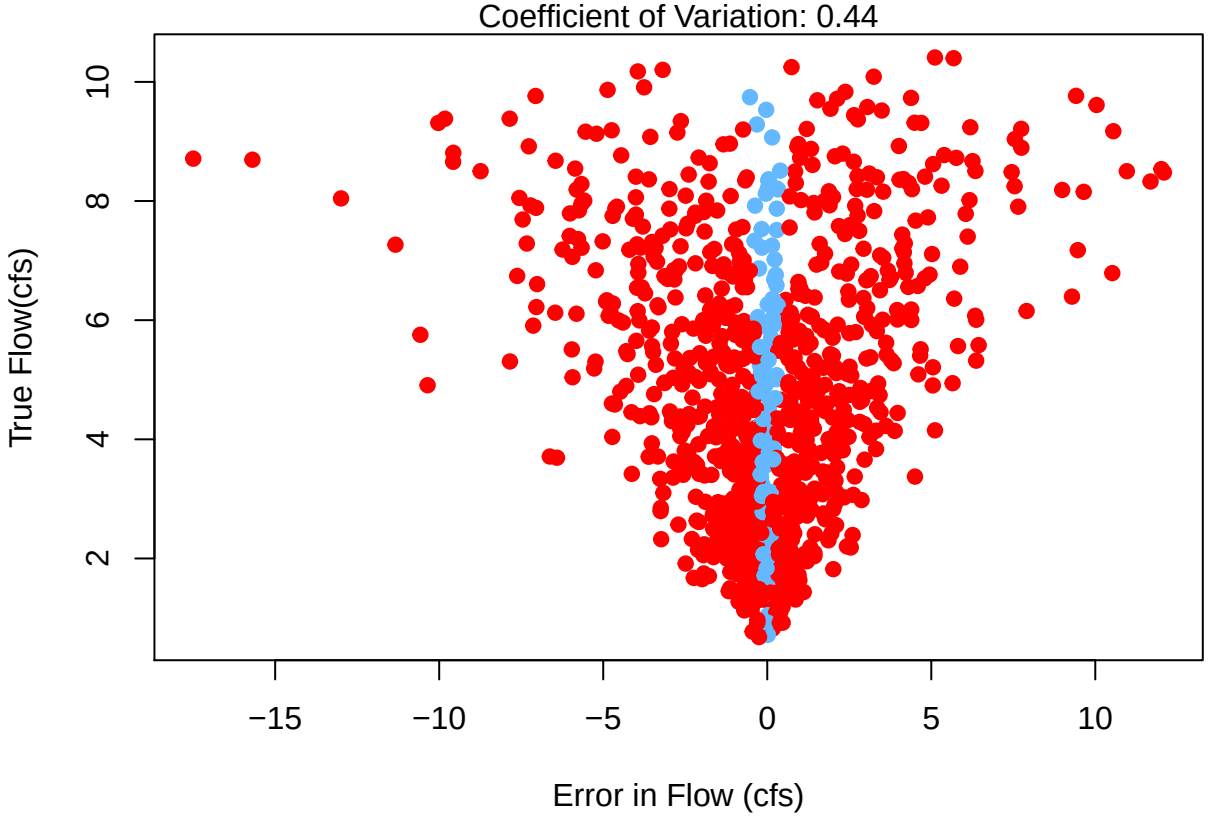

Coefficient of Variation: 0.39
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
10
Error in Flow (cfs)

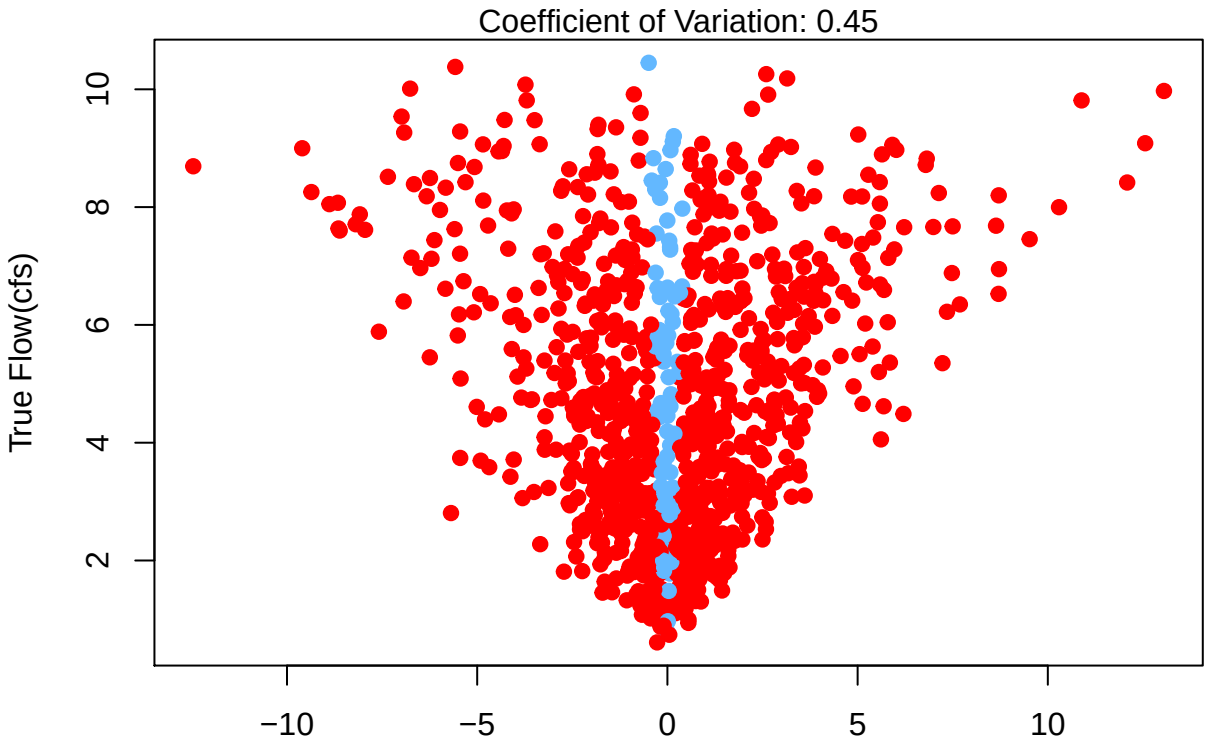

Coefficient of Variation: 0.4
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
10
Error in Flow (cfs)

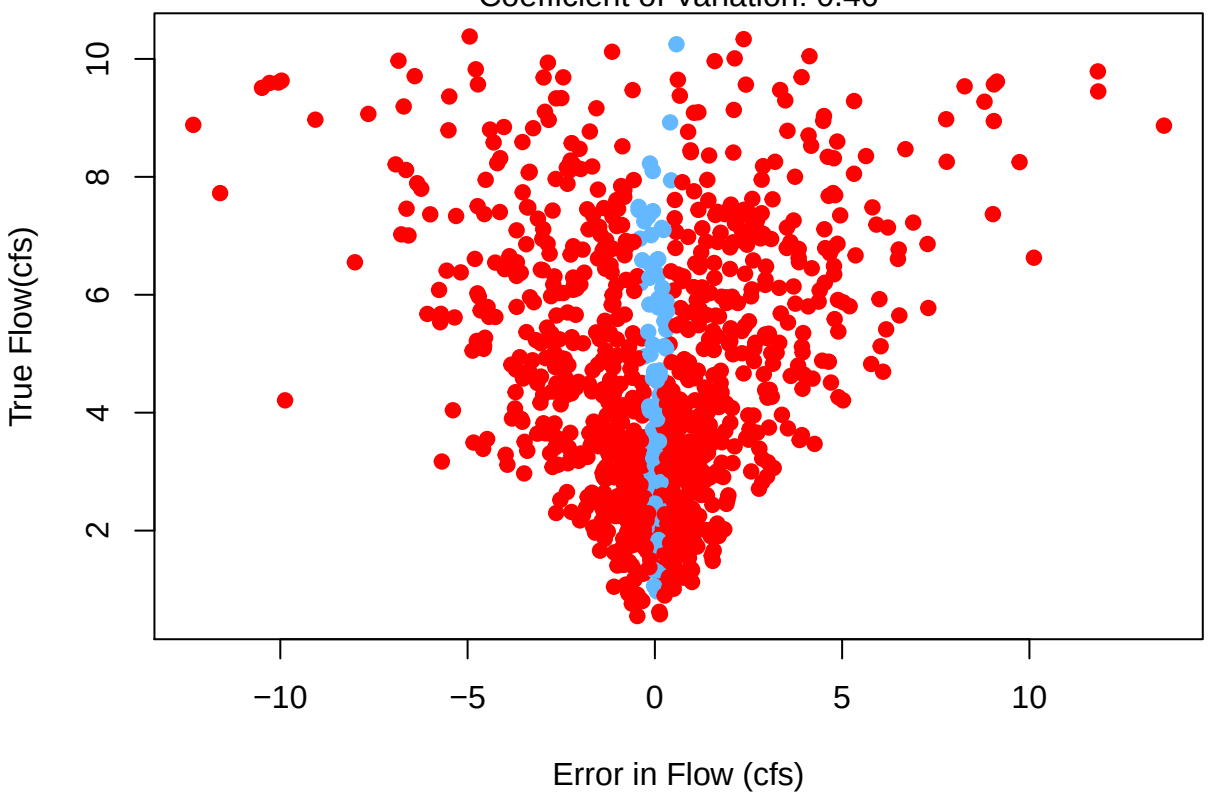

Coefficient of Variation: 0.41
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
10
Error in Flow (cfs)

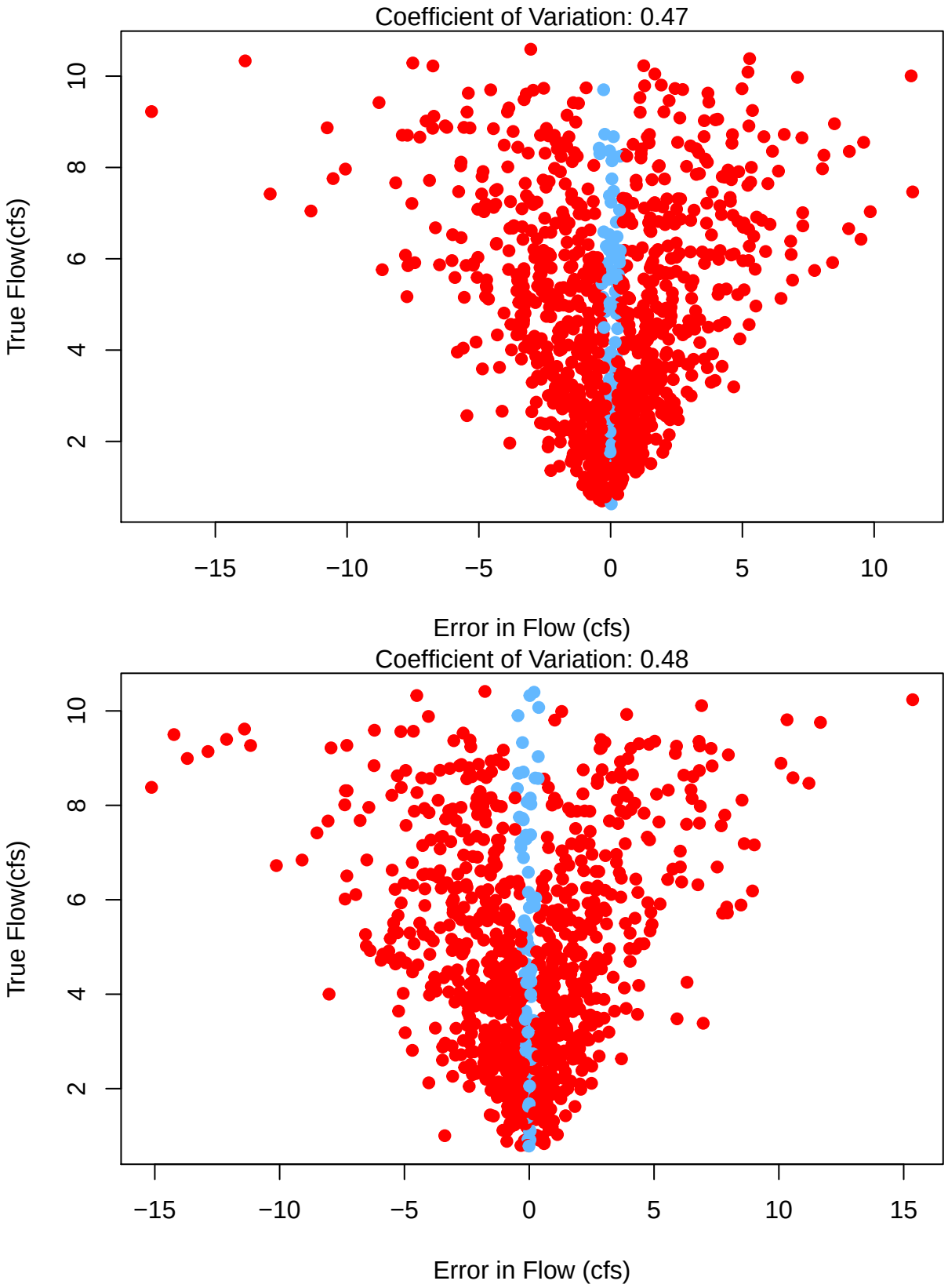

Coefficient of Variation: 0.42
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
10
Error in Flow (cfs)

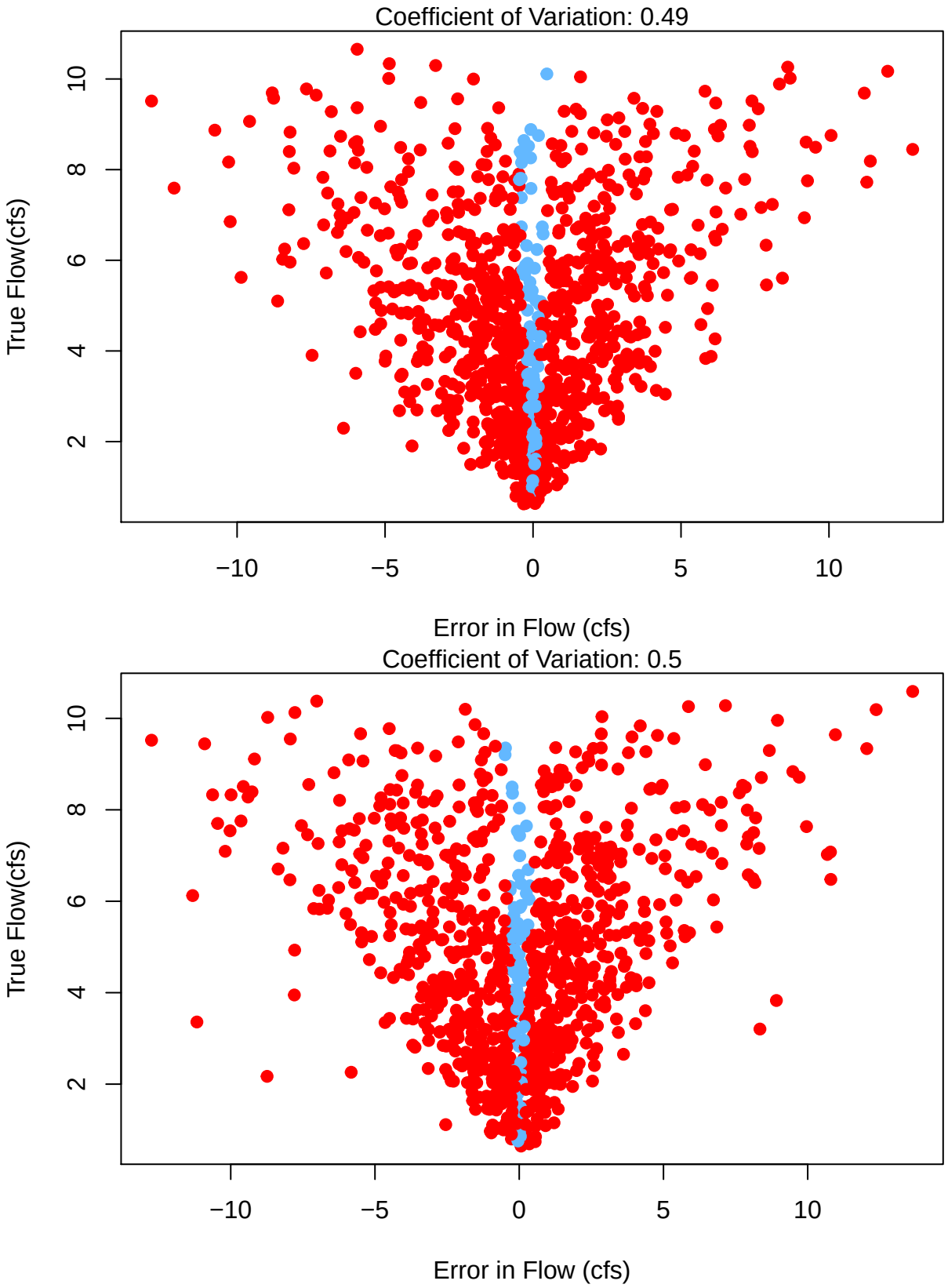

Coefficient of Variation: 0.43
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
10
Error in Flow (cfs)


Coefficient of Variation: 0.44
True Flow(cfs)
2
4
6
8
10
−15
−10
−5
0
5
10
Error in Flow (cfs)


Coefficient of Variation: 0.45
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
10
Error in Flow (cfs)


Coefficient of Variation: 0.46
True Flow(cfs)
2
4
6
8
10
−10
−5
0
5
10
Error in Flow (cfs)


Coefficient of Variation: 0.47
True Flow(cfs)
2
4
6
8
10
−15
−10
−5
0
5
10
Error in Flow (cfs)
Coefficient of Variation: 0.48
True Flow(cfs)
2
4
6
8
10
−15
−10
−5
0
5
10
15
Error in Flow (cfs)

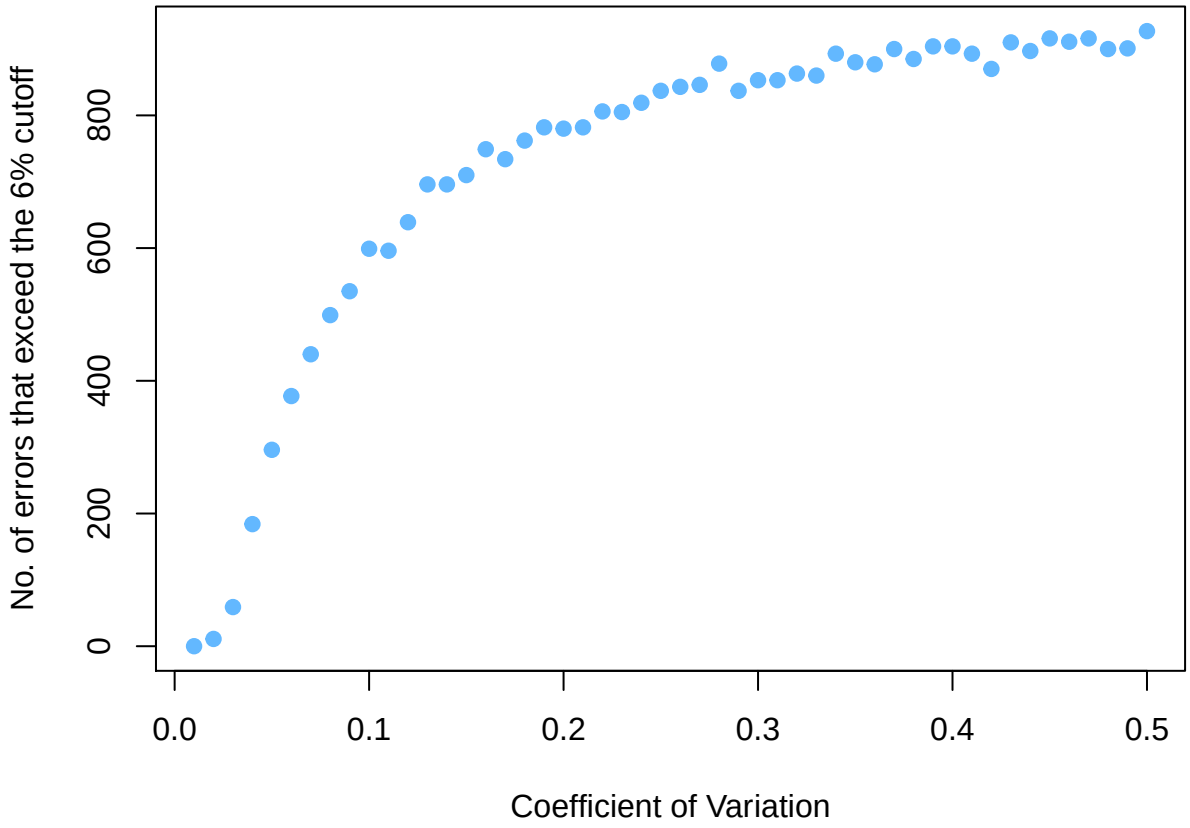



```
png("outputs/optfn_equation_error.png", width = 4.5, height = 4, units = "in", pointsize = 8,
    res = 1200)
```

```
par(mar = c(4, 4.5, 1, 1) + 0.1)
plot(cvrange, nsim - do.call(rbind, lapply(optresult, colSums))[, "ERRORCONDITION"],
    xlab = "Coefficient of Variation", ylab = "No. of errors that exceed the 6% cutoff",
    pch = 19, col = "steelblue1")
dev.off()
```

```
## pdf
##   2
```

```
###### Code for R Markdown Display, Can be Deleted
par(mar = c(4, 4.5, 1, 1) + 0.1)
plot(cvrange, nsim - do.call(rbind, lapply(optresult, colSums))[, "ERRORCONDITION"],
    xlab = "Coefficient of Variation", ylab = "No. of errors that exceed the 6% cutoff",
    pch = 19, col = "steelblue1")
```




```
dev.off()
```

```
## null device
##           1
```

5 Uncertainty From Operations – Simulations

Now that the equation has been defined, we move to the second part of the model - putting the equation into a dynamic system. This just means assigning standard deviations to the variables, G, H, and T. Later in the model there will be many gates in parallel and series, represented by an array, but here we are just looking at the behavior of one gate. The sensitivity analysis of the equation will also be calcualted for a single gate.

5.1 Uncertainty in Operational Flow for a Single Gate

```
# gsd, hsd, tsd are the standard deviations of G, H and t respectively note that
# optflowerrorp is the percentage 6 or 12 % error that is permissable. This error
# is related to the coefofvar allowed in the equation parameters. The largest
# allowed for 100% compliance with the 6% error is 0.02, and for 12% error is
# 0.06.

operationalerrorsim <- function(dataindex, coefofvar = 0.02, optflowerrorp = 0.06,
    gsd = 0.1, hsd = 0.1, tsd = 0.25) {

    # dataindex is probably unnecessary with only one gate
    di <- dataindex

    # we can adj the ability to call different functions here, but for now just
    # equation #1 These coefficients were the same as above as far as I can tell, not
    # sure why they were repeated in the code

    atrue <- modelcoef[[1]]  # assume the mean is the true value
    astdv <- abs(coefofvar * modelcoef[[1]])
    anorm <- rnorm(nsim, modelcoef[[1]], astdv)  # samples around true value, creating error

    btrue <- modelcoef[[2]]
    bstdv <- abs(coefofvar * modelcoef[[2]])
    bnorm <- rnorm(nsim, modelcoef[[2]], bstdv)

    ctrue <- modelcoef[[3]]
    cstdv <- abs(coefofvar * modelcoef[[3]])
    cnorm <- rnorm(nsim, modelcoef[[3]], cstdv)

    dtrue <- modelcoef[[4]]
    dstdv <- abs(coefofvar * modelcoef[[4]])
    dnorm <- rnorm(nsim, modelcoef[[4]], dstdv)

    etrue <- modelcoef[[5]]
    estdv <- abs(coefofvar * modelcoef[[5]])
    enorm <- rnorm(nsim, modelcoef[[5]], estdv)

    aatrue <- modelcoef[[6]]
    aastdv <- abs(coefofvar * modelcoef[[6]])
    aanorm <- rnorm(nsim, modelcoef[[6]], aastdv)

    bbtrue <- modelcoef[[7]]
    bbstdv <- abs(coefofvar * modelcoef[[7]])
    bbnorm <- rnorm(nsim, modelcoef[[7]], bbstdv)

    cctrue <- modelcoef[[8]]
    ccstdv <- abs(coefofvar * modelcoef[[8]])
    ccnorm <- rnorm(nsim, modelcoef[[8]], ccstdv)

    ddtrue <- modelcoef[[9]]
    ddstdv <- abs(coefofvar * modelcoef[[9]])
    ddnorm <- rnorm(nsim, modelcoef[[9]], ddstdv)
```

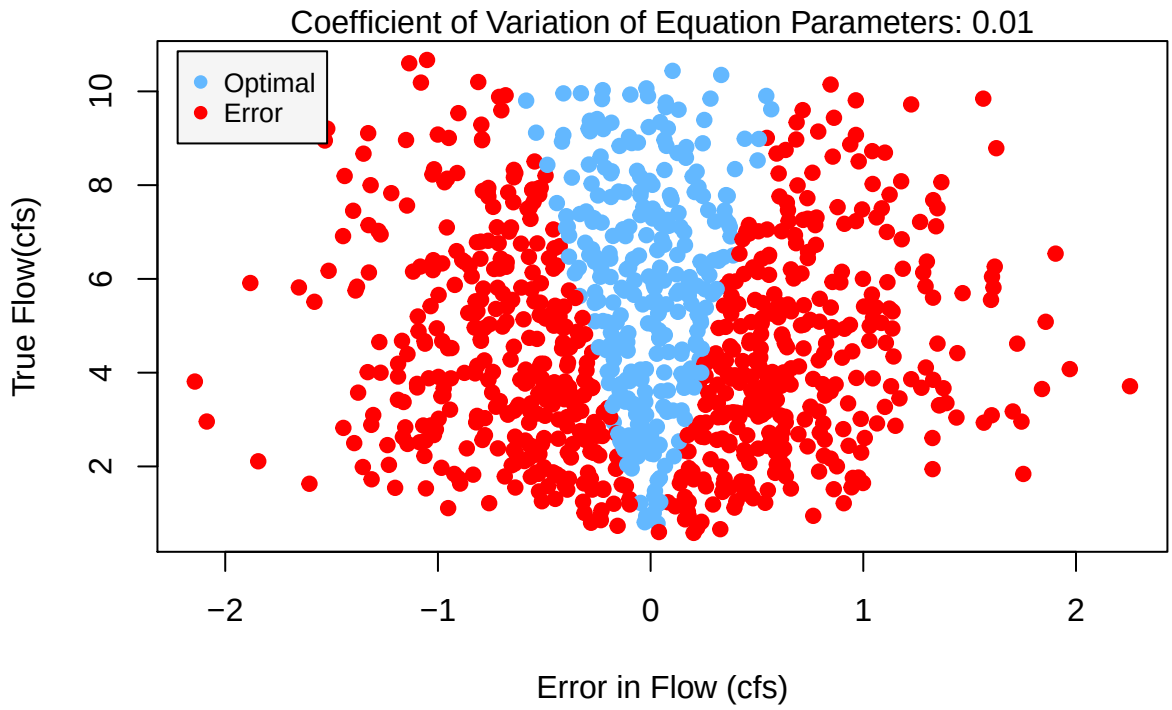
```
    # note that G and H are no longer certain here, let's assume a normal
    # distribution of errors sample a uniform distribution instead of iteratting over
    # all possible values
    guniform <- runif(nsim, min(flowdfl$G), max(flowdfl$G))
    huniform <- runif(nsim, min(flowdfl$H), max(flowdfl$H))
    tuniform <- runif(nsim, 0, 12)  # in hours

    # now assign an error to each sampled G and H, adjed t for volume calcs
    tnorm <- gnorm <- hnorm <- vector()
    for (i in 1:nsim) {
        gnorm[i] <- rnorm(1, guniform[i], gsd)
        hnorm[i] <- rnorm(1, huniform[i], hsd)
        tnorm[i] <- rnorm(1, tuniform[i], tsd)
    }

    volsim <- voltruth <- flowsim <- flowtruth <- vector()

    for (i in 1:nsim) {
        # we can adj the ability to call different functions here, but for now just
        # equation #1
        flowsim[i] <- (anorm[i] + bnorm[i] * gnorm[i] + cnorm[i] * gnorm[i]^2 + dnorm[i] *
            gnorm[i]^3 + enorm[i] * log(hnorm[i]))/(1 + aanorm[i] * gnorm[i] + bbnorm[i] *
            log(hnorm[i]) + ccnorm[i] * (log(hnorm[i]))^2 + ddnorm[i] * (log(hnorm[i]))^3)

        flowtruth[i] <- (atrue + btrue * guniform[i] + ctrue * guniform[i]^2 + dtrue *
            guniform[i]^3 + etrue * log(huniform[i]))/(1 + aatrue * gnorm[i] + bbtrue *
            log(huniform[i]) + cctrue * (log(huniform[i]))^2 + ddtrue * (log(huniform[i]))^3)

        volsim[i] <- flowsim[i] * tnorm[i]

        voltruth[i] <- flowtruth[i] * tuniform[i]
    }

    flowerror <- flowtruth - flowsim
    optflowerror <- optflowerrorp * flowtruth
    errorconditioncol <- ifelse(abs(flowerror) < optflowerror, 1, 0)
    volerror <- voltruth - volsim
    optvolerror <- optflowerrorp * voltruth
    volerrorconditioncol <- ifelse(abs(optvolerror) < optflowerror, 1, 0)

    # conceptual exercise
    resultsdf <- cbind(FLOWTRUTH = flowtruth, FLOWSIM = flowsim, FLOWERROR = flowerror,
        OPTFLOW = optflowerror, ERRORCONDITION = errorconditioncol, VOLTRUTH = voltruth,
        VOLSIM = volsim, VOLERROR = volerror, OPTVOL = optvolerror, VOLERRORCONDITION = volerrorconditi
    return(resultsdf)
}

# Have an option for 6 or 12 % error to atuomatically print both graphs
operationresult <- as.data.frame(operationalerrorsim(dataindex = di, coefofvar = 0.02,
    optflowerrorp = 0.06))

# QUESTION: Some results are NaN because gnorm or hnrom produce negative numbers,
# do we need to fix that?
```

##5.2 Plots I believe the code below prints the observed vs predicted plots for the code above. There is a note in the code below on line 358 about the graph looking weird.

```
png("outputs/operational_error/optimization_cloud.png", width = 4.5, height = 4,
    units = "in", pointsize = 8, res = 1200)
par(mar = c(4, 4, 2, 1) + 0.1)
plot(operationresult[, "FLOWERROR"], operationresult[, "FLOWTRUTH"], col = c("red",
    "steelblue1")[operationresult[, "ERRORCONDITION"] + 1], pch = 19, xlab = "Error in Flow (cfs)",
    ylab = "True Flow(cfs)")
mtext(paste0("Coefficient of Variation of Equation Parameters: 0.01"), side = 3,
    line = 0)
legend("topleft", c("Optimal", "Error"), pch = c(19, 19), col = c("steelblue1", "red"),
    inset = c(0.02, 0.02), cex = 0.8, bg = "grey96")
dev.off()
```

```
## pdf
##   2
```

```
###### Code for R Markdown Display, Can be Deleted
plot(operationresult[, "FLOWERROR"], operationresult[, "FLOWTRUTH"], col = c("red",
    "steelblue1")[operationresult[, "ERRORCONDITION"] + 1], pch = 19, xlab = "Error in Flow (cfs)",
    ylab = "True Flow(cfs)")
mtext(paste0("Coefficient of Variation of Equation Parameters: 0.01"), side = 3,
    line = 0)
legend("topleft", c("Optimal", "Error"), pch = c(19, 19), col = c("steelblue1", "red"),
    inset = c(0.02, 0.02), cex = 0.8, bg = "grey96")
```




```
dev.print()
```

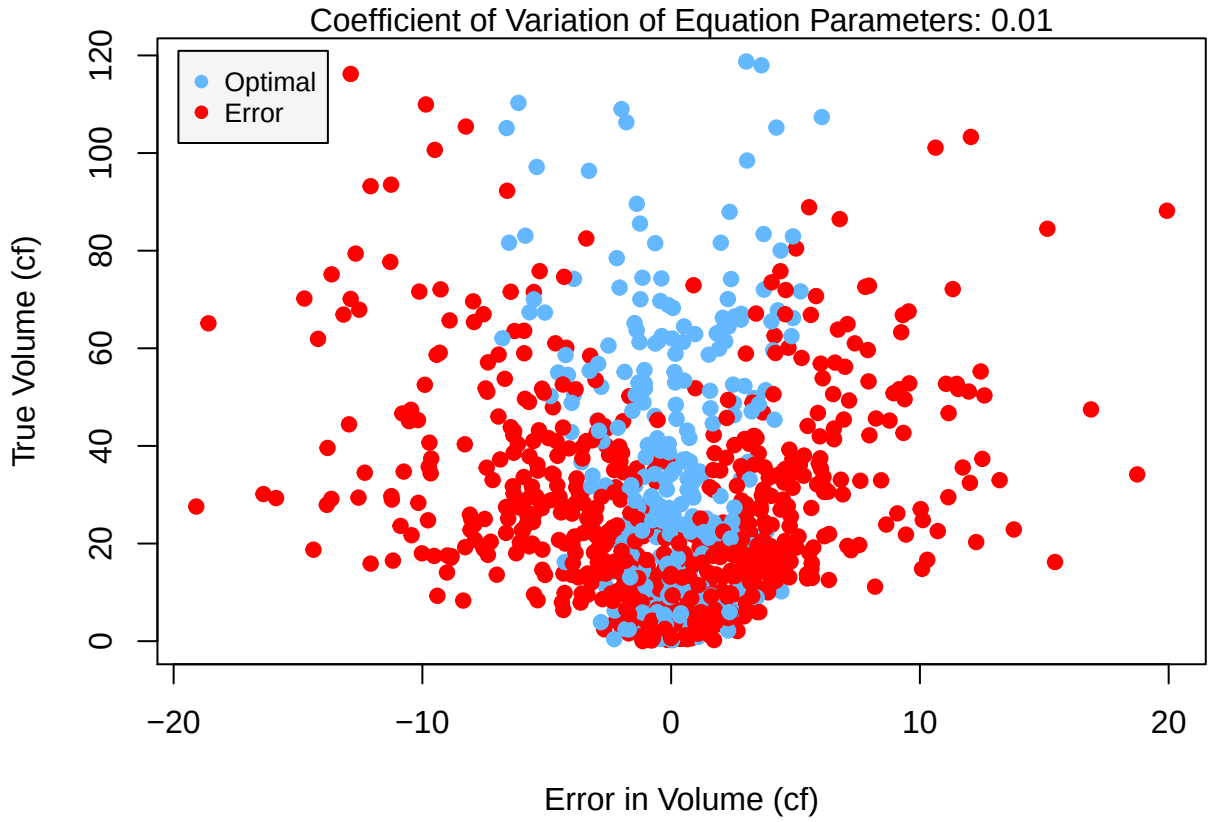
```
## pdf
##   2
```

```
##### This graphs looks weird because it uses the error conditional for flow, but
##### displays volume I tried making an error conditional for volume, but it didn't
```

```
##### work as I inititally expected, and I wasn't sure how much this really mattered
##### so I stopped working on it quickly. If it is a concern we can keep working on
##### it.
png("outputs/operational_volume_error/optimization_cloud.png", width = 4.5, height = 4,
    units = "in", pointsize = 8, res = 1200)
par(mar = c(4, 4, 2, 1) + 0.1)
plot(operationresult[, "VOLERROR"], operationresult[, "VOLTRUTH"], col = c("red",
    "steelblue1")[operationresult[, "ERRORCONDITION"] + 1], pch = 19, xlab = "Error in Volume (cf)",
    ylab = "True Volume (cf)")
mtext(paste0("Coefficient of Variation of Equation Parameters: 0.01"), side = 3,
    line = 0)
legend("topleft", c("Optimal", "Error"), pch = c(19, 19), col = c("steelblue1", "red"),
    inset = c(0.02, 0.02), cex = 0.8, bg = "grey96")
dev.off()
```

```
## pdf
##   2
```

```
###### Code for R Markdown Display, Can be Deleted
par(mar = c(4, 4, 2, 1) + 0.1)
plot(operationresult[, "VOLERROR"], operationresult[, "VOLTRUTH"], col = c("red",
    "steelblue1")[operationresult[, "ERRORCONDITION"] + 1], pch = 19, xlab = "Error in Volume (cf)",
    ylab = "True Volume (cf)")
mtext(paste0("Coefficient of Variation of Equation Parameters: 0.01"), side = 3,
    line = 0)
legend("topleft", c("Optimal", "Error"), pch = c(19, 19), col = c("steelblue1", "red"),
    inset = c(0.02, 0.02), cex = 0.8, bg = "grey96")
```



```
dev.print()
```

```
## pdf
##   2
```

```
#####
```

5.3 Sensitivity Analysis – uncertaintiy with varying parameters one at a time

The goal of the sensitivity analysis is to look at how the variance of the output changes by changing the variance of the inputs. The inputs are being changed one at a time, which is a super simplistic way to do a sensitivity analysis. I am open to suggestion, but also my PI is fine with this method (he's not super knowledgeable about statistics). This is a new portion of the code that was recently added.

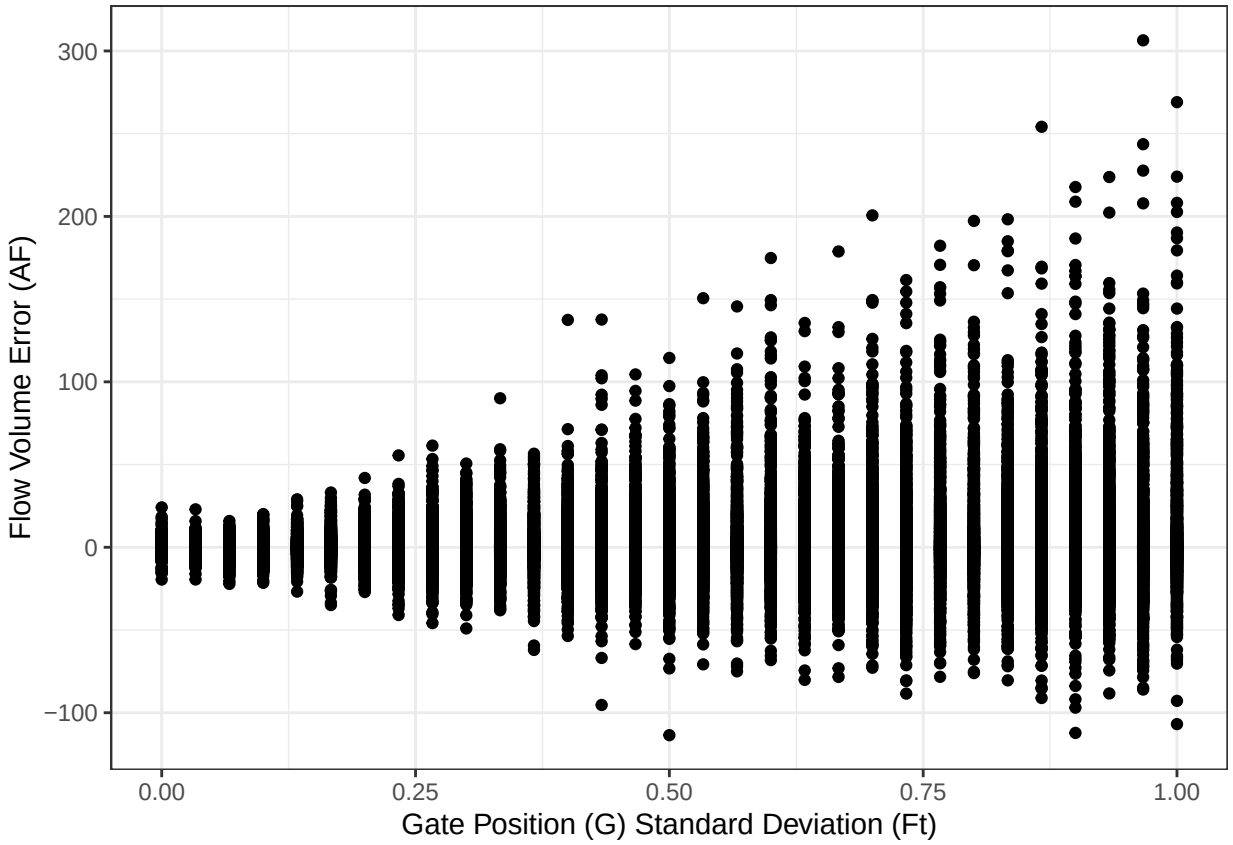
```
library(ggplot2)
SensitivityError_sd <- function(dataindex, coefofvar, optflowerrorp=0.06, gsd, hsd, tsd){

  # dataindex is probably unnecessary with only one gate
  di <- dataindex

  # we can adj the ability to call different functions here, but for now just equation #1
  #These coefficients were the same as above as far as I can tell, not sure why they were repeated in t

  atrue <- modelcoef[[1]] # assume the mean is the true value
  astdv <- abs(coefofvar*modelcoef[[1]])
  anorm <- rnorm(nsim, modelcoef[[1]], astdv) # samples around true value, creating error

  btrue <- modelcoef[[2]]
  bstdv <- abs(coefofvar*modelcoef[[2]])
  bnorm <- rnorm(nsim, modelcoef[[2]], bstdv)

  ctrue <- modelcoef[[3]]
  cstdv <- abs(coefofvar*modelcoef[[3]])
  cnorm <- rnorm(nsim, modelcoef[[3]], cstdv)

  dtrue <- modelcoef[[4]]
  dstdv <- abs(coefofvar*modelcoef[[4]])
  dnorm <- rnorm(nsim, modelcoef[[4]], dstdv)

  etrue <- modelcoef[[5]]
  estdv <- abs(coefofvar*modelcoef[[5]])
  enorm <- rnorm(nsim, modelcoef[[5]], estdv)

  aatrue <- modelcoef[[6]]
  aastdv <- abs(coefofvar*modelcoef[[6]])
  aanorm <- rnorm(nsim, modelcoef[[6]], aastdv)

  bbtrue <- modelcoef[[7]]
  bbstdv <- abs(coefofvar*modelcoef[[7]])
  bbnorm <- rnorm(nsim, modelcoef[[7]], bbstdv)

  cctrue <- modelcoef[[8]]
  ccstdv <- abs(coefofvar*modelcoef[[8]])
  ccnorm <- rnorm(nsim, modelcoef[[8]], ccstdv)
```

```
  ddtrue <- modelcoef[[9]]
  ddstdv <- abs(coefofvar*modelcoef[[9]])
  ddnorm <- rnorm(nsim, modelcoef[[9]], ddstdv)

  # note that G and H are no longer certain here, let's assume a normal distribution of errors
  # sample a uniform distribution instead of iteratting over all possible values
  guniform <- runif(nsim, min(flowdfl$G), max(flowdfl$G))
  huniform <- runif(nsim, min(flowdfl$H), max(flowdfl$H))
  tuniform <- runif(nsim, 0, 12) # in hours

  # now assign an error to each sampled G and H, adjed t for volume calcs
  tnorm <- gnorm <- hnorm <- vector()
  for(i in 1:nsim){
    gnorm[i] <- rnorm(1, guniform[i], gsd)
    hnorm[i] <- rnorm(1, huniform[i], hsd)
    tnorm[i] <- rnorm(1, tuniform[i], tsd)
  }

  volsim <- voltruth <-  flowsim <- flowtruth <- vector()

  for(i in 1:nsim){
    # we can adj the ability to call different functions here, but for now just equation #1
    flowsim[i] <- ( anorm[i] + bnorm[i] * gnorm[i] + cnorm[i] * gnorm[i] ^ 2 + dnorm[i] * gnorm[i] ^ 3 +

    flowtruth[i] <- ( atrue + btrue * guniform[i] + ctrue * guniform[i] ^ 2 + dtrue * guniform[i] ^ 3 +

    volsim[i] <- flowsim[i]*tnorm[i]

    voltruth[i] <- flowtruth[i]*tuniform[i]
  }

  flowerror <- flowtruth-flowsim
  optflowerror <- optflowerrorp*flowtruth
  errorconditioncol <- ifelse(abs(flowerror)<optflowerror, 1, 0)
  volerror <- voltruth-volsim
  optvolerror <- optflowerrorp * voltruth
  volerrorconditioncol <- ifelse(abs(optvolerror)<optflowerror, 1, 0)

  # conceptual exercise
  resultsdf <- cbind(FLOWTRUTH=flowtruth, FLOWSIM=flowsim, FLOWERROR=flowerror, OPTFLOW=optflowerror, E
  return(resultsdf)
}

#Plotting
plotList_Gsd<-list() #initialize list to store the data
Glist<-seq(0,1,1/30) #set up a sequence of values to loop over
for(i in 1:length(Glist)){
  cur_df<- as.data.frame(SensitivityError_sd(gsd=Glist[i],dataindex=di, coefofvar=0.02, optflowerrorp=0
  plotList_Gsd[[i]]<-data.frame(VOLERROR=cur_df$VOLERROR,Gsd=Glist[i]) #Save only the volume error
#print(i)#Counter to see if it runs correctly
}

G_Plot_Data<-do.call(rbind,plotList_Gsd) #change the list to a dataframe for plotting
```

```
g_plot<-ggplot()+geom_point(data=G_Plot_Data,aes(x=Gsd,y=VOLERROR))+ labs(
              y="Flow Volume Error (AF)",
              x="Gate Position (G) Standard Deviation (Ft)")+ #create the plot
              theme_bw() #remove gray background

ggsave(filename ="gPlot.png", #Save the file with the some options
       plot=g_plot, #R object to be saved
       path="/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/outputs/sensitivity_analysis_pm/", #P
      device="png", #type of saved file
      width=6.5,
      height=5,
      units="in") #units in inches

  ######Code for R Markdown Display, Can be Deleted
  print("sensitivity_analysis_pm_G_plot")
```

```
## [1] "sensitivity_analysis_pm_G_plot"
```

```
  print(g_plot)
```

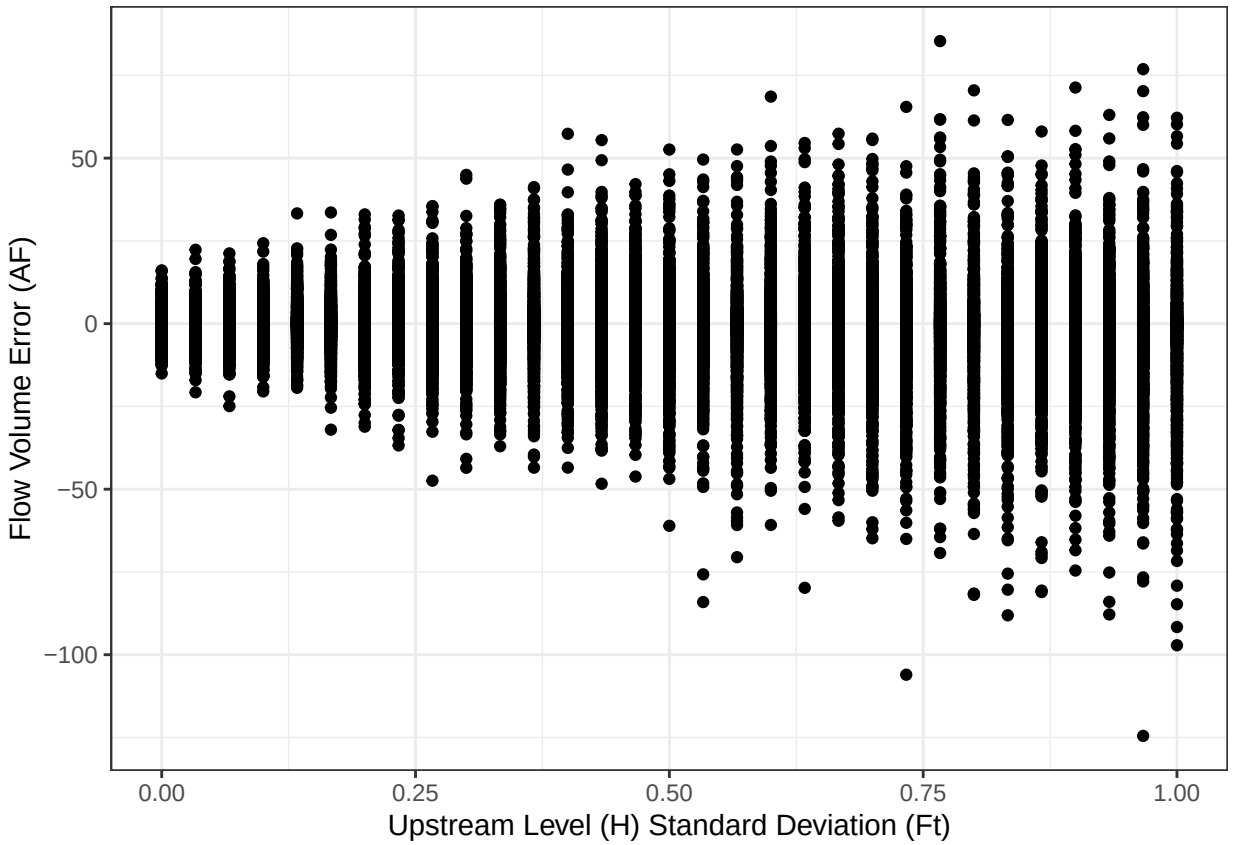



```
  #####


#H Sd
plotList_Hsd<-list() #Repeat aboue code, but with different variables
Hlist<-seq(0,1,1/30)
for(i in 1:length(Hlist)){
```

```
  cur_df<- as.data.frame(SensitivityError_sd(hsd=Hlist[i],dataindex=di, coefofvar=0.02, optflowerrorp=0
  plotList_Hsd[[i]]<-data.frame(VOLERROR=cur_df$VOLERROR,Hsd=Hlist[i])
#print(i)
}

H_Plot_Data<-do.call(rbind,plotList_Hsd)

H_plot<-ggplot()+geom_point(data=H_Plot_Data,aes(x=Hsd,y=VOLERROR))+ labs(
              y="Flow Volume Error (AF)",
              x="Upstream Level (H) Standard Deviation (Ft)")+
              theme_bw()

ggsave(filename ="hPlot.png",
       plot=H_plot,
       path="/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/outputs/sensitivity_analysis_pm/",
      device="png",
      width=6.5,
      height=5,
      units="in")

  ######Code for R Markdown Display, Can be Deleted
  print("sensitivity_analysis_pm_H_plot")
```

```
## [1] "sensitivity_analysis_pm_H_plot"
```

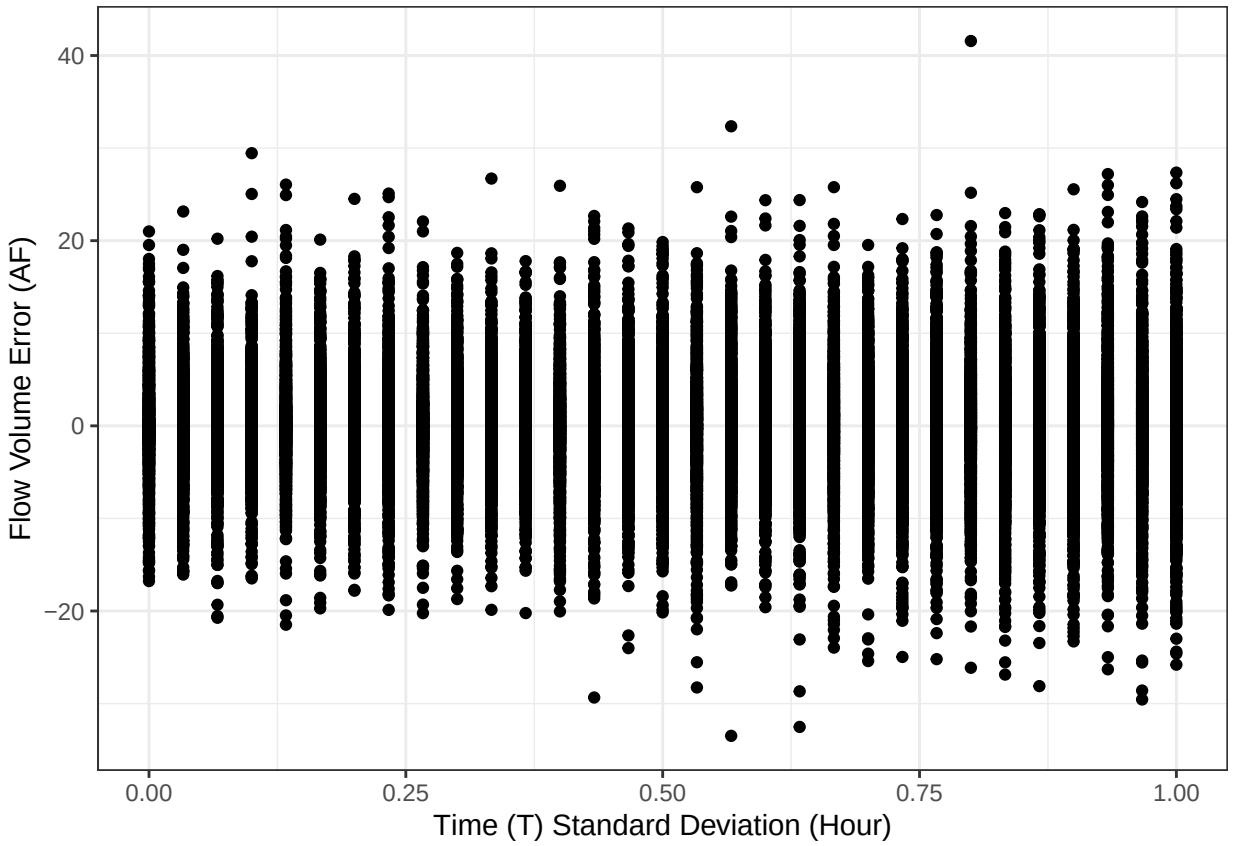
```
  print(H_plot)
```



```
  #####

#T Sd
plotList_Tsd<-list()
Tlist<-seq(0,1,1/30)
for(i in 1:length(Tlist)){
  cur_df<- as.data.frame(SensitivityError_sd(tsd=Tlist[i],dataindex=di, coefofvar=0.02, optflowerrorp=0
  plotList_Tsd[[i]]<-data.frame(VOLERROR=cur_df$VOLERROR,Tsd=Tlist[i])
#print(i)
}

T_Plot_Data<-do.call(rbind,plotList_Tsd)

T_plot<-ggplot()+geom_point(data=T_Plot_Data,aes(x=Tsd,y=VOLERROR))+ labs(
              y="Flow Volume Error (AF)",
              x="Time (T) Standard Deviation (Hour)")+
              theme_bw()

ggsave(filename ="tPlot.png",
       plot=T_plot,
       path="/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/outputs/sensitivity_analysis_pm/",
      device="png",
      width=6.5,
      height=5,
      units="in")

  ######Code for R Markdoum Display, Can be Deleted
  print("sensitivity_analysis_pm_T_plot")
```

```
## [1] "sensitivity_analysis_pm_T_plot"
```

```
  print(T_plot)
```

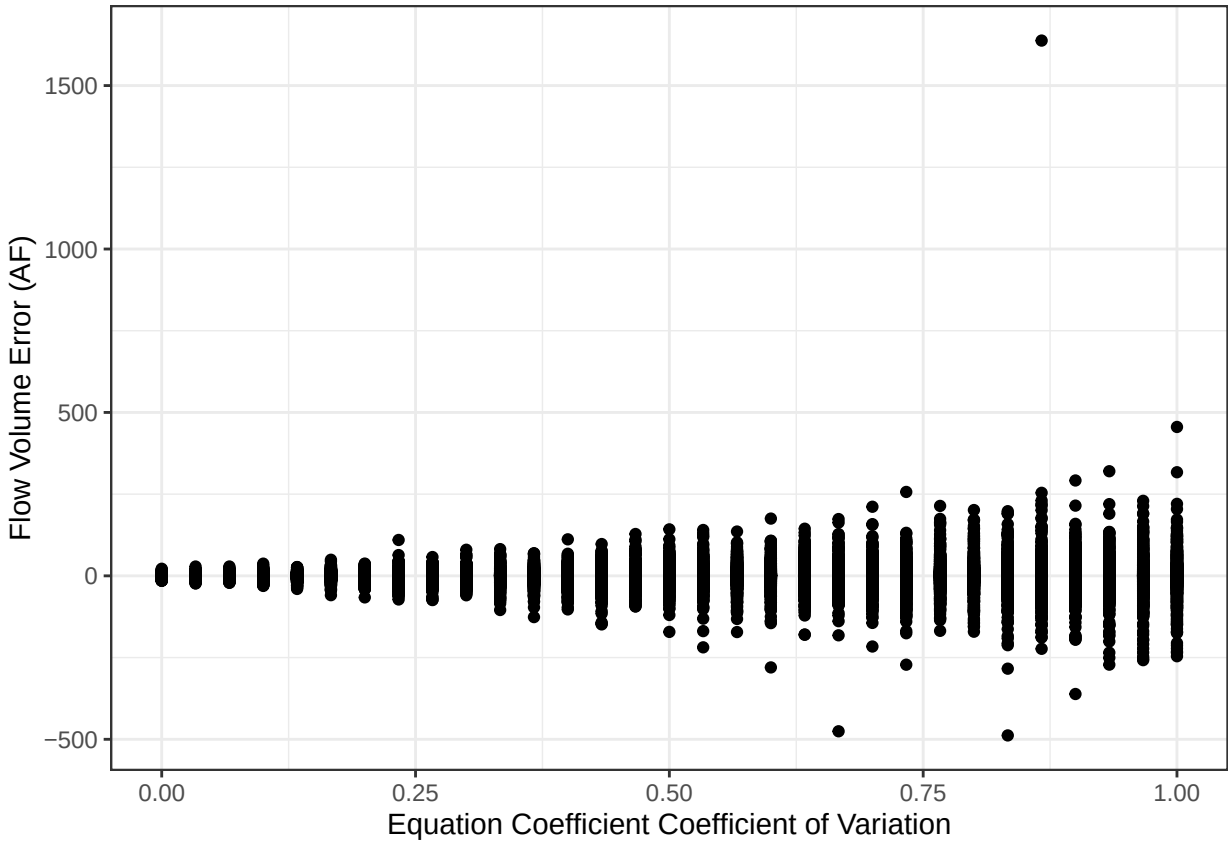



```
  #####

#CV
plotList_cv<-list()
cvlist<-seq(0,1,1/30)
for(i in 1:length(cvlist)){
  cur_df<- as.data.frame(SensitivityError_sd(tsd=0.1,dataindex=di, coefofvar=cvlist[i], optflowerrorp=0
  plotList_cv[[i]]<-data.frame(VOLERROR=cur_df$VOLERROR,cv=cvlist[i])
#print(i)
}

cv_Plot_Data<-do.call(rbind,plotList_cv)

cv_plot<-ggplot()+geom_point(data=cv_Plot_Data,aes(x=cv,y=VOLERROR))+ labs(
              y="Flow Volume Error (AF)",
              x="Equation Coefficient Coefficient of Variation")+
              theme_bw()

ggsave(filename ="cvPlot.png",
       plot=cv_plot,
       path="/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/outputs/sensitivity_analysis_pm/",
      device="png",
      width=6.5,
      height=5,
      units="in")

  ######Code for R Markdown Display, Can be Deleted
```

```
  print("sensitivity_analysis_pm_T_plot")
```

```
## [1] "sensitivity_analysis_pm_T_plot"
```

```
  print(cv_plot)
```




```
  #####

#Line Plots
#GSD Line Plot
G_Plot_DataAbs<-do.call(rbind,plotList_Gsd)
G_Plot_DataAbs$VOLERROR<-abs(G_Plot_DataAbs$VOLERROR)
GplotData_line<-G_Plot_DataAbs%>%group_by(Gsd)%>%summarise(meanErr=mean(VOLERROR, na.rm=T),
                                                           Max=(meanErr+sd(VOLERROR,na.rm=T)),
                                                           Min=(meanErr-sd(VOLERROR,na.rm=T)))
```

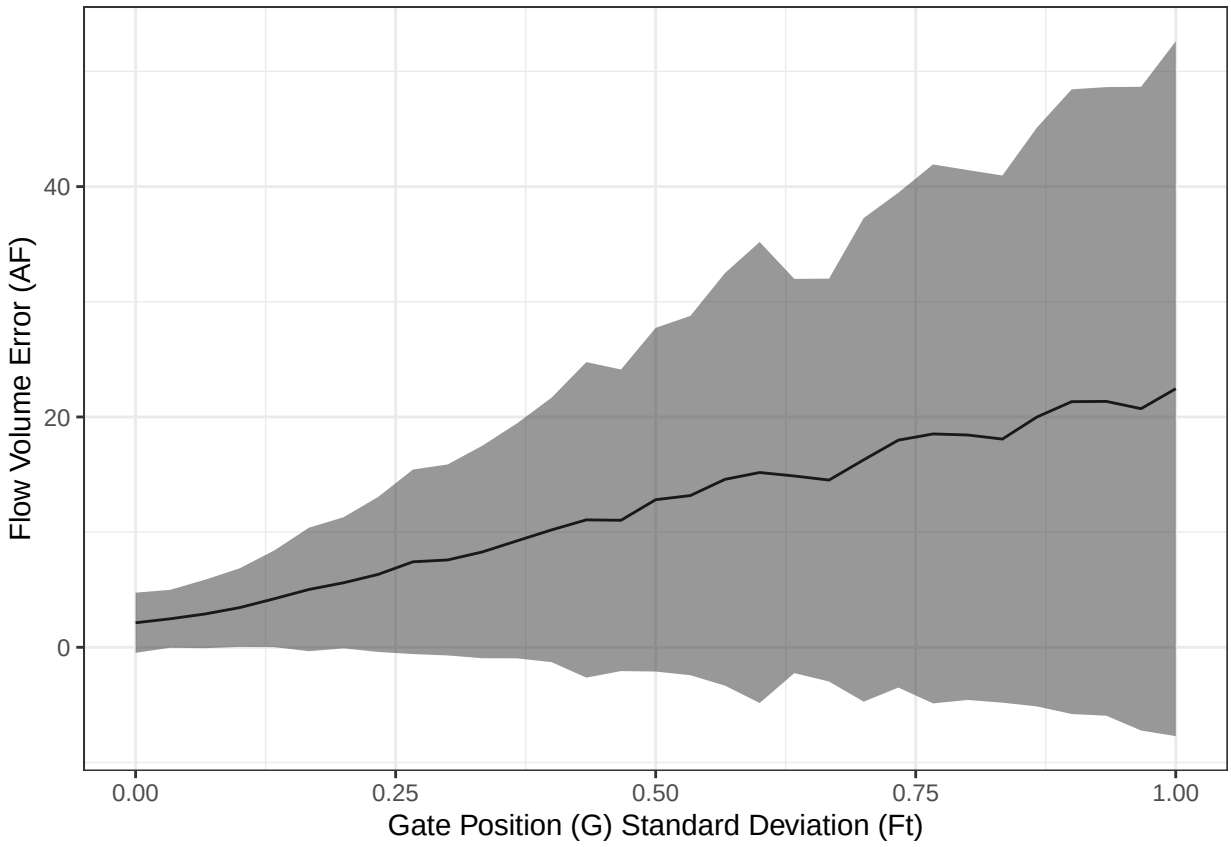
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
g_plot_line<-ggplot()+geom_line(data=GplotData_line,aes(x=Gsd,y=meanErr))+
              geom_ribbon(data=GplotData_line,aes(x=Gsd,ymax=Max,ymin=Min),alpha=.5)+
              labs(
              y="Flow Volume Error (AF)",
              x="Gate Position (G) Standard Deviation (Ft)")+
              theme_bw()

ggsave(filename ="gPlot_line.png",
       plot=g_plot_line,
       path="/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/outputs/sensitivity_analysis_pm/",
```

```
      device="png",
      width=6.5,
      height=5,
      units="in")

  ######Code for R Markdoum Display, Can be Deleted
  print("sensitivity_analysis_pm_gPlot_line")
```

```
## [1] "sensitivity_analysis_pm_gPlot_line"
```

```
  print(g_plot_line)
```




```
  #####


#HSD Line Plot
H_Plot_DataAbs<-do.call(rbind,plotList_Hsd)
H_Plot_DataAbs$VOLERROR<-abs(H_Plot_DataAbs$VOLERROR)
HplotData_line<-H_Plot_DataAbs%>%group_by(Hsd)%>%summarise(meanErr=mean(VOLERROR, na.rm=T),
                                                           Max=(meanErr+sd(VOLERROR,na.rm=T)),
                                                           Min=(meanErr-sd(VOLERROR,na.rm=T)))
```

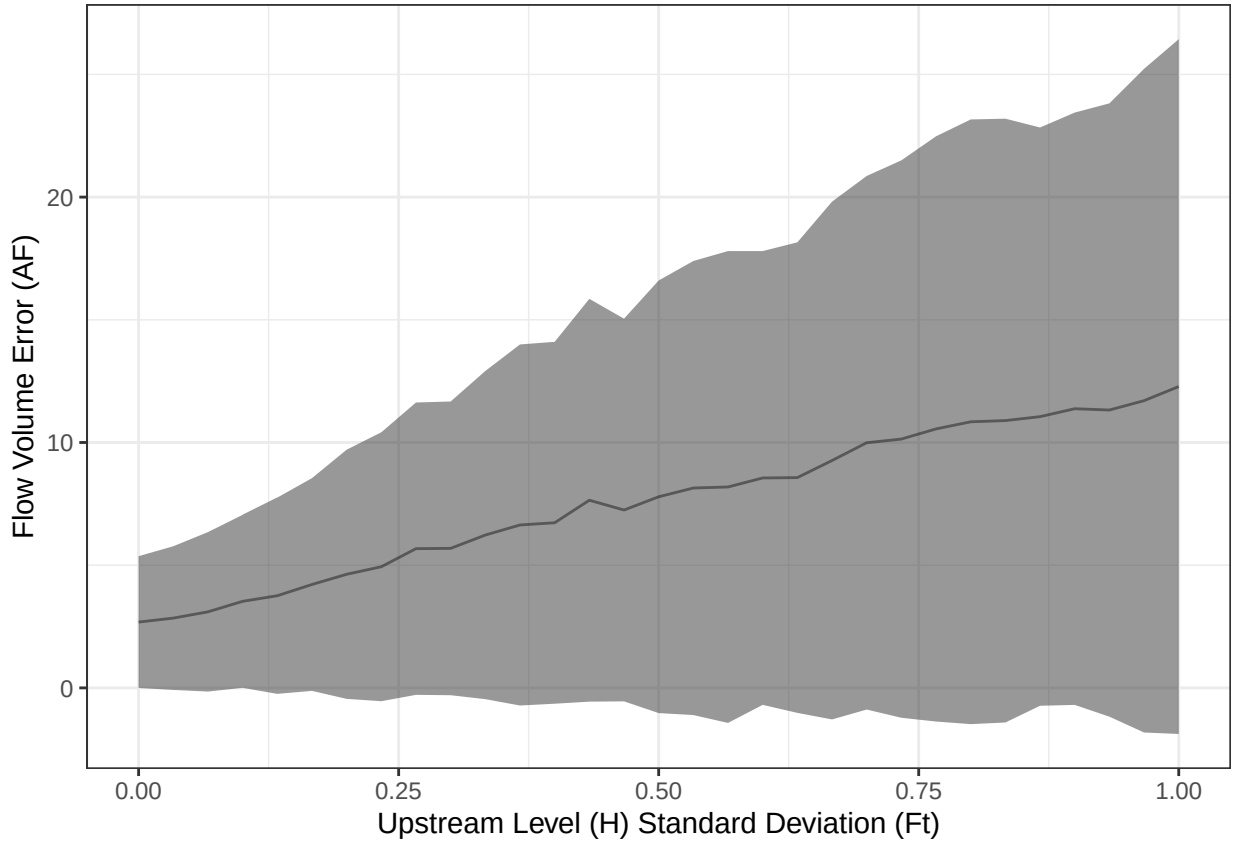
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
H_plot_line<-ggplot()+geom_line(data=HplotData_line,aes(x=Hsd,y=meanErr),alpha=.5)+
              geom_ribbon(data=HplotData_line,aes(x=Hsd,ymax=Max,ymin=Min),alpha=.5)+
              labs(
              y="Flow Volume Error (AF)",
```

```
              x="Upstream Level (H) Standard Deviation (Ft)")+
              theme_bw()

ggsave(filename ="hPlot_line.png",
       plot=H_plot_line,
       path="/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/outputs/sensitivity_analysis_pm/",
      device="png",
      width=6.5,
      height=5,
      units="in")

  ######Code for R Markdoum Display, Can be Deleted
  print("sensitivity_analysis_pm_HPlot_line")
```

```
## [1] "sensitivity_analysis_pm_HPlot_line"
```

```
  print(H_plot_line)
```



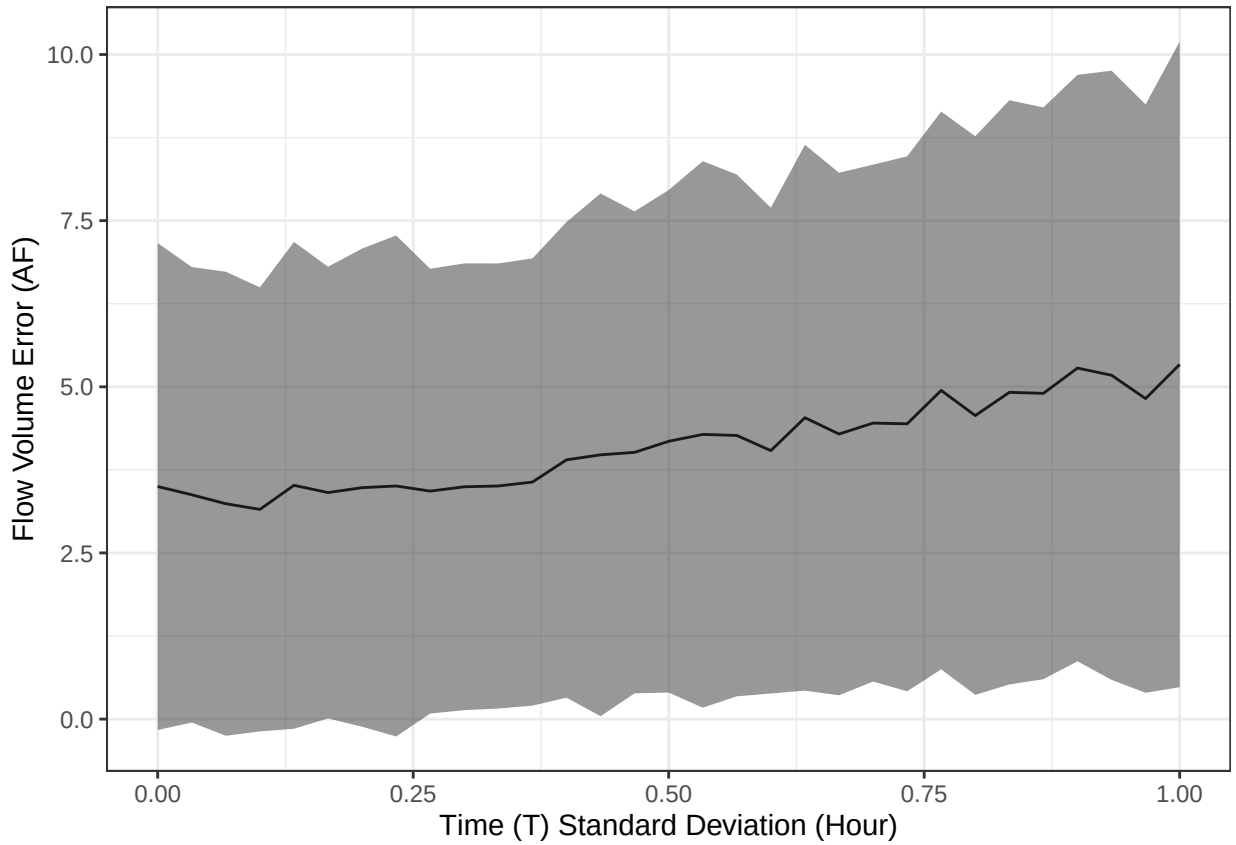

```
  #####

#TSd line plot
T_Plot_DataAbs<-do.call(rbind,plotList_Tsd)
T_Plot_DataAbs$VOLERROR<-abs(T_Plot_DataAbs$VOLERROR)
TplotData_line<-T_Plot_DataAbs%>%group_by(Tsd)%>%summarise(meanErr=mean(VOLERROR, na.rm=T),
                                                           Max=(meanErr+sd(VOLERROR,na.rm=T)),
                                                           Min=(meanErr-sd(VOLERROR,na.rm=T)))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
T_plot_line<-ggplot()+geom_line(data=TplotData_line,aes(x=Tsd,y=meanErr))+
              geom_ribbon(data=TplotData_line,aes(x=Tsd,ymax=Max,ymin=Min),alpha=.5)+
              labs(
              y="Flow Volume Error (AF)",
              x="Time (T) Standard Deviation (Hour)")+
              theme_bw()

ggsave(filename ="tPlot_line.png",
       plot=T_plot_line,
       path="/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/outputs/sensitivity_analysis_pm/",
      device="png",
      width=6.5,
      height=5,
      units="in")

  ######Code for R Markdoum Display, Can be Deleted
  print("sensitivity_analysis_pm_TPlot_line")
```

```
## [1] "sensitivity_analysis_pm_TPlot_line"
```

```
  print(T_plot_line)
```




```
  #####

#CV Plot
cv_Plot_DataAbs<-do.call(rbind,plotList_cv)
cv_Plot_DataAbs$VOLERROR<-abs(cv_Plot_DataAbs$VOLERROR)
```

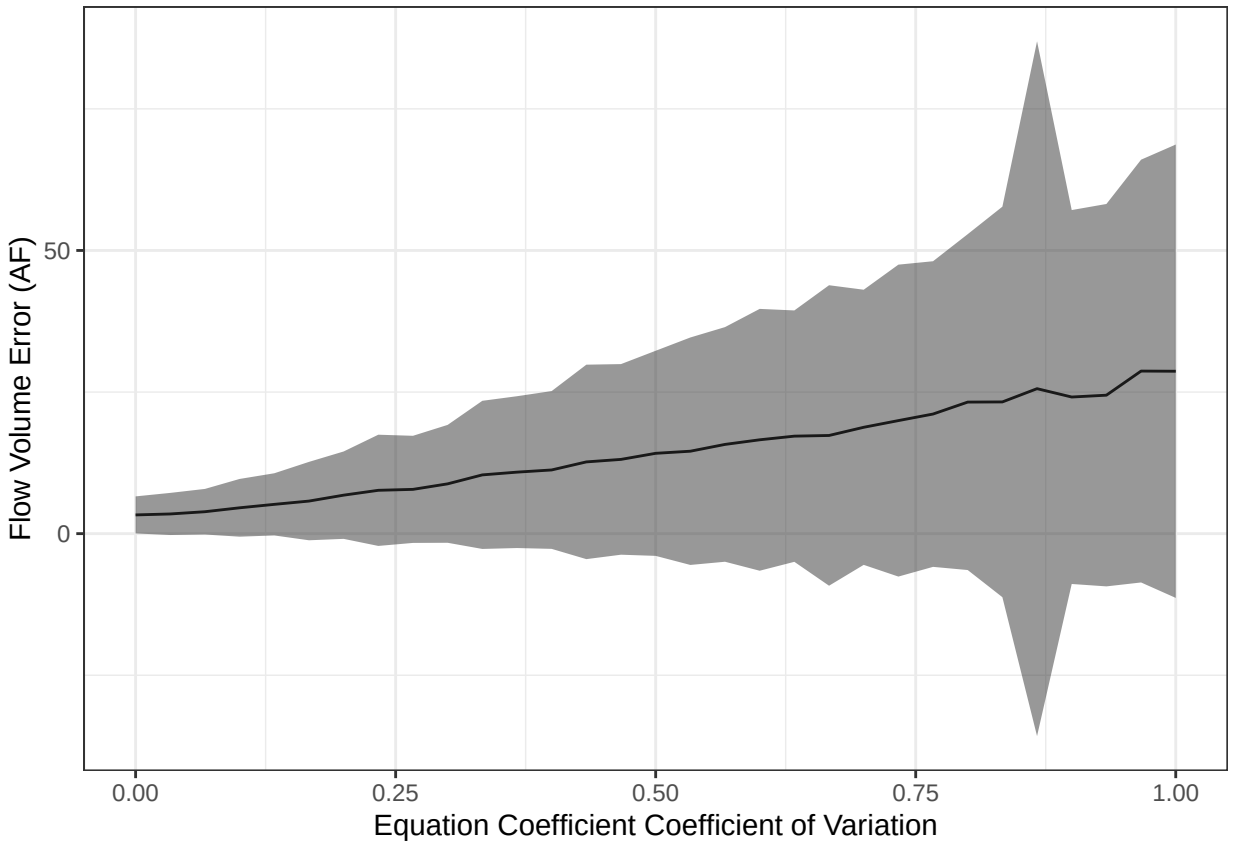
```
CVplotData_line<-cv_Plot_DataAbs%>%group_by(cv)%>%summarise(meanErr=mean(VOLERROR, na.rm=T),
                                                            Max=(meanErr+sd(VOLERROR,na.rm=T)),
                                                            Min=(meanErr-sd(VOLERROR,na.rm=T)))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
cv_plot_line<-ggplot()+geom_line(data=CVplotData_line,aes(x=cv,y=meanErr))+
               geom_ribbon(data=CVplotData_line,aes(x=cv,ymax=Max,ymin=Min),alpha=.5)+
              labs(
              y="Flow Volume Error (AF)",
              x="Equation Coefficient Coefficient of Variation")+
              theme_bw()

ggsave(filename ="cvPlot_line.png",
       plot=cv_plot_line,
       path="/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/outputs/sensitivity_analysis_pm/",
      device="png",
      width=6.5,
      height=5,
      units="in")

  ######Code for R Markdoum Display, Can be Deleted
  print("sensitivity_analysis_pm_cvPlot_line")
```

```
## [1] "sensitivity_analysis_pm_cvPlot_line"
```

```
  print(cv_plot_line)
```



#Old Code

6 Uncertainty at the Gate Level - Gates in Series

Now that the gate as been placed in a dynamic operating environment, the next step is to model multiple gates together. We do this in a few steps. First a "Toy Model" is built that is over a small, fake network. This was just built to look at how a network behaved. Then a larger network based on the physical site was built. Then different scenarios were run for the physical network.

6.1 Toy Model

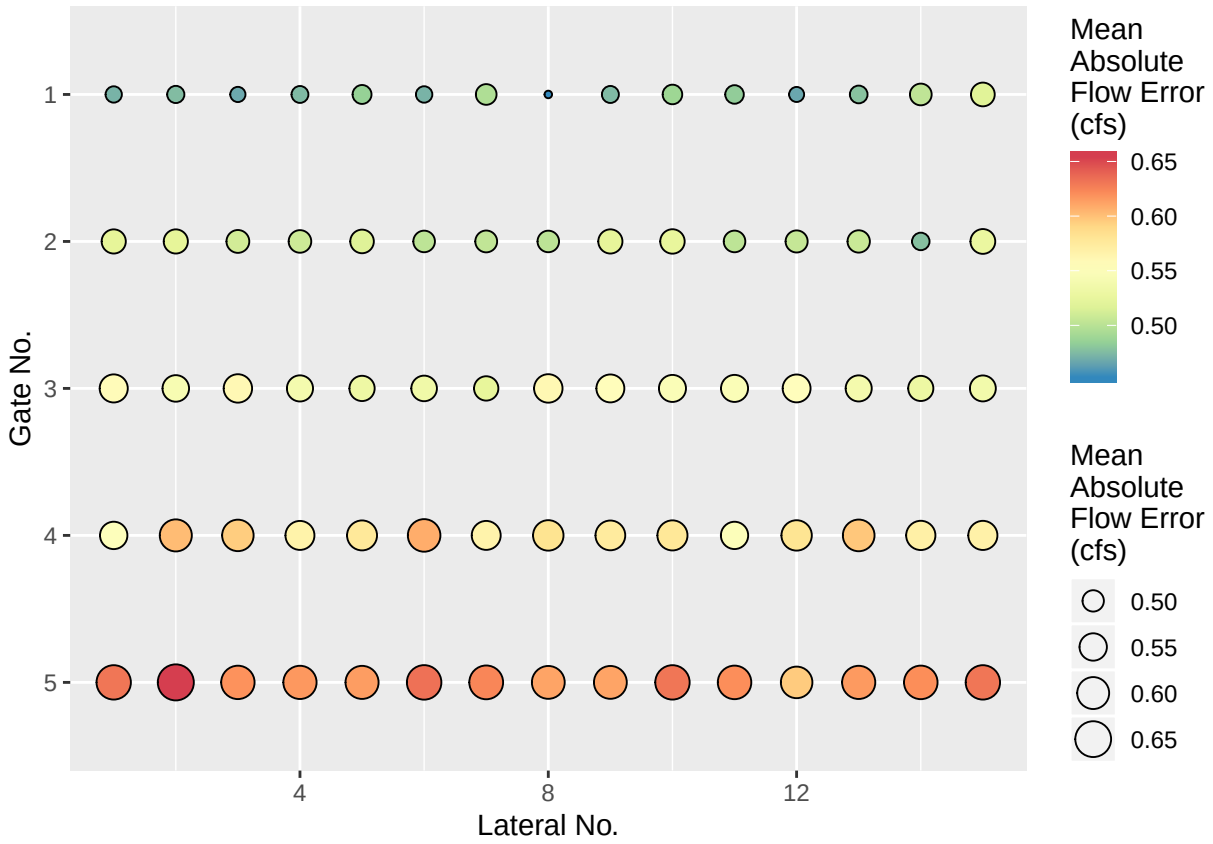
This model sets up a network of 15 lateral canals with 5 gates on them each. It plots a visual representation of the 75 gates as well as error scatter plots.

```
# Add option for choosing cv based on 6 or 12 %

# build a conceptual network for now, and later bring in the excel spreadsheet of
# the actual network nrow is max number of gates, ncol is number of laterals
maxgates <- 5
maxlaterals <- 15

network_gatetype <- data.frame(matrix(NA, nrow = maxgates, ncol = maxlaterals))
rownames(network_gatetype) <- paste0("G", 1:5)
colnames(network_gatetype) <- paste0("L", 1:15)
network_gatetype[, ] <- 1

network_hsd <- data.frame(matrix(NA, nrow = maxgates, ncol = maxlaterals))
rownames(network_hsd) <- paste0("G", 1:5)
colnames(network_hsd) <- paste0("L", 1:15)
network_hsd[1, ] <- 0.06
network_hsd[2, ] <- 0.08
network_hsd[3, ] <- 0.1
network_hsd[4, ] <- 0.12
network_hsd[5, ] <- 0.14

networkresult <- replicate(n = maxlaterals, expr = list())

# NOTE: only changing the uncertainty in upstream level based on location in the
# network change nsim
nsim <- 1000
for (l in 1:maxlaterals) {
    for (g in 1:maxgates) {
        di <- network_gatetype[g, l]
        hsdn <- network_hsd[g, l]
        networkresult[[l]][[g]] <- operationalerrorsim(dataindex = di, coefofvar = 0.02,
            optflowerrorp = 0.06, gsd = 0.1, hsd = hsdn, tsd = 0.25)
    }
}

# plot min, max, mean, first let's put it in a dataframe
toymod_results <- data.frame(do.call(rbind, unlist(networkresult, recursive = FALSE,
    use.names = TRUE)))
toymod_results$LATNUM <- rep(1:maxlaterals, times = 1, each = maxgates * nsim)  # for plotting on the x
toymod_results$GATENUM <- rep(1:maxgates, times = maxlaterals, each = nsim)  # for plotting on the y ax
```

```r
toymod_results$LATID <- colnames(network_gatetype)[toymod_results$LATNUM]
toymod_results$GATEID <- rownames(network_gatetype)[toymod_results$GATENUM]

# when calculating mean error the positives and negatives cancel out, so use
# absolute error instead. Same with min and max, more meaningful to disregard the
# error sign.
toymod_mean_flowerror <- aggregate(FLOWERROR ~ LATNUM + LATID + GATENUM + GATEID,
    data = toymod_results, FUN = mean, na.rm = TRUE)
colnames(toymod_mean_flowerror)[5] <- "MEANFLOWERROR"
toymod_abs_mean_flowerror <- aggregate(abs(FLOWERROR) ~ LATNUM + LATID + GATENUM +
    GATEID, data = toymod_results, FUN = mean, na.rm = TRUE)
colnames(toymod_abs_mean_flowerror)[5] <- "MEANABSFLOWERROR"
toymod_min_flowerror <- aggregate(abs(FLOWERROR) ~ LATNUM + LATID + GATENUM + GATEID,
    data = toymod_results, FUN = min, na.rm = TRUE)
colnames(toymod_min_flowerror)[5] <- "MINABSFLOWERROR"
toymod_max_flowerror <- aggregate(abs(FLOWERROR) ~ LATNUM + LATID + GATENUM + GATEID,
    data = toymod_results, FUN = max, na.rm = TRUE)
colnames(toymod_max_flowerror)[5] <- "MAXABSFLOWERROR"

# probability that the error condition is met
toymod_abs_mean_errorcondition <- aggregate(ERRORCONDITION ~ LATNUM + LATID + GATENUM +
    GATEID, data = toymod_results, FUN = mean, na.rm = TRUE)
toymod_abs_mean_errorcondition$ERRORCONDITIONNOTMET <- 1 - toymod_abs_mean_errorcondition$ERRORCONDITIO

# plot min, max, mean Reverse the order for plotting
toymod_abs_mean_flowerror$GATENUM <- factor(toymod_abs_mean_flowerror$GATENUM, levels = c("5",
    "4", "3", "2", "1"), ordered = TRUE)

library(ggplot2)
png("outputs/toymodel_mean_flowerror.png", width = 6.5, height = 4, units = "in",
    pointsize = 8, res = 300)
ggplot(toymod_abs_mean_flowerror, aes(x = LATNUM, y = GATENUM, size = MEANABSFLOWERROR,
    fill = MEANABSFLOWERROR)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectral") +
    labs(fill = "Mean \nAbsolute \nFlow Error \n(cfs)", size = "Mean \nAbsolute \nFlow Error \n(cfs)") +
    xlab("Lateral No.") + ylab("Gate No.")
dev.off()
```

```
## pdf
##   2
```

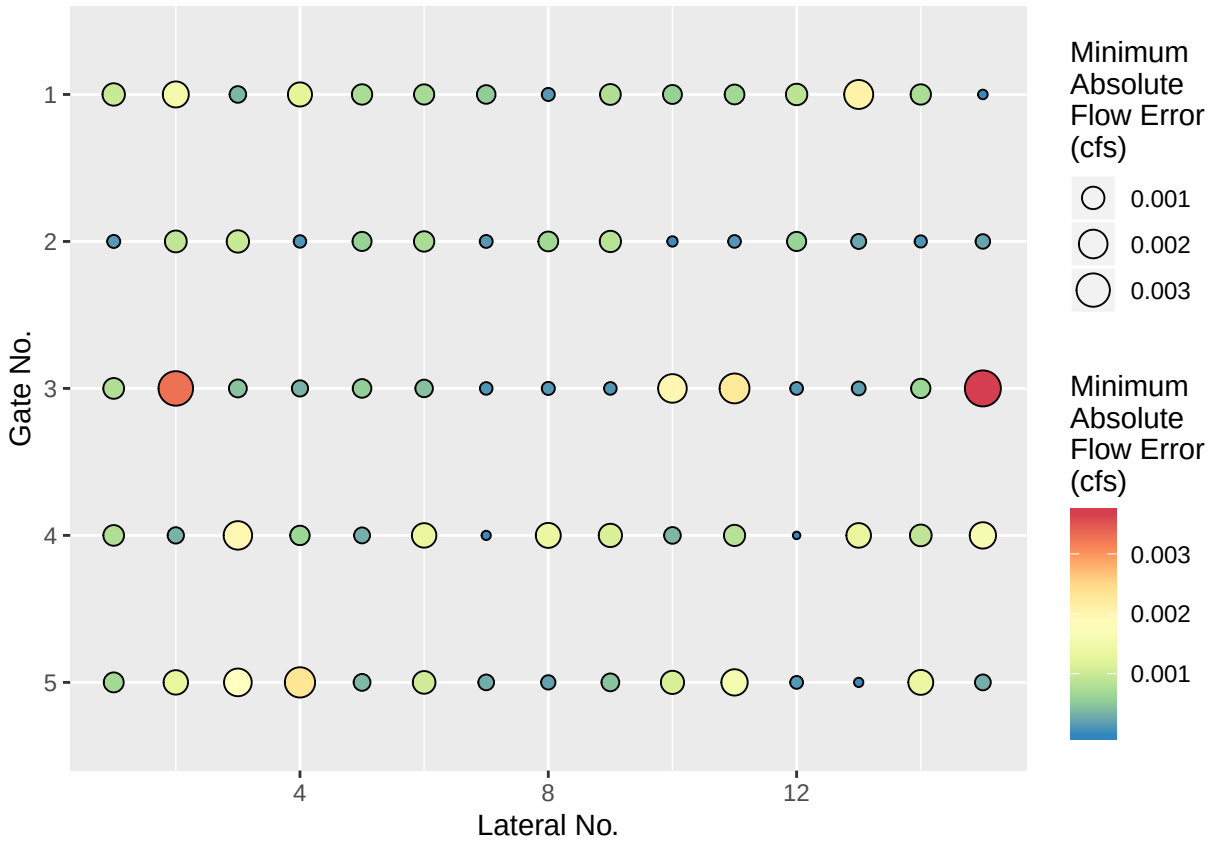
```r
###### Code for R Markdown Display, Can be Deleted
plot6a <- ggplot(toymod_abs_mean_flowerror, aes(x = LATNUM, y = GATENUM, size = MEANABSFLOWERROR,
    fill = MEANABSFLOWERROR)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectral") +
    labs(fill = "Mean \nAbsolute \nFlow Error \n(cfs)", size = "Mean \nAbsolute \nFlow Error \n(cfs)") +
    xlab("Lateral No.") + ylab("Gate No.")

print(plot6a)
```




```
########


png("outputs/toymodel_min_flowerror.png", width = 6.5, height = 4, units = "in",
    pointsize = 8, res = 300)
# Reverse the order for plotting
toymod_min_flowerror$GATENUM <- factor(toymod_min_flowerror$GATENUM, levels = c("5",
    "4", "3", "2", "1"), ordered = TRUE)
# Plot
ggplot(toymod_min_flowerror, aes(x = LATNUM, y = GATENUM, size = MINABSFLOWERROR,
    fill = MINABSFLOWERROR)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectral") +
    labs(fill = "Minimum \nAbsolute \nFlow Error \n(cfs)", size = "Minimum \nAbsolute \nFlow Error \n(c
    xlab("Lateral No.") + ylab("Gate No.")
dev.off()
```

```
## pdf
##   2
```

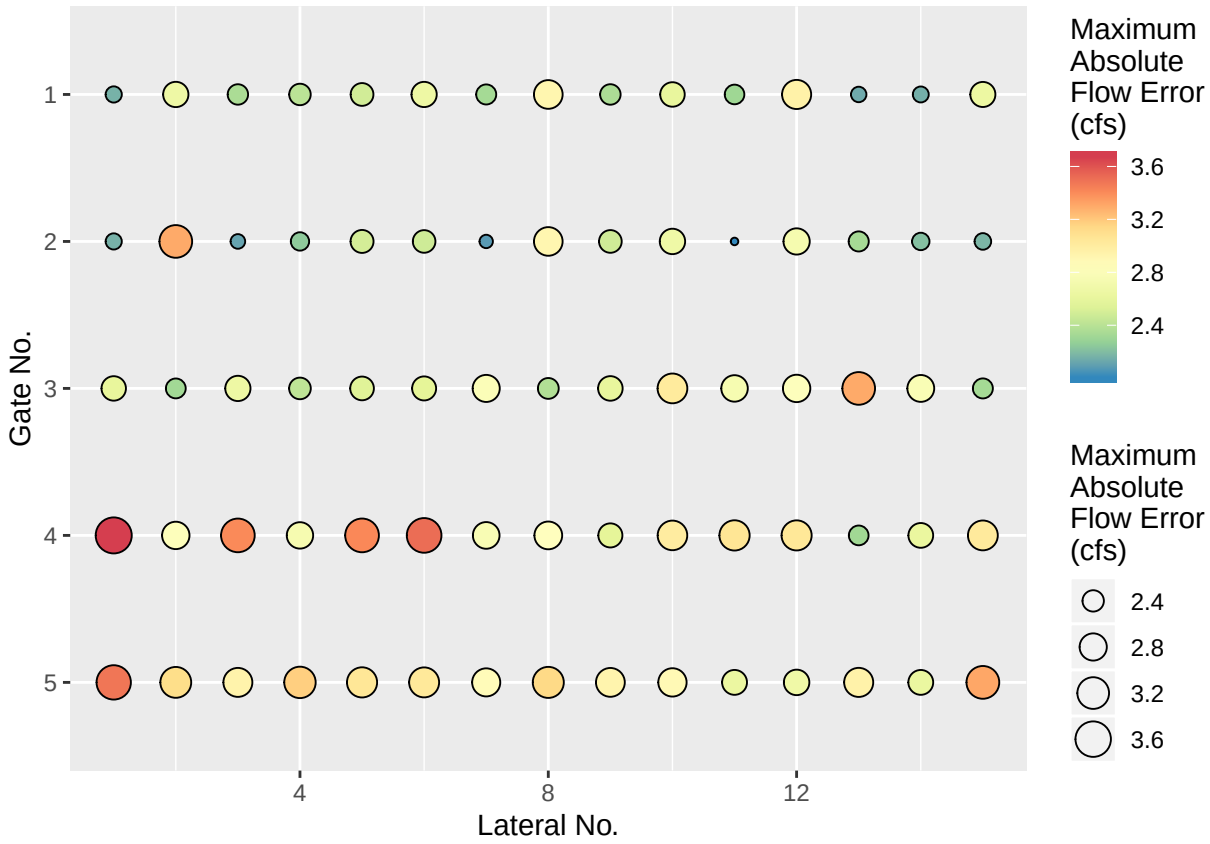
```
###### Code for R Markdown Display, Can be Deleted
plot6b <- ggplot(toymod_min_flowerror, aes(x = LATNUM, y = GATENUM, size = MINABSFLOWERROR,
    fill = MINABSFLOWERROR)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectral") +
    labs(fill = "Minimum \nAbsolute \nFlow Error \n(cfs)", size = "Minimum \nAbsolute \nFlow Error \n(c
    xlab("Lateral No.") + ylab("Gate No.")
print(plot6b)
```




```
###

png("outputs/toymodel_max_flowerror.png", width = 6.5, height = 4, units = "in",
    pointsize = 8, res = 300)
# Reverse the order for plotting
toymod_max_flowerror$GATENUM <- factor(toymod_max_flowerror$GATENUM, levels = c("5",
    "4", "3", "2", "1"), ordered = TRUE)
# Plot
ggplot(toymod_max_flowerror, aes(x = LATNUM, y = GATENUM, size = MAXABSFLOWERROR,
    fill = MAXABSFLOWERROR)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectral") +
    labs(fill = "Maximum \nAbsolute \nFlow Error \n(cfs)", size = "Maximum \nAbsolute \nFlow Error \n(c
    xlab("Lateral No.") + ylab("Gate No.")
dev.off()
```

```
## pdf
##   2
```

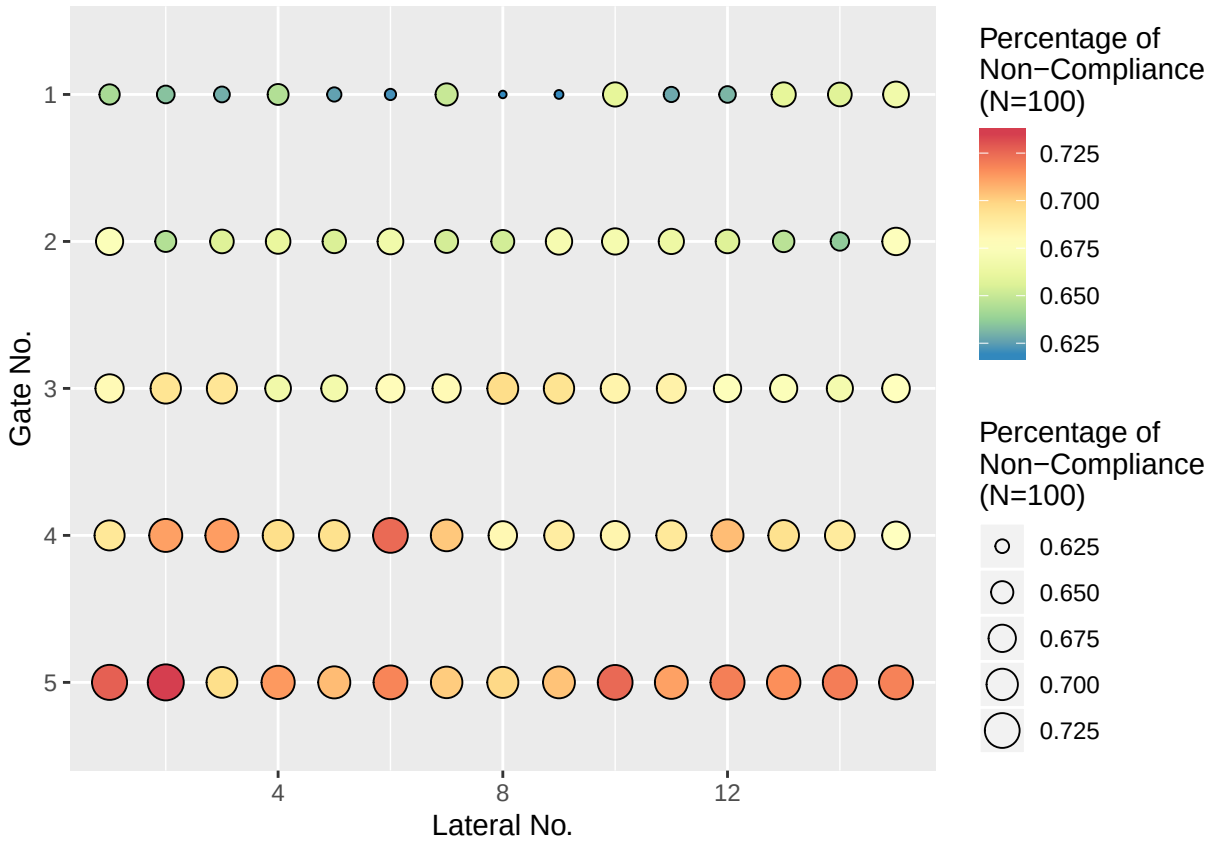
```
###### Code for R Markdown Display, Can be Deleted
plot6c <- ggplot(toymod_max_flowerror, aes(x = LATNUM, y = GATENUM, size = MAXABSFLOWERROR,
    fill = MAXABSFLOWERROR)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectral") +
    labs(fill = "Maximum \nAbsolute \nFlow Error \n(cfs)", size = "Maximum \nAbsolute \nFlow Error \n(c
    xlab("Lateral No.") + ylab("Gate No.")
print(plot6c)
```




```
########

png("outputs/toymodel_mean_noncompliancepercentage_flowerror.png", width = 6.5, height = 4,
    units = "in", pointsize = 8, res = 300)
# Reverse the order for plotting
toymod_abs_mean_errorcondition$GATENUM <- factor(toymod_abs_mean_errorcondition$GATENUM,
    levels = c("5", "4", "3", "2", "1"), ordered = TRUE)
# Plot
ggplot(toymod_abs_mean_errorcondition, aes(x = LATNUM, y = GATENUM, size = ERRORCONDITIONNOTMET,
    fill = ERRORCONDITIONNOTMET)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectral")
    labs(fill = "Percentage of \nNon-Compliance \n(N=100)", size = "Percentage of \nNon-Compliance \n(N=
    xlab("Lateral No.") + ylab("Gate No.")
dev.off()
```

```
## pdf
##   2
```

```
###### Code for R Markdown Display, Can be Deleted
plot6d <- ggplot(toymod_abs_mean_errorcondition, aes(x = LATNUM, y = GATENUM, size = ERRORCONDITIONNOTM
    fill = ERRORCONDITIONNOTMET)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectral")
    labs(fill = "Percentage of \nNon-Compliance \n(N=100)", size = "Percentage of \nNon-Compliance \n(N=
    xlab("Lateral No.") + ylab("Gate No.")
print(plot6d)
```




```
#######
```

6.2 Actual Network

This part of the code reads in the actual configuration of gates that exists in the test case, and builds a table to represent the layout of the gates.

```
# Set the number of Monte Carlo Runs
nsim <- 1000

# There was a note here about updating column names, not sure if that is a To-Do
# item? This is the second piece of input data. This data is a csv table that has
# the number of laterals and gates in the system. This data was extracted from
# GIS data (shapefiles) and put in a table.
gcid <- read.csv("/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/inputs/gcid_lateral_gates.csv")

# From observation there are a maximum of 72 gates for a single lateral.
maxgates <- 72
maxlaterals <- nrow(gcid)

# Setting up the network data frame
network_gatetype_gcid <- data.frame(matrix(NA, nrow = maxgates, ncol = maxlaterals))
rownames(network_gatetype_gcid) <- paste0("G", 1:maxgates)
colnames(network_gatetype_gcid) <- gcid$LATID
network_gatetype_gcid[, ] <- 1  # assume they all are gatetype=1 (this is allowing there to potential b

# Creating a cumulative vector for each lateral
network_numgates_gcid <- network_gatetype_gcid
```

```
network_numgates_gcid[, ] <- NA
for (i in 1:maxlaterals) {
    for (j in 1:gcid[i, "NUM_GATES_COL_COUNT"]) {
        network_numgates_gcid[j, i] <- j + gcid[i, "UPSTREAM_GATE_COUNT"]  # j is the gate count on the
    }
}
```

##6.3 Upstream Level Uncertainty Now that a dataframe representing the network exists, the next step is to assign uncertainty to the variables based on the network. Previously tsd, gsd, and hsd were the same for all gates. We are still leaving tsd and gsd as the same, but hsd, in reality, grows bigger as we move from the beginning of a lateral canal to the end.

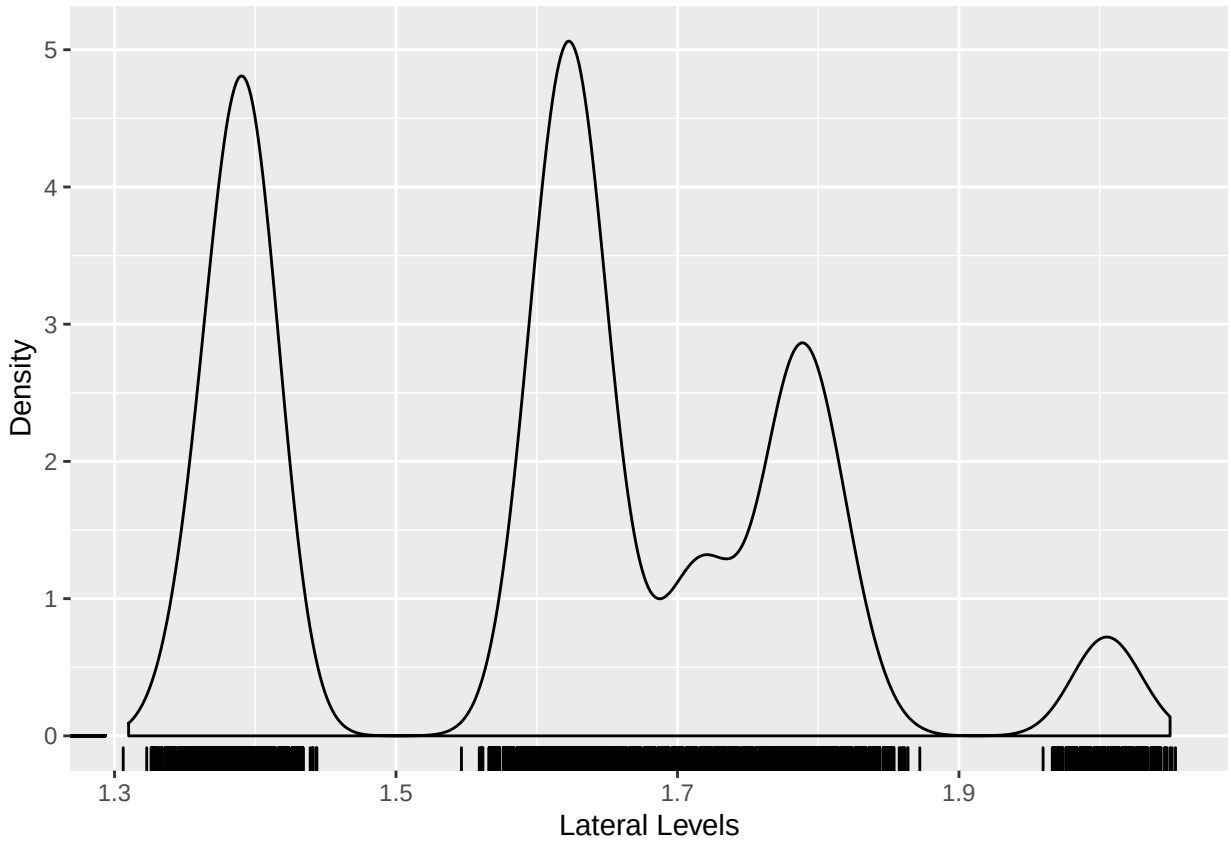
This model tries to replicate that by having two components to hsd. The first is the "baseline" hsd, that is a value that will be different depending if the lateral is "well behaved" (stable) or "non-well behaved" (chaotic). This chaos or stability is only regarding the water level. This type of behavior is dependent on the geometry of the lateral, how many gates there are, and the order of opening and closing the gates.

The second component is the gate-dependent hsd. This is the portion that increases as we move down the lateral. This component is important because the more gates are upstream of the model gate point, the more variability there is. This is because the upstream gates are being opened and closed at different times, and this impacts the water level.

```
# This data set is from field measurements showing the water level at the head of
# the lateral canal for three different laterals. The model doesn't need to
# directly follow this data, it just provides some example variance. At this
# point, though, this data is used in the next chunk of code to determine the
# baseline hsd.

latlvl <- read.csv("/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/inputs/lateral_levels_longforma

# These are then plotted.
png("outputs/lat_levels_density_48-1.png", width = 6.5, height = 4, units = "in",
    pointsize = 8, res = 300)
ggplot(latlvl[latlvl$LATID == "48-1", ], aes(VALUE)) + geom_density(alpha = 0.5,
    position = "stack") + geom_rug(aes(x = VALUE, y = 0), position = position_jitter(height = 0)) +
    xlab("Lateral Levels") + ylab("Density")
dev.off()
```

```
## pdf
##   2
```

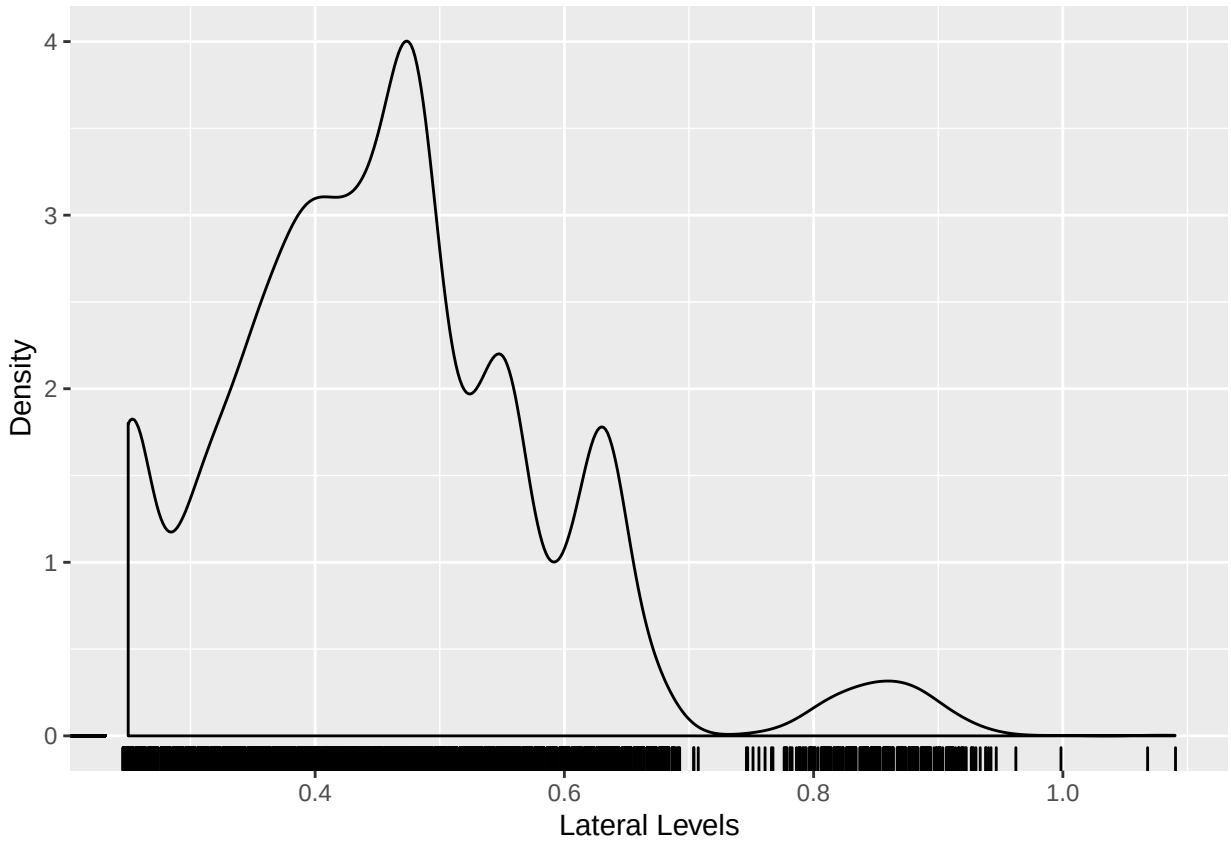
```
###### Code for R Markdown Display, Can be Deleted
plot6_3a <- ggplot(latlvl[latlvl$LATID == "48-1", ], aes(VALUE)) + geom_density(alpha = 0.5,
    position = "stack") + geom_rug(aes(x = VALUE, y = 0), position = position_jitter(height = 0)) +
    xlab("Lateral Levels") + ylab("Density")

print(plot6_3a)
```




```
######

png("outputs/lat_levels_density_22-1.png", width = 6.5, height = 4, units = "in",
    pointsize = 8, res = 300)
ggplot(latlvl[latlvl$LATID == "22-1", ], aes(VALUE)) + geom_density(alpha = 0.5,
    position = "stack") + geom_rug(aes(x = VALUE, y = 0), position = position_jitter(height = 0)) +
    xlab("Lateral Levels") + ylab("Density")
dev.off()
```

```
## pdf
##   2
```

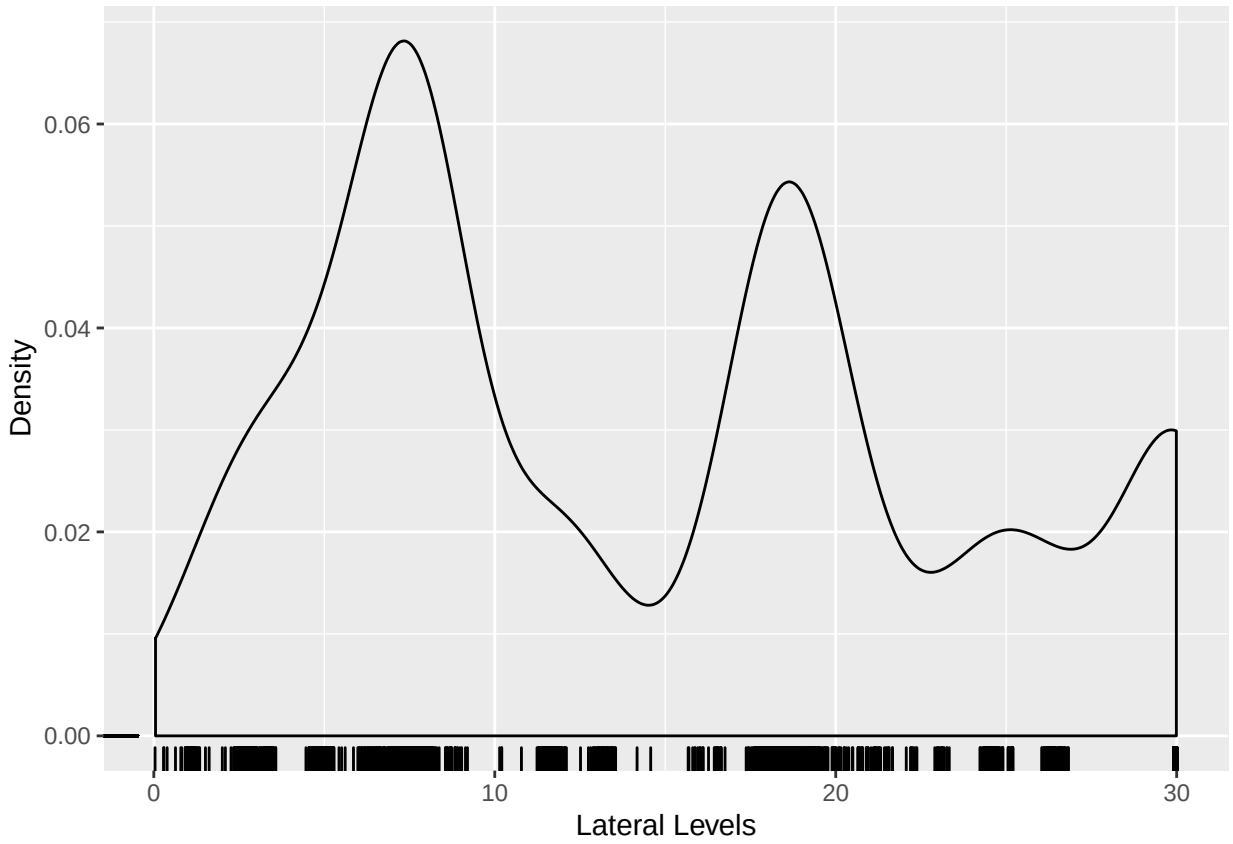
```
###### Code for R Markdown Display, Can be Deleted
plot6_3b <- ggplot(latlvl[latlvl$LATID == "22-1", ], aes(VALUE)) + geom_density(alpha = 0.5,
    position = "stack") + geom_rug(aes(x = VALUE, y = 0), position = position_jitter(height = 0)) +
    xlab("Lateral Levels") + ylab("Density")

print(plot6_3b)
```




```
######

png("outputs/lat_levels_density_21-4.png", width = 6.5, height = 4, units = "in",
    pointsize = 8, res = 300)
ggplot(latlvl[latlvl$LATID == "21-4", ], aes(VALUE)) + geom_density(alpha = 0.5,
    position = "stack") + geom_rug(aes(x = VALUE, y = 0), position = position_jitter(height = 0)) +
    xlab("Lateral Levels") + ylab("Density")
dev.off()
```

```
## pdf
##   2
```

```
###### Code for R Markdown Display, Can be Deleted
plot6_3c <- ggplot(latlvl[latlvl$LATID == "21-4", ], aes(VALUE)) + geom_density(alpha = 0.5,
    position = "stack") + geom_rug(aes(x = VALUE, y = 0), position = position_jitter(height = 0)) +
    xlab("Lateral Levels") + ylab("Density")

print(plot6_3c)
```




```
#######
```

##6.4 Upstream Level Uncertainty - 4 Cases This part of the model builds 4 cases for upstream level uncertainty: A stable lateral is Case 1, a chaotic lateral is Case 2. Accurate gates (6%) are A, less accurate gates (12%) are B. So there are four combinations: 1A, 1B, 2A, 2B. The only difference in the cases are the sd values for hsd (for Case 1 vs Case 2) and the cv (converted to sd) for the equation parameters (for Case A versus Case B).

```
# TODO: Figure out why some code is commented out of loops

# construct the cases using the data from lateral 48-1 and 22-1.  CASE I: Stable
# lateral, LATID=48-1
mean(latlvl[latlvl$LATID == "48-1", "VALUE"])
```

```
## [1] 1.605702
```

```
sd(latlvl[latlvl$LATID == "48-1", "VALUE"])
```

```
## [1] 0.1757031
```

```
max(latlvl[latlvl$LATID == "48-1", "VALUE"]) - min(latlvl[latlvl$LATID == "48-1",
    "VALUE"])
```

```
## [1] 0.74
```

```
# CASE II: Chaotic lateral, LATID=22-1
mean(latlvl[latlvl$LATID == "22-1", "VALUE"])
```

```
## [1] 0.4591362
```

```r
sd(latlvl[latlvl$LATID == "22-1", "VALUE"])
```

```
## [1] 0.1313706
```

```r
max(latlvl[latlvl$LATID == "22-1", "VALUE"]) - min(latlvl[latlvl$LATID == "22-1",
    "VALUE"])
```

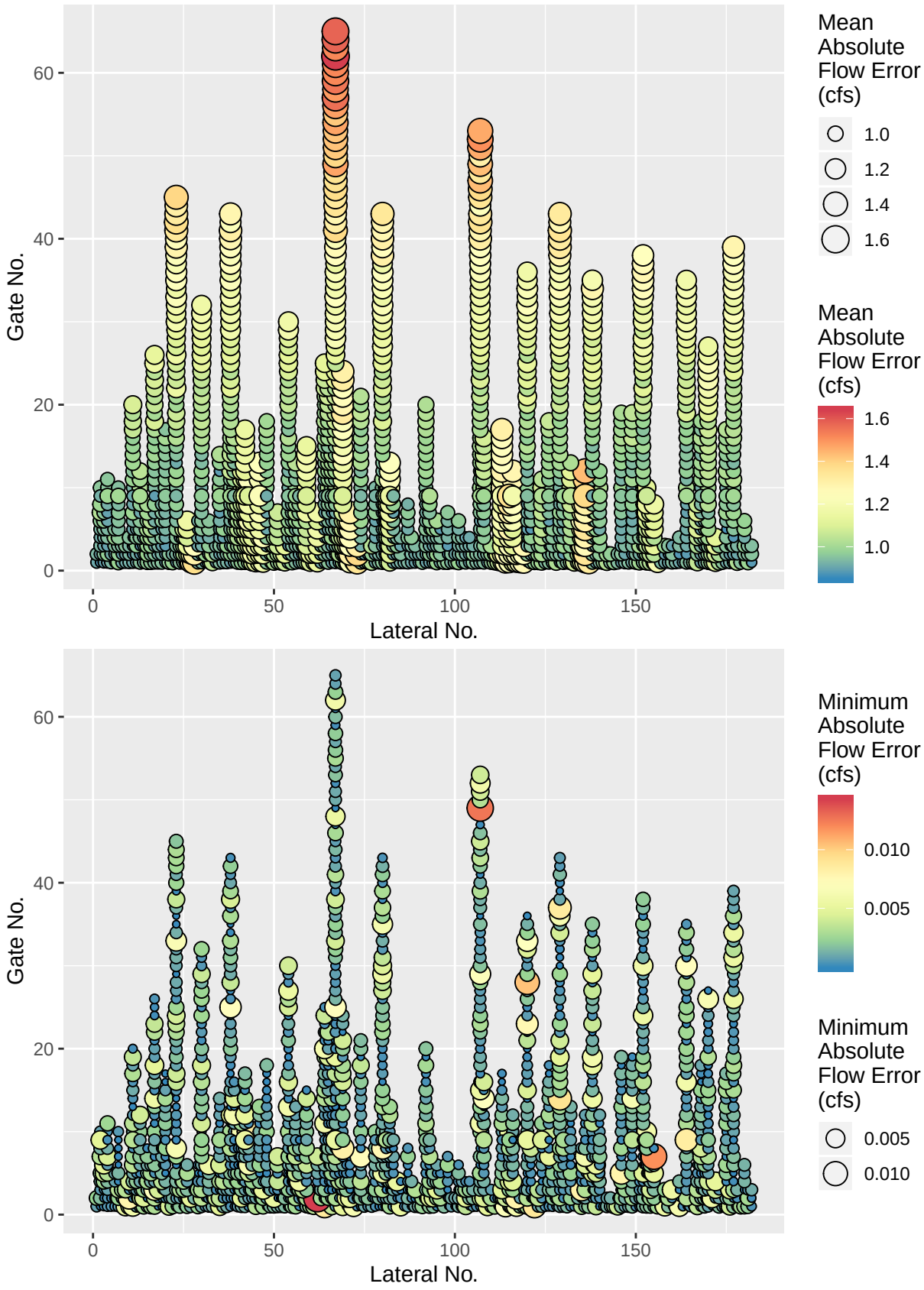
```
## [1] 0.84
```

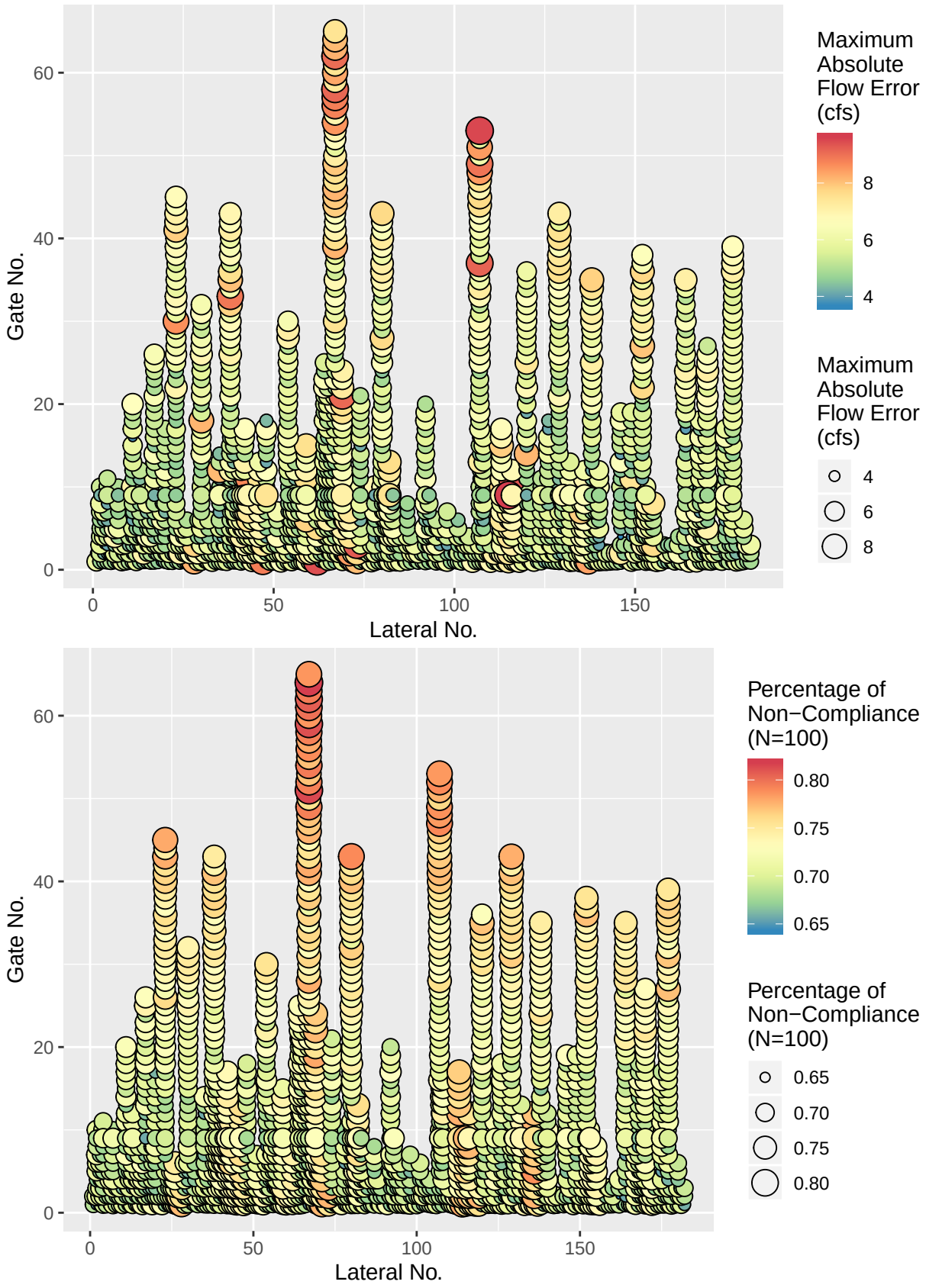
```r
# disregard LATID = 21-4 for now

network_hsd_gcid_case1 <- network_hsd_gcid_case2 <- network_gatetype_gcid
network_hsd_gcid_case1[, ] <- network_hsd_gcid_case2[, ] <- NA
for (i in 1:maxlaterals) {
    for (j in 1:maxgates) {
        # this 0.01 value will have to be tweaked after running this a few times. for now
        # just adjing the normalized standard deviation to the gate standard deviations.

        # Combine the two standard deviations using sdt = sqrt(sd1^2 + sd2^2)
        network_hsd_gcid_case1[j, i] <- ((0.01 * network_numgates_gcid[j, i])^2 +
            (sd(latlvl[latlvl$LATID == "48-1", "VALUE"])/mean(latlvl[latlvl$LATID ==
                "48-1", "VALUE"]) * 1)^2)^0.5
        network_hsd_gcid_case2[j, i] <- ((0.01 * network_numgates_gcid[j, i])^2 +
            (sd(latlvl[latlvl$LATID == "22-1", "VALUE"])/mean(latlvl[latlvl$LATID ==
                "22-1", "VALUE"]) * 1)^2)^0.5
    }
}

# construct a run for each case, and each scenario. scenario A is for 6% error
# (0.02 coefofvar) and scenario B is for 12% error (0.05 coefoffar). I want to
# change 0.05 to 0.06 I think.

networkresult_gcid_case1A <- networkresult_gcid_case2A <- networkresult_gcid_case1B <- networkresult_gc
    expr = list())
for (l in 1:maxlaterals) {
    for (g in 1:maxgates) {
        di <- network_gatetype_gcid[g, l]
        # CASE I (stable lateral), SCENARIO A (new, accurate)
        hsdn <- network_hsd_gcid_case1[g, l]
        networkresult_gcid_case1A[[l]][[g]] <- operationalerrorsim(dataindex = di,
            coefofvar = 0.02, optflowerrorp = 0.06, gsd = 0.1, hsd = hsdn, tsd = 0.25)
        # # CASE II (chaotic lateral), SCENARIO A (new, accurate)
        hsdn <- network_hsd_gcid_case2[g, l]
        networkresult_gcid_case2A[[l]][[g]] <- operationalerrorsim(dataindex = di,
            coefofvar = 0.02, optflowerrorp = 0.06, gsd = 0.1, hsd = hsdn, tsd = 0.25)
        # # CASE I (stable lateral), SCENARIO B (old, less accurate)
        hsdn <- network_hsd_gcid_case1[g, l]
        networkresult_gcid_case1B[[l]][[g]] <- operationalerrorsim(dataindex = di,
            coefofvar = 0.05, optflowerrorp = 0.1, gsd = 0.1, hsd = hsdn, tsd = 0.25)
        # # CASE II (chaotic lateral), SCENARIO B (old, less accurate)
        hsdn <- network_hsd_gcid_case2[g, l]
        networkresult_gcid_case2B[[l]][[g]] <- operationalerrorsim(dataindex = di,
            coefofvar = 0.05, optflowerrorp = 0.1, gsd = 0.1, hsd = hsdn, tsd = 0.25)
    }
}
```

##6.5 Plots of the results from the above code.

```
library(ggplot2)
network_postprocessing <- function(data, casenum, scenario) {
    # first let's put the data calculated above in a dataframe
    gcid_results <- data.frame(do.call(rbind, unlist(data, recursive = FALSE, use.names = TRUE)))
    gcid_results$LATNUM <- rep(1:maxlaterals, times = 1, each = maxgates * nsim)  # for plotting on the
    gcid_results$GATENUM <- rep(1:maxgates, times = maxlaterals, each = nsim)  # for plotting on the y
    gcid_results$LATID <- colnames(network_gatetype_gcid)[gcid_results$LATNUM]
    gcid_results$GATEID <- rownames(network_gatetype_gcid)[gcid_results$GATENUM]

    # when calculating mean error the positives and negatives cancel out, so use
    # absolute error instead. Same with min and max, more meaningful to disregard the
    # error sign.
    gcid_mean_flowerror <- aggregate(FLOWERROR ~ LATNUM + LATID + GATENUM + GATEID,
        data = gcid_results, FUN = mean, na.rm = TRUE)
    colnames(gcid_mean_flowerror)[5] <- "MEANFLOWERROR"
    gcid_abs_mean_flowerror <- aggregate(abs(FLOWERROR) ~ LATNUM + LATID + GATENUM +
        GATEID, data = gcid_results, FUN = mean, na.rm = TRUE)
    colnames(gcid_abs_mean_flowerror)[5] <- "MEANABSFLOWERROR"
    gcid_min_flowerror <- aggregate(abs(FLOWERROR) ~ LATNUM + LATID + GATENUM + GATEID,
        data = gcid_results, FUN = min, na.rm = TRUE)
    colnames(gcid_min_flowerror)[5] <- "MINABSFLOWERROR"
    gcid_max_flowerror <- aggregate(abs(FLOWERROR) ~ LATNUM + LATID + GATENUM + GATEID,
        data = gcid_results, FUN = max, na.rm = TRUE)
    colnames(gcid_max_flowerror)[5] <- "MAXABSFLOWERROR"

    # probability that the error condition is met
    gcid_abs_mean_errorcondition <- aggregate(ERRORCONDITION ~ LATNUM + LATID + GATENUM +
        GATEID, data = gcid_results, FUN = mean, na.rm = TRUE)
    gcid_abs_mean_errorcondition$ERRORCONDITIONNOTMET <- 1 - gcid_abs_mean_errorcondition$ERRORCONDITIO

    # plot min, max, mean
    gg1 <- ggplot(gcid_abs_mean_flowerror, aes(x = LATNUM, y = GATENUM, size = MEANABSFLOWERROR,
        fill = MEANABSFLOWERROR)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectral")
        labs(fill = "Mean \nAbsolute \nFlow Error \n(cfs)", size = "Mean \nAbsolute \nFlow Error \n(cfs
        xlab("Lateral No.") + ylab("Gate No.")

    png(paste0("outputs/gcid_mean_flowerror_C", casenum, "S", scenario, ".png"),
        width = 20, height = 8, units = "in", pointsize = 8, res = 300)
    plot(gg1)
    dev.off()

    ###### Code for R Markdown Display, Can be Deleted
    print(gg1)
    ####

    gg2 <- ggplot(gcid_min_flowerror, aes(x = LATNUM, y = GATENUM, size = MINABSFLOWERROR,
        fill = MINABSFLOWERROR)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectral") +
        labs(fill = "Minimum \nAbsolute \nFlow Error \n(cfs)", size = "Minimum \nAbsolute \nFlow Error
        xlab("Lateral No.") + ylab("Gate No.")

    png(paste0("outputs/gcid_min_flowerror_C", casenum, "S", scenario, ".png"), width = 20,
        height = 8, units = "in", pointsize = 8, res = 300)
```

```
    plot(gg2)
    dev.off()

    ###### Code for R Markdown Display, Can be Deleted
    print(gg2)
    ####

    gg3 <- ggplot(gcid_max_flowerror, aes(x = LATNUM, y = GATENUM, size = MAXABSFLOWERROR,
        fill = MAXABSFLOWERROR)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectral") +
        labs(fill = "Maximum \nAbsolute \nFlow Error \n(cfs)", size = "Maximum \nAbsolute \nFlow Error \
        xlab("Lateral No.") + ylab("Gate No.")

    png(paste0("outputs/gcid_max_flowerror_C", casenum, "S", scenario, ".png"), width = 20,
        height = 8, units = "in", pointsize = 8, res = 300)
    plot(gg3)
    dev.off()

    ###### Code for R Markdown Display, Can be Deleted
    print(gg3)
    ####

    gg4 <- ggplot(gcid_abs_mean_errorcondition, aes(x = LATNUM, y = GATENUM, size = ERRORCONDITIONNOTMET
        fill = ERRORCONDITIONNOTMET)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectra
        labs(fill = "Percentage of \nNon-Compliance \n(N=100)", size = "Percentage of \nNon-Compliance \
        xlab("Lateral No.") + ylab("Gate No.")

    png(paste0("outputs/gcid_mean_noncompliancepercentage_flowerror_C", casenum,
        "S", scenario, ".png"), width = 20, height = 8, units = "in", pointsize = 8,
        res = 300)
    plot(gg4)
    dev.off()

    ###### Code for R Markdown Display, Can be Deleted
    print(gg4)
    ####
}

# network_postprocessing(networkresult_gcid_case1A, '1', 'A')
# network_postprocessing(networkresult_gcid_case2A, '2', 'A')
# network_postprocessing(networkresult_gcid_case1B, '1', 'B')
network_postprocessing(networkresult_gcid_case2B, "2", "B")
```


Gate No.
Lateral No.
0
20
40
60
0
50
100
150
Mean Absolute Flow Error (cfs)
1.0
1.2
1.4
1.6
Mean Absolute Flow Error (cfs)
1.6
1.4
1.2
1.0
Minimum Absolute Flow Error (cfs)
0.010
0.005
Minimum Absolute Flow Error (cfs)
0.005
0.010


Gate No.
Lateral No.
0
20
40
60
0
50
100
150
Maximum Absolute Flow Error (cfs)
4
6
8
Maximum Absolute Flow Error (cfs)
4
6
8
Percentage of Non−Compliance (N=100)
0.65
0.70
0.75
0.80
Percentage of Non−Compliance (N=100)
0.65
0.70
0.75
0.80

7 Disaggregate results to be "per gate"

There are two remaining chunks of code, this one and the one following. These were the last two pieces built and the most "in-progress". They should run fully though. I am open to deleting them completely and rewriting them, if that's easier than deciphering them.

This section of the code reads in the amount of flow that passes through each gate, from field data, and disaggregates the results so that laterals that have more flow are assigned more error, i.e., the final plots of network look more realistic.

7.1 Disaggregateing the results and calculating statistics.

```
# Reading the data and assigning the flow per gate
lateralflow <- read.csv("/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/inputs/lateral_acre_flow.
lateralflow <- merge(lateralflow, gcid[, c("LATID", "NUM_GATES_COL_COUNT")], by = "LATID")
lateralflow$FLOW_PER_GATE <- lateralflow$WATER_DELIVERED_AF/lateralflow$NUM_GATES_COL_COUNT

network_disaggregated_flow <- function(data, casenum, scenario) {
    # first let's put the results in a dataframe to calculate the percentage of error
    # at each gate
    gcid_results <- data.frame(do.call(rbind, unlist(data, recursive = FALSE, use.names = TRUE)))
    gcid_results$LATNUM <- rep(1:maxlaterals, times = 1, each = maxgates * nsim)  # for plotting on the
    gcid_results$GATENUM <- rep(1:maxgates, times = maxlaterals, each = nsim)  # for plotting on the y
    gcid_results$LATID <- colnames(network_gatetype_gcid)[gcid_results$LATNUM]
    gcid_results$GATEID <- rownames(network_gatetype_gcid)[gcid_results$GATENUM]
    gcid_results$FLOWERRORP <- gcid_results$FLOWERROR/gcid_results$FLOWTRUTH

    # aggregate the mean error percentage
    gcid_mean_flowerrorp <- aggregate(FLOWERRORP ~ LATNUM + LATID + GATENUM + GATEID,
        data = gcid_results, FUN = mean, na.rm = TRUE)
    colnames(gcid_mean_flowerrorp)[5] <- "MEANFLOWERRORPERCENT"

    lateralflow <- merge(lateralflow, gcid_mean_flowerrorp, by = "LATID")
    lateralflow$FLOWERROR_PER_GATE <- lateralflow$FLOW_PER_GATE * lateralflow$MEANFLOWERRORPERCENT
    write.csv(lateralflow, paste0("outputs/lateral_flow_disaggregate_uniformly_C",
        casenum, "S", scenario, ".csv"))
    return(lateralflow)
}

network_disaggregated_flow_1A <- network_disaggregated_flow(networkresult_gcid_case1A,
    "1", "A")
network_disaggregated_flow_2A <- network_disaggregated_flow(networkresult_gcid_case2A,
    "2", "A")
network_disaggregated_flow_1B <- network_disaggregated_flow(networkresult_gcid_case1B,
    "1", "B")
network_disaggregated_flow_2B <- network_disaggregated_flow(networkresult_gcid_case2B,
    "2", "B")
```

#New Section 7.1

```
library(dplyr)

# This data set is from field measurements showing the water level at the head of
# the lateral canal for three different laterals. The model doesn't need to
# directly follow this data, it just provides some example variance. At this
```

```
# point, though, this data is used in the next chunk of code to determine the
# baseline hsd.

# I have included a spreadsheet with a rough cut of the different flows through
# different laterals. For now, all the gates are being treated the same, although
# ideally that would change if I get some data to show the differences there.

# Ouputs:<br /> 1.1. a csv that shows the mean and standard deviation / error for
# the flow through each lateral. <br /> 1.2 graph like
# 'gcid_mean_flowerror_C1SA'.<br />

# Set the number of Monte Carlo Runs
nsim <- 1000
# Sum of Squares Function
ssd <- function(x) sqrt(sum(x * x))
# There was a note here about updating column names, not sure if that is a To-Do
# item?

# This is the second piece of input data. This data is a csv table that has the
# number of laterals and gates in the system. This data was extracted from GIS
# data (shapefiles) and put in a table.
gcid <- read.csv("/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/inputs/gcid_lateral_gates.csv")

# From observation there are a maximum of 72 gates for a single lateral.
maxgates <- 72
maxlaterals <- nrow(gcid)

# Setting up the network data frame
network_gatetype_gcid <- data.frame(matrix(NA, nrow = maxgates, ncol = maxlaterals))
rownames(network_gatetype_gcid) <- paste0("G", 1:maxgates)
colnames(network_gatetype_gcid) <- gcid$LATID
network_gatetype_gcid[, ] <- 1  # assume they all are gatetype=1 (this is allowing there to potential b

# Creating a cumulative vector for each lateral
network_numgates_gcid <- network_gatetype_gcid
network_numgates_gcid[, ] <- NA
for (i in 1:maxlaterals) {
    for (j in 1:gcid[i, "NUM_GATES_COL_COUNT"]) {
        network_numgates_gcid[j, i] <- j + gcid[i, "UPSTREAM_GATE_COUNT"]  # j is the gate count on the
    }
}

## Get Data from LatLVL
latlvl <- read.csv("/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/inputs/lateral_levels_longforma
network_hsd_gcid_case1 <- network_hsd_gcid_case2 <- network_gatetype_gcid
network_hsd_gcid_case1[, ] <- network_hsd_gcid_case2[, ] <- NA

for (i in 1:maxlaterals) {
    for (j in 1:maxgates) {
        # this 0.01 value will have to be tweaked after running this a few times. for now
        # just adjing the normalized standard deviation to the gate standard deviations.
        # Combine the two standard deviations using sdt = sqrt(sd1^2 + sd2^2)
        network_hsd_gcid_case1[j, i] <- ((0.01 * network_numgates_gcid[j, i])^2 +
```

```
            (sd(latlvl[latlvl$LATID == "48-1", "VALUE"])/mean(latlvl[latlvl$LATID ==
                "48-1", "VALUE"]) * 1)^2)^0.5
        network_hsd_gcid_case2[j, i] <- ((0.01 * network_numgates_gcid[j, i])^2 +
            (sd(latlvl[latlvl$LATID == "22-1", "VALUE"])/mean(latlvl[latlvl$LATID ==
                "22-1", "VALUE"]) * 1)^2)^0.5
    }
}

### Setting up a run based on the total volume, this runs the loop only up to the
### point where we get 'a total volume for that gate'
volumeerrorsim <- function(dataindex, coefofvar = 0.02, optflowerrorp = 0.06, gsd = 0.1,
    hsd = 0.1, tsd = 0.25, totalvolume) {

    # dataindex is probably unnecessary with only one gate
    di <- dataindex

    # we can adj the ability to call different functions here, but for now just
    # equation #1 These coefficients were the same as above as far as I can tell, not
    # sure why they were repeated in the code

    atrue <- modelcoef[[1]]  # assume the mean is the true value
    astdv <- abs(coefofvar * modelcoef[[1]])
    anorm <- rnorm(nsim, modelcoef[[1]], astdv)  # samples around true value, creating error

    btrue <- modelcoef[[2]]
    bstdv <- abs(coefofvar * modelcoef[[2]])
    bnorm <- rnorm(nsim, modelcoef[[2]], bstdv)

    ctrue <- modelcoef[[3]]
    cstdv <- abs(coefofvar * modelcoef[[3]])
    cnorm <- rnorm(nsim, modelcoef[[3]], cstdv)

    dtrue <- modelcoef[[4]]
    dstdv <- abs(coefofvar * modelcoef[[4]])
    dnorm <- rnorm(nsim, modelcoef[[4]], dstdv)

    etrue <- modelcoef[[5]]
    estdv <- abs(coefofvar * modelcoef[[5]])
    enorm <- rnorm(nsim, modelcoef[[5]], estdv)

    aatrue <- modelcoef[[6]]
    aastdv <- abs(coefofvar * modelcoef[[6]])
    aanorm <- rnorm(nsim, modelcoef[[6]], aastdv)

    bbtrue <- modelcoef[[7]]
    bbstdv <- abs(coefofvar * modelcoef[[7]])
    bbnorm <- rnorm(nsim, modelcoef[[7]], bbstdv)

    cctrue <- modelcoef[[8]]
    ccstdv <- abs(coefofvar * modelcoef[[8]])
    ccnorm <- rnorm(nsim, modelcoef[[8]], ccstdv)

    ddtrue <- modelcoef[[9]]
```

```
ddstdv <- abs(coefofvar * modelcoef[[9]])
ddnorm <- rnorm(nsim, modelcoef[[9]], ddstdv)

# note that G and H are no longer certain here, let's assume a normal
# distribution of errors sample a uniform distribution instead of iteratting over
# all possible values
guniform <- runif(nsim, min(flowdfl$G), max(flowdfl$G))
huniform <- runif(nsim, min(flowdfl$H), max(flowdfl$H))
tuniform <- runif(nsim, 6, 12)  # in hours, changed to between 6 and 12

# now assign an error to each sampled G and H, adjed t for volume calcs
tnorm <- gnorm <- hnorm <- vector()
for (i in 1:nsim) {
    gnorm[i] <- rnorm(1, guniform[i], gsd)
    hnorm[i] <- rnorm(1, huniform[i], hsd)
    tnorm[i] <- rnorm(1, tuniform[i], tsd)
}

volsim <- voltruth <- flowsim <- flowtruth <- vector()
# SetUpVolume Counter
volumeCounter <- 0
i <- 1
while (volumeCounter < totalvolume) {
    # we can adj the ability to call different functions here, but for now just
    # equation #1
    flowsim[i] <- (anorm[i] + bnorm[i] * gnorm[i] + cnorm[i] * gnorm[i]^2 + dnorm[i] *
        gnorm[i]^3 + enorm[i] * log(hnorm[i]))/(1 + aanorm[i] * gnorm[i] + bbnorm[i] *
        log(hnorm[i]) + ccnorm[i] * (log(hnorm[i]))^2 + ddnorm[i] * (log(hnorm[i]))^3)

    flowtruth[i] <- (atrue + btrue * guniform[i] + ctrue * guniform[i]^2 + dtrue *
        guniform[i]^3 + etrue * log(huniform[i]))/(1 + aatrue * gnorm[i] + bbtrue *
        log(huniform[i]) + cctrue * (log(huniform[i]))^2 + ddtrue * (log(huniform[i]))^3)

    volsim[i] <- flowsim[i] * tnorm[i]

    voltruth[i] <- flowtruth[i] * tuniform[i]
    volumeCounter <- voltruth[i] + volumeCounter
    # print(i)
    i <- i + 1
}

gSDvec <- gnorm[1:(i - 1)]
hSDvec <- hnorm[1:(i - 1)]
tSDvec <- tnorm[1:(i - 1)]

flowerror <- flowtruth - flowsim
optflowerror <- optflowerrorp * flowtruth
errorconditioncol <- ifelse(abs(flowerror) < optflowerror, 1, 0)
volerror <- voltruth - volsim
optvolerror <- optflowerrorp * voltruth
volerrorconditioncol <- ifelse(abs(optvolerror) < optflowerror, 1, 0)

# conceptual exercise
```

```
    resultsdf <- cbind(FLOWTRUTH = flowtruth, FLOWSIM = flowsim, FLOWERROR = flowerror,
        OPTFLOW = optflowerror, ERRORCONDITION = errorconditioncol, VOLTRUTH = voltruth,
        VOLSIM = volsim, VOLERROR = volerror, OPTVOL = optvolerror, VOLERRORCONDITION = volerrorconditi
        gSDvec = gSDvec, hSDvec = hSDvec, tSDvec = tSDvec)
    return(resultsdf)
}

# Reading the data and assigning the flow per gate
lateralflow <- read.csv("/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/inputs/lateral_acre_flow.
totalGatesGCID <- gcid %>% group_by(PARENT) %>% summarise(NUM_GATES_COL_COUNT = sum(NUM_GATES_COL_COUNT
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
colnames(totalGatesGCID)[1] <- "LATID"
lateralflow <- merge(lateralflow, totalGatesGCID[, c("LATID", "NUM_GATES_COL_COUNT")],
    by = "LATID", all = T)
lateralflow$FLOW_PER_GATE <- lateralflow$WATER_DELIVERED_AF/lateralflow$NUM_GATES_COL_COUNT

# Set Up Per Gate Water Limits in the needed structre Need to deal with subgates

network_volumepergate_gcid <- data.frame(network_numgates_gcid, check.names = F)
GateNames <- colnames(network_volumepergate_gcid)
GateNames[1:176] <- gsub("([[:alpha:]])", "", colnames(network_volumepergate_gcid)[1:176])
GateNames[1:176] <- substr(GateNames[1:176], 1, 5)
GateNames[177:182] <- " ESC"

for (i in 1:ncol(network_volumepergate_gcid)) {
    # Put the average flow per gate
    network_volumepergate_gcid[!is.na(network_volumepergate_gcid[, i]), i] <- lateralflow$FLOW_PER_GATE
        lateralflow$LATID)]
    # print(i)
}

# construct a run for each case, and each scenario. scenario A is for 6% error
# (0.02 coefofvar) and scenario B is for 12% error (0.05 coefoffar).  I want to
# change 0.05 to 0.06 I think. NumGates
numgates <- melt(network_numgates_gcid)
```

```
## No id variables; using all as measure variables
```

```
colnames(numgates)[1] <- "LATID"
colnames(numgates)[2] <- "GATENUM"
# FlowPerGate
volume_data <- melt(network_volumepergate_gcid)
```

```
## No id variables; using all as measure variables
```

```
colnames(volume_data)[1] <- "LATID"
colnames(volume_data)[2] <- "FLOW_PER_GATE"
gatedata_long <- cbind(numgates, volume_data$FLOW_PER_GATE)
# HSDN
hsdn_data <- melt(network_hsd_gcid_case1)
```

```
## No id variables; using all as measure variables
```

```r
colnames(hsdn_data)[1] <- "LATID"
colnames(hsdn_data)[2] <- "hsdn"
gatedata_long <- cbind(gatedata_long, hsdn_data$hsdn)
# GateType
gatetype_data <- melt(network_gatetype_gcid)
```

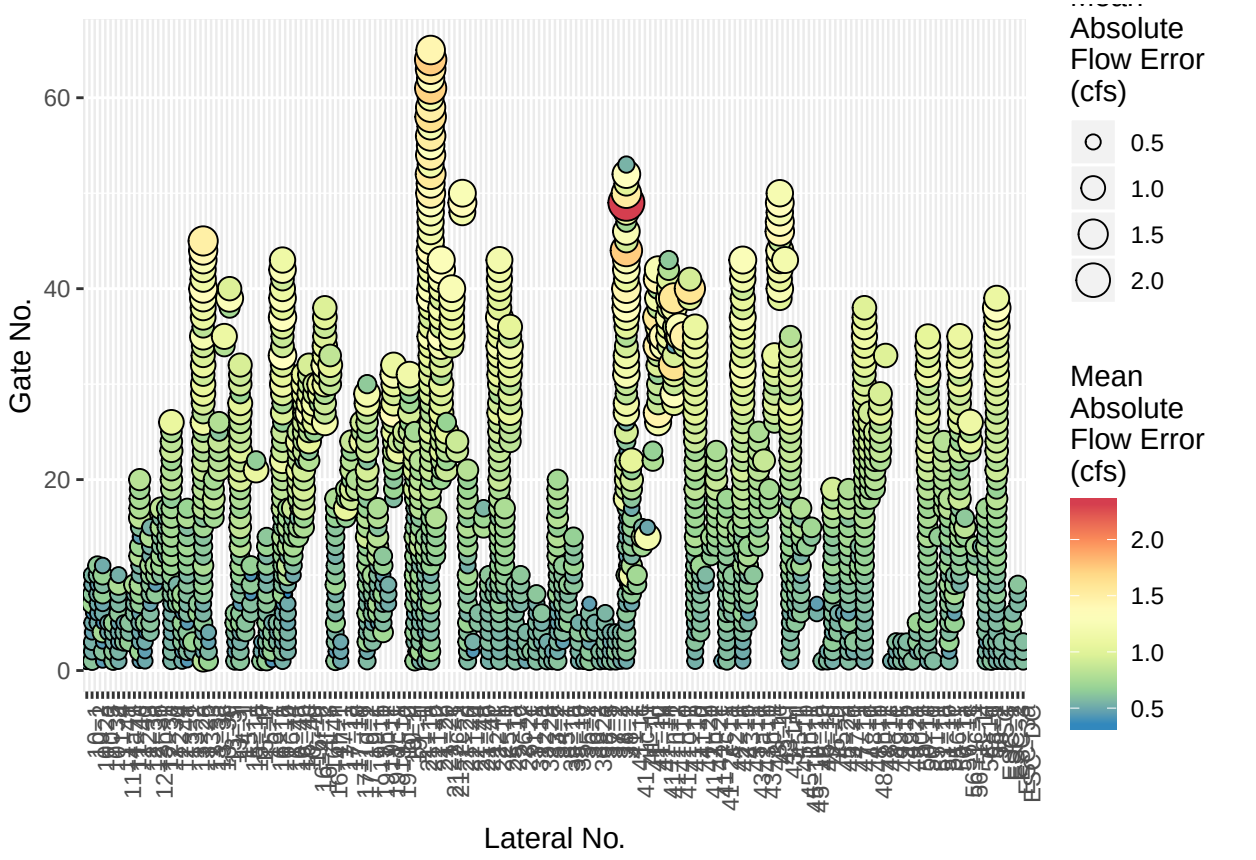
```
## No id variables; using all as measure variables
```

```r
colnames(gatetype_data)[1] <- "LATID"
colnames(gatetype_data)[2] <- "gatetype"
gatedata_long <- cbind(gatedata_long, gatetype_data$gatetype)
# RenameColum
colnames(gatedata_long)[3] <- "FLOW_PER_GATE"
colnames(gatedata_long)[4] <- "hsdn"
colnames(gatedata_long)[5] <- "gatetype"
# Remove Missing Gates
gatedata_long <- gatedata_long[!is.na(gatedata_long$GATENUM), ]
gatedata_long <- gatedata_long[!is.na(gatedata_long$FLOW_PER_GATE), ]


# Remove Sub Laterals
laterals <- unique(gatedata_long$LATID)
# laterals<-laterals[-grep('([[:alpha:]])',laterals)] Add back in ESC
# laterals<-c(as.character(laterals),' ESC')
gatedata_long_ml <- gatedata_long[gatedata_long$LATID %in% laterals, ]

# Set the total number of loops Run loop
networkresult_gcid_volume <- list()
network_all <- list()
for (h in 1:10) {

    for (j in 1:nrow(gatedata_long_ml)) {
        di <- gatedata_long_ml$gatetype[j]
        hsdn <- gatedata_long_ml$hsdn[j]
        totalvolumen <- gatedata_long_ml$FLOW_PER_GATE[j]
        result <- data.frame(volumeerrorsim(dataindex = di, coefofvar = 0.02, optflowerrorp = 0.06,
            gsd = 0.1, hsd = hsdn, tsd = 0.25, totalvolume = totalvolumen))
        result$LATID <- gatedata_long_ml$LATID[j]
        result$GATENUM <- gatedata_long_ml$GATENUM[j]
        result$FLOWERRORP <- result$FLOWERROR/result$FLOWTRUTH
        networkresult_gcid_volume[[j]] <- result
        # print(j)
    }

    gcid_results <- rbindlist(networkresult_gcid_volume)

    # Summarise the run
    loopResult <- gcid_results

    network_all[[h]] <- loopResult
    # print(paste0('H-',h))
}

# EXAMPLE Plots
```

```
plotData <- rbindlist(network_all)


plotData_final <- plotData %>% group_by(LATID, GATENUM) %>% summarise(FLOWERROR = mean(abs(FLOWERROR),
    na.rm = T), VOLTRUTH = sum(VOLTRUTH, na.rm = T), hsd = sqrt(sum(hSDvec^2)), tsd = sqrt(sum(tSDvec^2
    gsd = sqrt(sum(gSDvec^2)))
```

```
## `summarise()` regrouping output by 'LATID' (override with `.groups` argument)
```

```
library(ggplot2)
section7chart <- ggplot(plotData_final, aes(x = LATID, y = GATENUM, size = FLOWERROR,
    fill = FLOWERROR)) + geom_point(shape = 21) + scale_fill_distiller(palette = "Spectral") +
    labs(fill = "Mean \nAbsolute \nFlow Error \n(cfs)", size = "Mean \nAbsolute \nFlow Error \n(cfs)") +
    xlab("Lateral No.") + ylab("Gate No.") + theme(axis.text.x = element_text(angle = 90,
    hjust = 1))


print(section7chart)
```



#8 New Section Experimenting with Error Propegation

So, what I would like to do is write this code in a kind of "dummy" version, where I just choose some inputs, and then those inputs will change once I run the flow simulations.

The goal of this model is to run the model as it exists with two changes. One change is to create a "stable" and "chaotic" lateral scheme. The idea is that some laterals have water levels that fluctuate, and some do not. It's based on the geometry, if there are any long crested weirs, and how deliveries are run. So for the stable lateral we can pick an initial USL sd of, say, 0.2 ft. For the chaotic lateral let's pick 0.5 ft. That is

what I'm calling the "initial" USL sd. Then, I also want to have an USL sd that increases as we move down the lateral. This will be a function of how many gates are upstream of a given gate.

Calculating that is a couple steps, but not too complicated I think. In the input "gcid_lateral_gates.csv" all the laterals are laid out. There are two columns that apply here. The values in column "L" are for how many gates are upstream of each lateral. This is because the laterals branch off each other. Column "C", that you don't need to use here, shows which lateral is the "parent" of each sub-lateral. So I counted the gates above the junction for each lateral, and that's column "L". The other column you need to use is column "H". Column "H" shows how many gates there are along the lateral in question. So, to calculate the USL sd for each gate in the system, we would need to combine the initial USL sd with (USL per gate)*(number of upstream gates). I believe these numbers are combined by taking the square root of the sum of the squares? For the stable lateral, let's have USL_sd_ngates = 0.01/gate and for the chaotic lateral let's try 0.05/gate. That might be too much, but let's start there.

```
# Set the number of Monte Carlo Runs
nsim <- 1000

# There was a note here about updating column names, not sure if that is a To-Do
# item? This is the second piece of input data. This data is a csv table that has
# the number of laterals and gates in the system. This data was extracted from
# GIS data (shapefiles) and put in a table.
gcid <- read.csv("/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/inputs/gcid_lateral_gates.csv")

# From observation there are a maximum of 72 gates for a single lateral.
maxgates <- 72
maxlaterals <- nrow(gcid)

# Setting up the network data frame
network_gatetype_gcid <- data.frame(matrix(NA, nrow = maxgates, ncol = maxlaterals))
rownames(network_gatetype_gcid) <- paste0("G", 1:maxgates)
colnames(network_gatetype_gcid) <- gcid$LATID
network_gatetype_gcid[, ] <- 1  # assume they all are gatetype=1 (this is allowing there to potential b

# Creating a cumulative vector for each lateral
network_numgates_gcid <- network_gatetype_gcid
network_numgates_gcid[, ] <- NA
for (i in 1:maxlaterals) {
    for (j in 1:gcid[i, "NUM_GATES_COL_COUNT"]) {
        network_numgates_gcid[j, i] <- j + gcid[i, "UPSTREAM_GATE_COUNT"]  # j is the gate count on the
    }
}

## Get Data from LatLVL
latlvl <- read.csv("/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/inputs/lateral_levels_longforma
network_hsd_gcid_case1 <- network_hsd_gcid_case2 <- network_gatetype_gcid
network_hsd_gcid_case1[, ] <- network_hsd_gcid_case2[, ] <- NA

for (i in 1:maxlaterals) {
    for (j in 1:maxgates) {
        # this 0.01 value will have to be tweaked after running this a few times. for now
        # just adjing the normalized standard deviation to the gate standard deviations.
        # Combine the two standard deviations using sdt = sqrt(sd1^2 + sd2^2)
        network_hsd_gcid_case1[j, i] <- ((0.01 * network_numgates_gcid[j, i])^2 +
            (sd(latlvl[latlvl$LATID == "48-1", "VALUE"])/mean(latlvl[latlvl$LATID ==
                "48-1", "VALUE"]) * 1)^2)^0.5
```

```
        network_hsd_gcid_case2[j, i] <- ((0.01 * network_numgates_gcid[j, i])^2 +
            (sd(latlvl[latlvl$LATID == "22-1", "VALUE"])/mean(latlvl[latlvl$LATID ==
                "22-1", "VALUE"]) * 1)^2)^0.5
    }
}


# Reading the data and assigning the flow per gate
lateralflow <- read.csv("/Users/TheVault/Desktop/Upwork/Dissertation/PM_Edits/inputs/lateral_acre_flow.
totalGatesGCID <- gcid %>% group_by(PARENT) %>% summarise(NUM_GATES_COL_COUNT = sum(NUM_GATES_COL_COUNT
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
colnames(totalGatesGCID)[1] <- "LATID"
lateralflow <- merge(lateralflow, totalGatesGCID[, c("LATID", "NUM_GATES_COL_COUNT")],
    by = "LATID", all = T)
lateralflow$FLOW_PER_GATE <- lateralflow$WATER_DELIVERED_AF/lateralflow$NUM_GATES_COL_COUNT

# Set Up Per Gate Water Limits in the needed structre Need to deal with subgates,
# right now assuming that they are cumulative
network_volumepergate_gcid <- data.frame(network_numgates_gcid, check.names = F)
GateNames <- colnames(network_volumepergate_gcid)
GateNames[1:176] <- gsub("([[:alpha:]])", "", colnames(network_volumepergate_gcid)[1:176])
GateNames[1:176] <- substr(GateNames[1:176], 1, 5)
GateNames[177:182] <- " ESC"

for (i in 1:ncol(network_volumepergate_gcid)) {
    # Put the average flow per gate
    network_volumepergate_gcid[!is.na(network_volumepergate_gcid[, i]), i] <- lateralflow$FLOW_PER_GATE
        lateralflow$LATID)]
    # print(i)
}

# construct a run for each case, and each scenario. scenario A is for 6% error
# (0.02 coefofvar) and scenario B is for 12% error (0.05 coefoffar). I want to
# change 0.05 to 0.06 I think. NumGates
numgates <- melt(network_numgates_gcid)
```

```
## No id variables; using all as measure variables
```

```
colnames(numgates)[1] <- "LATID"
colnames(numgates)[2] <- "GATENUM"
# FlowPerGate
volume_data <- melt(network_volumepergate_gcid)
```

```
## No id variables; using all as measure variables
```

```
colnames(volume_data)[1] <- "LATID"
colnames(volume_data)[2] <- "FLOW_PER_GATE"
gatedata_long <- cbind(numgates, volume_data$FLOW_PER_GATE)
# Number of UpstreamGases
gatedata_long$upstreamGates <- gcid$UPSTREAM_GATE_COUNT[match(gatedata_long$LATID,
    gcid$LATID)]
# HSDN hsdn_data<-melt(network_hsd_gcid_case1) colnames(hsdn_data)[1]<-'LATID'
# colnames(hsdn_data)[2]<-'hsdn'
```

```
# gatedata_long<-cbind(gatedata_long,hsdn_data$hsdn)

# GateType
gatetype_data <- melt(network_gatetype_gcid)
```

```
## No id variables; using all as measure variables
```

```
colnames(gatetype_data)[1] <- "LATID"
colnames(gatetype_data)[2] <- "gatetype"
gatedata_long <- cbind(gatedata_long, gatetype_data$gatetype)
# RenameColum
colnames(gatedata_long)[3] <- "FLOW_PER_GATE"
colnames(gatedata_long)[5] <- "gatetype"
colnames(gatedata_long)[4] <- "UpstreamGates"
# Remove Missing Gates
gatedata_long <- gatedata_long[!is.na(gatedata_long$GATENUM), ]
gatedata_long <- gatedata_long[!is.na(gatedata_long$FLOW_PER_GATE), ]


# Pick Initial SD
gatedata_long$USL_sd_ngates <- NA
gatedata_long$UCL_sd_ngates <- NA
for (i in 1:nrow(gatedata_long)) {
    gatedata_long$USL_sd_ngates[i] <- ssd(c(0.2, rep(0.2, gatedata_long$UpstreamGates[i])))
    gatedata_long$UCL_sd_ngates[i] <- ssd(c(0.5, rep(0.5, gatedata_long$UpstreamGates[i])))
}

sectionEightGateDataUSL <- list()
sectionEightGateDataUCL <- list()
for (i in 1:nrow(gatedata_long)) {
    sectionEightGateDataUCL[[i]] <- operationalerrorsim(dataindex = di, coefofvar = 0.05,
        optflowerrorp = 0.1, gsd = 0.1, hsd = gatedata_long$UCL_sd_ngates[i], tsd = 0.25)
    sectionEightGateDataUSL[[i]] <- operationalerrorsim(dataindex = di, coefofvar = 0.05,
        optflowerrorp = 0.1, gsd = 0.1, hsd = gatedata_long$USL_sd_ngates[i], tsd = 0.25)
    # print(i)
}
```

#8 Old Experimenting with Error Propegation

This section should probably be dropped and re-written from scratch. It had tons of lines of plots I already stripped out. The idea here was to be able to tweak the network model to change how much the hsd changed per gate location (i.e., number of upstream gates) or per behavior type (chaotic versus stable). The changes per gate location are called “alpha” and the changes per behavior type are called “beta”

The code below was written to posit multiple scenarios at once, because my original programmer wanted to just run it once and have all the results. But that makes it more challenging for me to try to edit it / run it. So what I would like to do here is have a section where I can change those two aspects once, and then run it for different scenarios. I don’t need to be able to run multiple scenarios at the same time, the man hours to run this code are not that big.

```
# I kept the calculations in this section, but deleted the plots. The idea here
# was to add one more dimension to the 4 scenarios above, by changing the way
# error propegates down the canal. This wasn't leading to results that meant
# anything, so when we get to this step, we may end up changing the methods.
# That's part of why I stripped the plots. So this code may all be tossed in
# favor of something new.
```

```r
# These functions don't return anything, so without generating plots they really
# do nothing so far as I can tell.

network_disaggregated_flow2 <- function(data, casenum, scenario, alpha1, alpha2,
    beta1, beta2) {
    # first let's put the results in a dataframe to calculate the percentage of error
    # at each gate
    gcid_results <- data.frame(do.call(rbind, unlist(data, recursive = FALSE, use.names = TRUE)))
    gcid_results$LATNUM <- rep(1:maxlaterals, times = 1, each = maxgates * nsim)  # for plotting on the
    gcid_results$GATENUM <- rep(1:maxgates, times = maxlaterals, each = nsim)  # for plotting on the y
    gcid_results$LATID <- colnames(network_gatetype_gcid)[gcid_results$LATNUM]
    gcid_results$GATEID <- rownames(network_gatetype_gcid)[gcid_results$GATENUM]
    gcid_results$FLOWERRORP <- gcid_results$FLOWERROR/gcid_results$FLOWTRUTH

    # aggregate the mean error percentage
    gcid_mean_flowerrorp <- aggregate(FLOWERRORP ~ LATNUM + LATID + GATENUM + GATEID,
        data = gcid_results, FUN = mean, na.rm = TRUE)
    colnames(gcid_mean_flowerrorp)[5] <- "MEANFLOWERRORPERCENT"

    lateralflow <- merge(lateralflow, gcid_mean_flowerrorp, by = "LATID")
    lateralflow$FLOWERROR_PER_GATE <- lateralflow$FLOW_PER_GATE * lateralflow$MEANFLOWERRORPERCENT
    write.csv(lateralflow, paste0("outputs/network_alpha_beta_runs/lateral_flow_disaggregate_uniformly_
        casenum, "S", scenario, "A1", alpha1, "_A2", alpha2, "_B1", beta1, "_B2",
        beta2, ".csv"))
    return(lateralflow)
}

network_postprocessing2 <- function(data, casenum, scenario, alpha1, alpha2, beta1,
    beta2) {
    # first let's put the data calculated above in a dataframe
    gcid_results <- data.frame(do.call(rbind, unlist(data, recursive = FALSE, use.names = TRUE)))
    gcid_results$LATNUM <- rep(1:maxlaterals, times = 1, each = maxgates * nsim)  # for plotting on the
    gcid_results$GATENUM <- rep(1:maxgates, times = maxlaterals, each = nsim)  # for plotting on the y
    gcid_results$LATID <- colnames(network_gatetype_gcid)[gcid_results$LATNUM]
    gcid_results$GATEID <- rownames(network_gatetype_gcid)[gcid_results$GATENUM]

    # when calculating mean error the positives and negatives cancel out, so use
    # absolute error instead. Same with min and max, more meaningful to disregard the
    # error sign.
    gcid_mean_flowerror <- aggregate(FLOWERROR ~ LATNUM + LATID + GATENUM + GATEID,
        data = gcid_results, FUN = mean, na.rm = TRUE)
    colnames(gcid_mean_flowerror)[5] <- "MEANFLOWERROR"
    gcid_abs_mean_flowerror <- aggregate(abs(FLOWERROR) ~ LATNUM + LATID + GATENUM +
        GATEID, data = gcid_results, FUN = mean, na.rm = TRUE)
    colnames(gcid_abs_mean_flowerror)[5] <- "MEANABSFLOWERROR"
    gcid_min_flowerror <- aggregate(abs(FLOWERROR) ~ LATNUM + LATID + GATENUM + GATEID,
        data = gcid_results, FUN = min, na.rm = TRUE)
    colnames(gcid_min_flowerror)[5] <- "MINABSFLOWERROR"
    gcid_max_flowerror <- aggregate(abs(FLOWERROR) ~ LATNUM + LATID + GATENUM + GATEID,
        data = gcid_results, FUN = max, na.rm = TRUE)
    colnames(gcid_max_flowerror)[5] <- "MAXABSFLOWERROR"

    # probability that the error condition is met
```

```
    gcid_abs_mean_errorcondition <- aggregate(ERRORCONDITION ~ LATNUM + LATID + GATENUM +
        GATEID, data = gcid_results, FUN = mean, na.rm = TRUE)
    gcid_abs_mean_errorcondition$ERRORCONDITIONNOTMET <- 1 - gcid_abs_mean_errorcondition$ERRORCONDITIO
}

# The two lines of code below came at the end of the plots, not sure if it should
# be deleted with the plots or not network_hsd_gcid_case1 <-
# network_hsd_gcid_case2 <- network_gatetype_gcid network_hsd_gcid_case1[,] <-
# network_hsd_gcid_case2[,] <- NA


networkoperationalerrorsim <- function(alpha1 = 0.01, alpha2 = 0.01, beta1 = 1, beta2 = 1) {
    for (i in 1:maxlaterals) {
        for (j in 1:maxgates) {
            # assuming these errors are independent of one another, therefore sum the squares
            # and take a square root
            network_hsd_gcid_case1[j, i] <- ((alpha1 * network_numgates_gcid[j, i])^2 +
                (sd(latlvl[latlvl$LATID == "48-1", "VALUE"])/mean(latlvl[latlvl$LATID ==
                  "48-1", "VALUE"]) * beta1)^2)^0.5
            network_hsd_gcid_case2[j, i] <- ((alpha2 * network_numgates_gcid[j, i])^2 +
                (sd(latlvl[latlvl$LATID == "22-1", "VALUE"])/mean(latlvl[latlvl$LATID ==
                  "22-1", "VALUE"]) * beta2)^2)^0.5
        }
    }
}

# The below line would wipe data that took along time to generate before in the
# data frames 'networkresult_gcid_case' networkresult_gcid_case1A <-
# networkresult_gcid_case2A <- networkresult_gcid_case1B <-
# networkresult_gcid_case2B <- replicate(n=maxlaterals, expr=list())
for (l in 1:maxlaterals) {
    for (g in 1:maxgates) {
        di <- network_gatetype_gcid[g, l]
        # CASE I (well-behaved lateral), SCENARIO A (accurate)
        hsdn <- network_hsd_gcid_case1[g, l]
        networkresult_gcid_case1A[[l]][[g]] <- operationalerrorsim(dataindex = di,
            coefofvar = 0.02, optflowerrorp = 0.06, gsd = 0.1, hsd = hsdn, tsd = 0.25)
        # CASE II (not well-behaved lateral), SCENARIO A (accurate)
        hsdn <- network_hsd_gcid_case2[g, l]
        networkresult_gcid_case2A[[l]][[g]] <- operationalerrorsim(dataindex = di,
            coefofvar = 0.02, optflowerrorp = 0.06, gsd = 0.1, hsd = hsdn, tsd = 0.25)
        # CASE I (well-behaved lateral), SCENARIO B (less accurate)
        hsdn <- network_hsd_gcid_case1[g, l]
        networkresult_gcid_case1B[[l]][[g]] <- operationalerrorsim(dataindex = di,
            coefofvar = 0.05, optflowerrorp = 0.1, gsd = 0.1, hsd = hsdn, tsd = 0.25)
        # CASE II (not well-behaved lateral), SCENARIO B (less accurate)
        hsdn <- network_hsd_gcid_case2[g, l]
        networkresult_gcid_case2B[[l]][[g]] <- operationalerrorsim(dataindex = di,
            coefofvar = 0.05, optflowerrorp = 0.1, gsd = 0.1, hsd = hsdn, tsd = 0.25)
    }
}

# These lines seem to just be aboout generating graphs, which no longer happens
```

```
# in the functions.
network_disaggregated_flow2(networkresult_gcid_case1A, "1", "A", alpha1, alpha2,
    beta1, beta2)
network_disaggregated_flow2(networkresult_gcid_case2A, "2", "A", alpha1, alpha2,
    beta1, beta2)
network_disaggregated_flow2(networkresult_gcid_case1B, "1", "B", alpha1, alpha2,
    beta1, beta2)
network_disaggregated_flow2(networkresult_gcid_case2B, "2", "B", alpha1, alpha2,
    beta1, beta2)

network_postprocessing2(networkresult_gcid_case1A, "1", "A", alpha1, alpha2, beta1,
    beta2)
network_postprocessing2(networkresult_gcid_case2A, "2", "A", alpha1, alpha2, beta1,
    beta2)
network_postprocessing2(networkresult_gcid_case1B, "1", "B", alpha1, alpha2, beta1,
    beta2)
network_postprocessing2(networkresult_gcid_case2B, "2", "B", alpha1, alpha2, beta1,
    beta2)
```