gate_regression

Nadya Alexander

September 10th, 2019

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)
```

1.0 Data

1.1 Data Gathering

```
# put all the flow data from the different type of gates here
flowdf <- list()

# check.names=FALSE removes leadinging X's from column names
# this used to be "inputs/armco_gate36.csv", for now overwrite it and use the 18" gate data from Nadya
flowdf[[1]] <- read.csv("inputs/armco_gate18.csv", header = TRUE, check.names = FALSE, fileEncoding="UTF
flowdf[[2]] <- read.csv("inputs/armco_gate18.csv", header = TRUE, check.names = FALSE, fileEncoding="UTF
flowdf[[3]] <- read.csv("inputs/armco_gate18.csv", header = TRUE, check.names = FALSE, fileEncoding="UTF
flowdf[[4]] <- read.csv("inputs/armco_gate18.csv", header = TRUE, check.names = FALSE, fileEncoding="UTF
ngates <- length(flowdf)
```

1.2 Data Preprocessing

```
library(reshape2)
preprocessing <- function(data){
  # deleting last two NA rows from stubborn Excel
  data <- na.omit(data)
  # reshape the data from wide to long format
  datalong <- melt(data, id.vars = "H", measure.var = 2:ncol(data), var.name = colnames(data), value.nam
```

```
  datalong$LOGFLOW <- log(datalong$FLOW)
  datalong$G <- as.numeric(datalong$G)
  datalong
}

# now we can easily apply the function we wrote to all elements of the list with lapply
flowdfl <- lapply(flowdf, preprocessing)

# to take a look do this
head(flowdfl[[ngates]])
tail(flowdfl[[ngates]])
```

2.0 Model Simulations

models will come in 8 forms, complex to simple. data will come in 4 forms, for each gate type. (subject to change) Therefore, we will have 8*4=32 models. (subject to change)

```
library(nls2)

# note these models were only written for one flow dataframe: flowdf[[1]]
# attempt 1: model form is
# Q = (aG^2 + bG + c)*H^(dG^2 + eG + f)
# log[Q] = log[(aG^2 + bG + c)*H^(dG + eG + f)]
# log[Q] = log[(aG^2 + bG + c)] +(dG + eG + f)*log[H]
modelsim <- nls(LOGFLOW~log(A*G^2+B*G+C)+(D*G^2+E*G+FF)*log(H), data=flowdfl[[1]], start=list(A=0, B=0.5

# attempt 2: model form is
# Q = (bG + c)*H^(dG^2 + eG + f)
# log[Q] = log[(bG + c)*H^(dG + eG + f)]
# log[Q] = log[(bG + c)] +(dG + eG + f)*log[H]
modelsim <- nls(LOGFLOW~log(B*G+C)+(D*G^2+E*G+FF)*log(H), data=flowdfl[[1]], start=list(B=1, C=1, D=1, E

# attempt 3: model form is
# Q = (aG^2 + bG + c)*H^(eG + f)
# log[Q] = log[(aG^2 + bG + c)*H^(eG + f)]
# log[Q] = log[(aG^2 + bG + c)] + (eG + f)*log[H]
modelsim <- nls(LOGFLOW~log(A*G^2+B*G+C)+(E*G+FF)*log(H), data=flowdfl[[1]], start=list(A=0, B=0.5, C=2

# attempt 4: model form is
# Q = (c)*H^(dG^2 + eG + f)
# log[Q] = log[(c)*H^(dG^2 + eG + f)]
# log[Q] = log[(c)] +(dG^2 + eG + f)*log[H]
modelsim <- nls(LOGFLOW~log(C)+(D*G^2+E*G+FF)*log(H), data=flowdfl[[1]], start=list(C=1, D=1, E=1, FF=1

# attempt 5: model form is
# Q = (aG^2 + bG + c)*H^(f)
# log[Q] = log[(aG^2 + bG + c)*H^(f)]
# log[Q] = log[(aG^2 + bG + c)] +(f)*log[H]
modelsim <- nls(LOGFLOW~log(A*G^2+B*G+C)+(FF)*log(H), data=flowdfl[[1]], start=list(A=0, B=0.5, C=2.5, F

# attempt 6: model form is
# Q = (c)*H^(eG + f)
# log[Q] = log[(c)*H^(eG + f)]
```

```
# log[Q] = log[(c)] +(eG + f)*log[H]
modelsim <- nls(LOGFLOW~log(C)+(E*G+FF)*log(H), data=flowdfl[[1]], start=list(C=1, E=1, FF=1))

# attempt 7: model form is
# Q = (bG + c)*H^(f)
# log[Q] = log[(bG + c)*H^(f)]
# log[Q] = log[(bG + c)] +(f)*log[H]
modelsim <- nls(LOGFLOW~log(B*G+C)+(FF)*log(H), data=flowdfl[[1]], start=list(B=1, C=1, FF=1))

# attempt 8: model form is
# Q = (G)*H^(f)
# log[Q] = log[(G)*H^(f)]
# log[Q] = log[(G)] + (f)*log[H]
modelsim <- nls(LOGFLOW~log(G)+(FF)*log(H), data=flowdfl[[1]], start=list(FF=1))

# run this to see the optimized parameters in the description
modelsim
```

```
# check these with Nadya

# model formulas
formulastring <- list()
formulastring[[1]] <- "LOGFLOW ~log(A*G^2+B*G+C)+(D*G^2+E*G+FF)*log(H)"
formulastring[[2]] <- "LOGFLOW ~log(B*G+C)+(D*G^2+E*G+FF)*log(H)"
formulastring[[3]] <- "LOGFLOW ~log(A*G^2+B*G+C)+(E*G+FF)*log(H)"
formulastring[[4]] <- "LOGFLOW ~log(C)+(D*G^2+E*G+FF)*log(H)"
formulastring[[5]] <- "LOGFLOW ~log(A*G^2+B*G+C)+(FF)*log(H)"
formulastring[[6]] <- "LOGFLOW ~log(C)+(E*G+FF)*log(H)"
formulastring[[7]] <- "LOGFLOW ~log(B*G+C)+(FF)*log(H)"
formulastring[[8]] <- "LOGFLOW ~log(G)+(FF)*log(H)"
nformulas <- length(formulastring)

# term starts:    attempt No.1    2   3    4   5    6   7   8
termstart <- data.frame( A=c(0,   NA, 0,   NA, 0,   NA, NA, NA),
                         B=c(0.5, 1,  0.5, NA, 0.5, NA, 1,  NA),
                         C=c(2,   1,  2.5, 1,  2.5, 1,  1,  NA),
                         D=c(0,   1,  NA,  1,  NA,  NA, NA, NA),
                         E=c(0,   1,  0,   1,  NA,  1,  NA, NA),
                        FF=c(0.5, 1,  0.5, 1,  0.5, 1,  1,  1))
```

```
modelsim <- function(data, formulastring, termstart){
  nls(as.formula(formulastring), data=data, start=termstart)
}

# d for the number of flow dataframes
# m for the number of model forms
modelfits <- replicate(n=4, expr=list())
for(dd in 1:ngates){
  for (m in 1:nformulas){
    termstartsub <- list(A=termstart[m, 1], B=termstart[m, 2], C=termstart[m, 3], D=termstart[m, 4], E=
    termstartsub <- termstartsub[!is.na(termstartsub)]
    modelfits[[dd]][[m]] <- modelsim(flowdfl[[dd]], formulastring[[m]], termstartsub)
    # print(paste("d:", d))
    # print(paste("m:", m))
```

```
  }
}
```

3.0 Model Stats

```
# library(hydroGOF) # this is giving wrong functions, do not load it in make sure the search path is cl
goffuncs <- list.files("libraries/HydroGOFm/R")
for(i in 1:length(goffuncs)){
  source(paste0("libraries/HydroGOFm/R/", goffuncs[i]))
}
remove(goffuncs)

search()
```

```
modelcoef <- replicate(n=ngates, expr=list())
modelpred <- replicate(n=ngates, expr=list())
modelstats <- replicate(n=ngates, expr=list())

for(dd in 1:ngates){
  for (m in 1:nformulas){
    # get the coefficients
    modelcoef[[dd]][[m]] <- coef(modelfits[[dd]][[m]])

    # use the fits to predict
    modelpred[[dd]][[m]] <- predict(modelfits[[dd]][[m]], data=flowdfl[[dd]], type="response")

    # use the hydroGOF package to calculate model measures of fit
    modelstats[[dd]][[m]] <- gof(modelpred[[dd]][[m]], flowdfl[[dd]]$LOGFLOW)

    # # no need to do all of this now because the stats package does it all for us
    # # fit an lm to pred vs. obs to calculate b
    # lmmod <- lm(modelpred[[dd]][[m]]~flowdfl[[dd]]$LOGFLOW)
    #
    # # calculate bR2 or bias-corrected R^2
    # # For slope greater than one in the predicted vs. observed graph (meaning you are overpredicting)
    # if(lmmod$coefficients[2]>1){
    #   lmmod <- lm(flowdfl[[dd]]$LOGFLOW~modelpred[[dd]][[m]])
    # }
    #
    # modelstats_b[[dd]][[m]] <- lmmod$coefficients[2]
    # modelstats_r2[[dd]][[m]] <- r2(modelpred[[dd]][[m]], flowdfl[[dd]]$LOGFLOW)
    # modelstats_br2[[dd]][[m]] <- modelstats_b[[dd]][[m]]*modelstats_r2[[dd]][[m]]
  }
}
```

```
# model observations and predictions
modelcsv <- do.call("rbind", flowdfl)
modelcsv$NGATE <- c(rep(1, nrow(flowdfl[[1]])), rep(2, nrow(flowdfl[[2]])), rep(3, nrow(flowdfl[[3]])),

modelpredm <- do.call("rbind", modelpred)
modelcsv$MODEL1 <- c(unlist(modelpredm[1,1]), unlist(modelpredm[2,1]), unlist(modelpredm[3,1]), unlist(m
modelcsv$MODEL2 <- c(unlist(modelpredm[1,2]), unlist(modelpredm[2,2]), unlist(modelpredm[3,2]), unlist(m
modelcsv$MODEL3 <- c(unlist(modelpredm[1,3]), unlist(modelpredm[2,3]), unlist(modelpredm[3,3]), unlist(m
```

```
modelcsv$MODEL4 <- c(unlist(modelpredm[1,4]), unlist(modelpredm[2,4]), unlist(modelpredm[3,4]), unlist(m
modelcsv$MODEL5 <- c(unlist(modelpredm[1,5]), unlist(modelpredm[2,5]), unlist(modelpredm[3,5]), unlist(m
modelcsv$MODEL6 <- c(unlist(modelpredm[1,6]), unlist(modelpredm[2,6]), unlist(modelpredm[3,6]), unlist(m
modelcsv$MODEL7 <- c(unlist(modelpredm[1,7]), unlist(modelpredm[2,7]), unlist(modelpredm[3,7]), unlist(m
modelcsv$MODEL8 <- c(unlist(modelpredm[1,8]), unlist(modelpredm[2,8]), unlist(modelpredm[3,8]), unlist(m

write.csv(modelcsv, "outputs/gate_equation_modeling.csv", row.names=FALSE)
```

```
# model coefficients and measures of fit
modelcsv <- data.frame(matrix(nrow=ngates*nformulas, ncol=0))
modelcsv$NGATE <- c(rep(1,8), rep(2,8), rep(3,8), rep(4,8))
modelcsv$NMODEL <- rep(c(1:8), 4)

i <- 1
for(dd in 1:ngates){
  for (m in 1:nformulas){
    modelcsv$COEF_A[i] <- modelcoef[[dd]][[m]]["A"]
    modelcsv$COEF_B[i] <- modelcoef[[dd]][[m]]["B"]
    modelcsv$COEF_C[i] <- modelcoef[[dd]][[m]]["C"]
    modelcsv$COEF_D[i] <- modelcoef[[dd]][[m]]["D"]
    modelcsv$COEF_E[i] <- modelcoef[[dd]][[m]]["E"]
    modelcsv$COEF_FF[i] <- modelcoef[[dd]][[m]]["FF"]
    modelcsv$ME[i] <- t(modelstats[[dd]][[m]])[1]
    modelcsv$MAE[i] <- t(modelstats[[dd]][[m]])[2]
    modelcsv$MSE[i] <- t(modelstats[[dd]][[m]])[3]
    modelcsv$RMSE[i] <- t(modelstats[[dd]][[m]])[4]
    modelcsv$NRMSE[i] <- t(modelstats[[dd]][[m]])[5]
    modelcsv$PBIAS[i] <- t(modelstats[[dd]][[m]])[6]
    modelcsv$RSR[i] <- t(modelstats[[dd]][[m]])[7]
    modelcsv$rSD[i] <- t(modelstats[[dd]][[m]])[8]
    modelcsv$NSE[i] <- t(modelstats[[dd]][[m]])[9]
    modelcsv$mNSE[i] <- t(modelstats[[dd]][[m]])[10]
    modelcsv$rNSE[i] <- t(modelstats[[dd]][[m]])[11]
    modelcsv$d[i] <- t(modelstats[[dd]][[m]])[12]
    modelcsv$md[i] <- t(modelstats[[dd]][[m]])[13]
    modelcsv$rd[i] <- t(modelstats[[dd]][[m]])[14]
    modelcsv$cp[i] <- t(modelstats[[dd]][[m]])[15]
    modelcsv$r[i] <- t(modelstats[[dd]][[m]])[16]
    modelcsv$R2[i] <- t(modelstats[[dd]][[m]])[17]
    modelcsv$bR2[i] <- t(modelstats[[dd]][[m]])[18]
    modelcsv$KGE[i] <- t(modelstats[[dd]][[m]])[19]
    modelcsv$VE[i] <- t(modelstats[[dd]][[m]])[20]
    i <- i+1
  }
}
write.csv(modelcsv, "outputs/gate_equation_modeling2.csv", row.names=FALSE)
```

4.0 Model Plots

```
# paste the equation for flow (use formula string) inside the plots, so we know what equation it was mo
for(dd in 1:ngates){
  for (m in 1:nformulas){
```

```
    png(paste0("outputs/obspreplots/ovp_d", dd, "_m", m, ".png"), width=3.25, height=2.85, units="in", p
      plot(flowdfl[[dd]]$LOGFLOW, modelpred[[dd]][[m]], ylab="Observed Log Flow", xlab="Predicted Log Fl

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

      # add line for linear fit
      lmmod <- lm(modelpred[[dd]][[m]]~flowdfl[[dd]]$LOGFLOW)
      abline(lmmod, col="red")
      legend("bottomright", horiz=FALSE, inset=c(0.01, 0.01), cex=0.6, c("Y = X line", "regression line
      mtext(paste("Y =", round(lmmod$coefficients[2],3), "X +", round(lmmod$coefficients[1],0)), side=3
      mtext(paste("Model bR2:", round(t(modelstats[[dd]][[m]])[18],3)), side=3, line=0.3, cex=0.8, adj=0
    dev.off()
  }
}
```

5.0 Uncertainty from Flow Table Equation

In this section, we are searching for the distribution of errors (i.e., the shape of the PDF) necessary in the inputs (i.e., coefficients, G, and H) of the flow equation that produce a certain (i.e., +/-6% or +/-12%) error in the flow. 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 simplicity, we will consider all possible gate and upstream level heights as uniformly possible. Their distributions will be defined by the range of possible values. The variance in Q from the Monte-Carlo simulation is to equal the desired variance (objective function). The decision variables are the parameters of the distributions that describe the PDF of the coefficients.

```
# model #5: Q = (aG^2 + bG + c)*H^(f)
# for data #1, picked model #5
# for data #2, picked model #5 update this with new data!!!!
# for data #3, picked model #5
# for data #4, picked model #5

# the number of Monte-Carlo loops
nsim <- 100

# for +/- 6% error of Q
# 2SD = 0.06 * Q
# SD = 0.03 * Q
# VAR^0.5 = 0.03 * Q
# VAR = 0.0009 * Q^2
set.seed(5152019)

# OLD CODE TBD
# coefofvar <- 0.459781
# optvarflowp <- 9e-4
# montecarlosim <- function(coefofvar, dataindex){
#   d <- dataindex
#
#   # we can add the ability to call different functions here, but for now just equation #5
#   atrue <- modelcoef[[dd]][[5]]["A"] # assume the mean is the true value
```

```
#   astdv <- coefofvar*modelcoef[[dd]][[5]]["A"]
#   anorm <- rnorm(nsim, modelcoef[[dd]][[5]]["A"], astdv) # samples around true value, creating error
#
#   btrue <- modelcoef[[dd]][[5]]["B"]
#   bstdv <- coefofvar*modelcoef[[dd]][[5]]["B"]
#   bnorm <- rnorm(nsim, modelcoef[[dd]][[5]]["B"], bstdv)
#
#   ctrue <- modelcoef[[dd]][[5]]["C"]
#   cstdv <- coefofvar*modelcoef[[dd]][[5]]["C"]
#   cnorm <- rnorm(nsim, modelcoef[[dd]][[5]]["C"], cstdv)
#
#   fftrue <- modelcoef[[dd]][[5]]["FF"]
#   ffstdv <- coefofvar*modelcoef[[dd]][[5]]["FF"]
#   ffnorm <- rnorm(nsim, modelcoef[[dd]][[5]]["FF"], ffstdv)
#
#   # note that G and H are certain here, the only source of uncertainty is in the model parameters! We
#   gunif <- runif(nsim, min(flowdfl[[dd]]$G), max(flowdfl[[dd]]$G))
#   hunif <- runif(nsim, min(flowdfl[[dd]]$H), max(flowdfl[[dd]]$H))
#
#   # to view them you can plot the histograms
#   # hist(gunif)
#
#   flowsim <- flowtruth <- vector()
#   for(i in 1:nsim){
#     # we can add the ability to call different functions here, but for now just equation #5
#     flowsim[i] <- (anorm[i]*gunif[i]^2 + bnorm[i]*gunif[i] + cnorm[i])*(hunif[i]^ffnorm[i])
#     flowtruth[i] <- (atrue*gunif[i]^2 + btrue*gunif[i] + ctrue)*(hunif[i]^fftrue)
#
#     # can do the same calcs in the log transformed version of the equations, but not going to make a
#     # LOGFLOW ~log(A*G^2+B*G+C)+(FF)*log(H)
#     # logflowsim[i] <- log(anorm[i]*gunif[i]^2 + bnorm[i]*gunif[i] + cnorm[i]) + ffnorm[i]*log(hunif[
#   }
#
#   flowerror <- flowtruth-flowsim
#   meanflowerror <- mean(flowerror)
#   stdvflowerror <- sd(flowerror)
#   varflowerror <- sd(flowerror)^2
#
#   # optvarflow <- optvarflowp*mean(flowsim)^2 # mean(flowdfl[[dd]]$FLOW) should be mean(flowsim), do
#   # resultstomin <- varflow - optvarflow
#
#   # a hard condition for optimization: all errors in flow have to be less than the maximum error allo
#   optflow <- optvarflowp*flowtruth
#   errorconditioncol <- ifelse(flowerror<optflow, 1, 0)
#   # resultstomin <- nsim-sum(errorconditioncol)
#   df <- cbind(FLOWTRUTH=flowtruth, FLOWSIM=flowsim, FLOWERROR=flowerror, OPTFLOW=optflow, ERRORCONDIT
# }

# # why are the results so unstable, and so far from the optimum? and why are there warnings...
# optresult <- replicate(n=ngates, expr=list())
# for(dd in 1:ngates){
#   optresult[[dd]] <- optim(par=c(1), fn=montecarlosim, dataindex=d, method="L-BFGS-B", lower=0, upper
# }
```

```
#
# library(optimx)
# optresult <- replicate(n=ngates, expr=list())
# for(dd in 1:ngates){
#   optresult[[dd]] <- optimx(par=c(1), fn=montecarlosim, dataindex=d, method="nlm", control=list(trace
# }

# the function is too nonlinear for this optimization (?)

# OLD CODE TBD
# for(dd in 1:ngates){
#   coefofvar <- 0.01 # very little uncertainty in the equation
#   astdv <- coefofvar*modelcoef[[dd]][[5]]["A"]
#   anorm <- rnorm(nsim, modelcoef[[dd]][[5]]["A"], astdv)
#   bstdv <- coefofvar*modelcoef[[dd]][[5]]["B"]
#   bnorm <- rnorm(nsim, modelcoef[[dd]][[5]]["B"], bstdv)
#   cstdv <- coefofvar*modelcoef[[dd]][[5]]["C"]
#   cnorm <- rnorm(nsim, modelcoef[[dd]][[5]]["C"], cstdv)
#   ffstdv <- coefofvar*modelcoef[[dd]][[5]]["FF"]
#   ffnorm <- rnorm(nsim, modelcoef[[dd]][[5]]["FF"], ffstdv)
#   # set the G, and H at a specific value, or itterate over operational G and H, either way if you are
#   gnorm <- rnorm(nsim, 15,  0.01)
#   hnorm <- rnorm(nsim, 0.79, 0.5)
#
#   # uncertainty in operational flow
#   for(i in 1:nsim){
#     opflowsim[i,d] <- (anorm[i]*gnorm[i]^2 + bnorm[i]*gnorm[i] + cnorm[i])*(hnorm[i]^ffnorm[i])
#   }
#   opvarflow[[dd]] <- sd(opflowsim[,d], na.rm=TRUE)^2
# }
```

```
nsim <- 100
set.seed(5152019)

# optflowerrorp is the percent of error allowed, like 6%, 12%
equationerrorsim <- function(dataindex, coefofvar, optflowerrorp){
  dd <- dataindex
  # we can add the ability to call different functions here, but for now just equation #5
  atrue <- modelcoef[[dd]][[5]]["A"] # assume the mean is the true value
  astdv <- coefofvar*modelcoef[[dd]][[5]]["A"]
  anorm <- rnorm(nsim, modelcoef[[dd]][[5]]["A"], astdv) # samples around true value, creating error
  btrue <- modelcoef[[dd]][[5]]["B"]
  bstdv <- coefofvar*modelcoef[[dd]][[5]]["B"]
  bnorm <- rnorm(nsim, modelcoef[[dd]][[5]]["B"], bstdv)
  ctrue <- modelcoef[[dd]][[5]]["C"]
  cstdv <- coefofvar*modelcoef[[dd]][[5]]["C"]
  cnorm <- rnorm(nsim, modelcoef[[dd]][[5]]["C"], cstdv)
  fftrue <- modelcoef[[dd]][[5]]["FF"]
  ffstdv <- coefofvar*modelcoef[[dd]][[5]]["FF"]
  ffnorm <- rnorm(nsim, modelcoef[[dd]][[5]]["FF"], ffstdv)

  # note that G and H are certain here, the only source of uncertainty is in the model parameters! We a
  gunif <- runif(nsim, min(flowdfl[[dd]]$G), max(flowdfl[[dd]]$G))
  hunif <- runif(nsim, min(flowdfl[[dd]]$H), max(flowdfl[[dd]]$H))
```

```
  flowsim <- flowtruth <- vector()
  for(i in 1:nsim){
    # we can add the ability to call different functions here, but for now just equation #5
    flowsim[i] <- (anorm[i]*gunif[i]^2 + bnorm[i]*gunif[i] + cnorm[i])*(hunif[i]^ffnorm[i])
    flowtruth[i] <- (atrue*gunif[i]^2 + btrue*gunif[i] + ctrue)*(hunif[i]^fftrue)
  }
  flowerror <- flowtruth-flowsim

  # a hard condition for optimization: all errors in flow have to be less than the maximum error allowe
  optflowerror <- optflowerrorp*flowtruth
  errorconditioncol <- ifelse(abs(flowerror)<optflowerror, 1, 0)
  # resultstomin <- nsim-sum(errorconditioncol)
  resultsdf <- cbind(FLOWTRUTH=flowtruth, FLOWSIM=flowsim, FLOWERROR=flowerror, OPTFLOW=optflowerror, E
}

cvrange <- seq(0.01, 0.5, 0.01)
optresult <- replicate(n=ngates, expr=list())
for(dd in 1:ngates){
  for(c in 1:length(cvrange)){
    cv <- cvrange[c]
    optresult[[dd]][[c]] <- equationerrorsim(dataindex=dd, coefofvar=cv, optflowerrorp=0.06)
  }
}
```

```
for(c in 1:length(cvrange)){
  png(paste0('outputs/equation_error/optimization_cloud_', c, ".png"), width=3.25, height=3, units="in"
    par(mar=c(4,4,2,1)+0.1)
    plot(optresult[[1]][[c]][, "FLOWERROR"], optresult[[1]][[c]][, "FLOWTRUTH"], col=c("red", "steelblu
    points(optresult[[2]][[c]][, "FLOWERROR"], optresult[[2]][[c]][, "FLOWTRUTH"], col=c("red", "steelb
    points(optresult[[3]][[c]][, "FLOWERROR"], optresult[[3]][[c]][, "FLOWTRUTH"], col=c("red", "steelb
    points(optresult[[4]][[c]][, "FLOWERROR"], optresult[[4]][[c]][, "FLOWTRUTH"], col=c("red", "steelb
    mtext(paste0("Coefficient of Variation: ", cvrange[c]), side=3, line=0)
    legend("topleft", c("Gate 1", "Gate 2", "Gate 3", "Gate 4"), pch=c(19,19,19,19), col=c("steelblue1"
  dev.off()
}

png('outputs/optfn_equation_error.png', width=3.25, height=3, units="in", pointsize=8, res=1200)
  par(mar=c(4,4.5,1,1)+0.1)
  plot(cvrange, nsim-do.call(rbind,lapply(optresult[[1]], colSums))[, "ERRORCONDITION"], xlab="Coefficie
  points(cvrange, nsim-do.call(rbind,lapply(optresult[[2]], colSums))[, "ERRORCONDITION"], pch=19, col=
  points(cvrange, nsim-do.call(rbind,lapply(optresult[[3]], colSums))[, "ERRORCONDITION"], pch=19, col=
  points(cvrange, nsim-do.call(rbind,lapply(optresult[[4]], colSums))[, "ERRORCONDITION"], pch=19, col=

  # points(optresult[which.min(optresult$X4), "CV"], min(optresult$X4), pch=19, col="red")
  # text(paste0("min=(", optresult[which.min(optresult$X4), "CV"], ",", round(min(optresult$X4), 0), ")
  legend("topleft", c("Gate 1", "Gate 2", "Gate 3", "Gate 4"), pch=c(19,19,19,19), col=c("steelblue1",
dev.off()
```

6.0 Uncertainty From Operations – Simulations

6.1 Uncertainty in Operational Flow - without consideration for the network

```
# volume = t[Q] = t[(aG^2 + bG + c)*H^(f)]
# gsd, hsd, tsd are the standard deviations of G, H and t respectively
# note that optflowerrorp is the percentage 6 or 10 % error that is permissable. This error is related
operationalerrorsim <- function(dataindex, coefofvar=0.02, optflowerrorp=0.06, gsd=0.1, hsd=0.1, tsd=0.1
  d <- dataindex
  # we can add the ability to call different functions here, but for now just equation #5
  atrue <- modelcoef[[dd]][[5]]["A"] # assume the mean is the true value
  astdv <- coefofvar*modelcoef[[dd]][[5]]["A"]
  anorm <- rnorm(nsim, modelcoef[[dd]][[5]]["A"], astdv) # samples around true value, creating error
  btrue <- modelcoef[[dd]][[5]]["B"]
  bstdv <- coefofvar*modelcoef[[dd]][[5]]["B"]
  bnorm <- rnorm(nsim, modelcoef[[dd]][[5]]["B"], bstdv)
  ctrue <- modelcoef[[dd]][[5]]["C"]
  cstdv <- coefofvar*modelcoef[[dd]][[5]]["C"]
  cnorm <- rnorm(nsim, modelcoef[[dd]][[5]]["C"], cstdv)
  fftrue <- modelcoef[[dd]][[5]]["FF"]
  ffstdv <- coefofvar*modelcoef[[dd]][[5]]["FF"]
  ffnorm <- rnorm(nsim, modelcoef[[dd]][[5]]["FF"], ffstdv)

  # 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
  gunif <- runif(nsim, min(flowdfl[[dd]]$G), max(flowdfl[[dd]]$G))
  hunif <- runif(nsim, min(flowdfl[[dd]]$H), max(flowdfl[[dd]]$H))
  tunif <- runif(nsim, 0, 12) # in hours

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

  volsim <- voltruth <-  flowsim <- flowtruth <- vector()
  for(i in 1:nsim){
    # we can add the ability to call different functions here, but for now just equation #5
    flowsim[i] <- (anorm[i]*gnorm[i]^2 + bnorm[i]*gnorm[i] + cnorm[i])*(hnorm[i]^ffnorm[i])
    flowtruth[i] <- (atrue*gunif[i]^2 + btrue*gunif[i] + ctrue)*(hunif[i]^fftrue)
    volsim[i] <- (anorm[i]*gnorm[i]^2 + bnorm[i]*gnorm[i] + cnorm[i])*(hnorm[i]^ffnorm[i])*tnorm[i]
    voltruth[i] <- (atrue*gunif[i]^2 + btrue*gunif[i] + ctrue)*(hunif[i]^fftrue)*tunif[i]
  }
  flowerror <- flowtruth-flowsim
  optflowerror <- optflowerrorp*flowtruth
  errorconditioncol <- ifelse(abs(flowerror)<optflowerror, 1, 0)
  volerror <- voltruth-volsim

  # consider having t=f(i or location)
  # canal has a capacity, they don't deliver to everyone at the same time (maybe won't matter)
  # dependency, he may pick a point of accuracy, deliver accurately to one and over deliver to someone
  # conceptual exercise
```

```
  resultsdf <- cbind(FLOWTRUTH=flowtruth, FLOWSIM=flowsim, FLOWERROR=flowerror, OPTFLOW=optflowerror, E
}

operationresult <- replicate(n=ngates, expr=list())
for(dd in 1:ngates){
  operationresult[[dd]] <- operationalerrorsim(dataindex=dd, coefofvar=0.02, optflowerrorp=0.06)
}
```

```
png('outputs/operational_error/optimization_cloud.png', width=3.25, height=3, units="in", pointsize=8, 
  par(mar=c(4,4,2,1)+0.1)
  plot(operationresult[[1]][, "FLOWERROR"], operationresult[[1]][, "FLOWTRUTH"], col=c("red", "steelblu
  points(operationresult[[2]][, "FLOWERROR"], operationresult[[2]][, "FLOWTRUTH"], col=c("red", "steelb
  points(operationresult[[3]][, "FLOWERROR"], operationresult[[3]][, "FLOWTRUTH"], col=c("red", "steelb
  points(operationresult[[4]][, "FLOWERROR"], operationresult[[4]][, "FLOWTRUTH"], col=c("red", "steelb
  mtext(paste0("Coefficient of Variation of Equation Parameters: 0.01"), side=3, line=0)
  legend("topleft", c("Gate 1", "Gate 2", "Gate 3", "Gate 4"), pch=c(19,19,19,19), col=c("steelblue1", 
dev.off()

png('outputs/operational_volume_error/optimization_cloud.png', width=3.25, height=3, units="in", points
  par(mar=c(4,4,2,1)+0.1)
  plot(operationresult[[1]][, "VOLERROR"], operationresult[[1]][, "VOLTRUTH"], col=c("red", "steelblue1
  points(operationresult[[2]][, "VOLERROR"], operationresult[[2]][, "VOLTRUTH"], col=c("red", "steelblu
  points(operationresult[[3]][, "VOLERROR"], operationresult[[3]][, "VOLTRUTH"], col=c("red", "steelblu
  points(operationresult[[4]][, "VOLERROR"], operationresult[[4]][, "VOLTRUTH"], col=c("red", "steelblu
  mtext(paste0("Coefficient of Variation of Equation Parameters: 0.01"), side=3, line=0)
  legend("topleft", c("Gate 1", "Gate 2", "Gate 3", "Gate 4"), pch=c(19,19,19,19), col=c("steelblue1", 
dev.off()
```

7.0 Uncertainty at the Gate Level - Gates in Series

7.1 Toy Model

```
# 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,] <- 0.10
network_hsd[4,] <- 0.12
network_hsd[5,] <- 0.14

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

```r
# NOTE: only changing the uncertainty in upstream level based on location in the network
for(l in 1:maxlaterals){
  for(g in 1:maxgates){
    dd <- network_gatetype[g, l]
    hsdn <- network_hsd[g, l]
    networkresult[[l]][[g]] <- operationalerrorsim(dataindex=dd, coefofvar=0.02, optflowerrorp=0.06, gs
  }
}

# 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$LATNUM <- rep(1:maxlaterals, times=1, each=maxgates*nsim) # for plotting on the x axis
toymod_results$GATENUM <- rep(1:maxgates, times=maxlaterals, each=nsim) # for plotting on the y axis
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. Sa
toymod_mean_flowerror <- aggregate(FLOWERROR~LATNUM+LATID+GATENUM+GATEID, data=toymod_results, FUN=mean
colnames(toymod_mean_flowerror)[5] <- "MEANFLOWERROR"
toymod_abs_mean_flowerror <- aggregate(abs(FLOWERROR)~LATNUM+LATID+GATENUM+GATEID, data=toymod_results,
colnames(toymod_abs_mean_flowerror)[5] <- "MEANABSFLOWERROR"
toymod_min_flowerror <- aggregate(abs(FLOWERROR)~LATNUM+LATID+GATENUM+GATEID, data=toymod_results, FUN=
colnames(toymod_min_flowerror)[5] <- "MINABSFLOWERROR"
toymod_max_flowerror <- aggregate(abs(FLOWERROR)~LATNUM+LATID+GATENUM+GATEID, data=toymod_results, FUN=
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_resu
toymod_abs_mean_errorcondition$ERRORCONDITIONNOTMET <- 1-toymod_abs_mean_errorcondition$ERRORCONDITION

# plot min, max, mean, ...
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=MEANABSFLOWERRO
    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()

png('outputs/toymodel_min_flowerror.png', width=6.5, height=4, units="in", pointsize=8, res=300)
  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(cfs
    xlab("Lateral No.") +
    ylab("Gate No.")
dev.off()

png('outputs/toymodel_max_flowerror.png', width=6.5, height=4, units="in", pointsize=8, res=300)
  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(cfs)
    xlab("Lateral No.") +
    ylab("Gate No.")
dev.off()

png('outputs/toymodel_mean_noncompliancepercentage_flowerror.png', width=6.5, height=4, units="in", poi
  ggplot(toymod_abs_mean_errorcondition, aes(x=LATNUM, y=GATENUM, size=ERRORCONDITIONNOTMET, fill=ERRORC
    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=10
    xlab("Lateral No.") +
    ylab("Gate No.")
dev.off()
```

7.2 GCID Network

```
# updated column names
gcid <- read.csv("inputs/gcid_lateral_gates.csv")

maxgates <- 72
maxlaterals <- nrow(gcid)

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

# first let's make 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 branc
  }
}
```

```
# make the uncertainty of the upstream level a function of the number of gates above the gate in the ne
latlvl <- read.csv("inputs/lateral_levels_longformat.csv")

# first let's look at the densities of these laterals
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()

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()

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()
```

```
# construct the cases
# CASE I: Well-behaved lateral, LATID=48-1
mean(latlvl[latlvl$LATID=="48-1", "VALUE"])
sd(latlvl[latlvl$LATID=="48-1", "VALUE"])
max(latlvl[latlvl$LATID=="48-1", "VALUE"]) - min(latlvl[latlvl$LATID=="48-1", "VALUE"])

# CASE II: NOT Well-behaved lateral, LATID=22-1
mean(latlvl[latlvl$LATID=="22-1", "VALUE"])
sd(latlvl[latlvl$LATID=="22-1", "VALUE"])
max(latlvl[latlvl$LATID=="22-1", "VALUE"]) - min(latlvl[latlvl$LATID=="22-1", "VALUE"])

# 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 n
    # WARNING!! NOTE!!! multiplied the normalized sd by ...
    network_hsd_gcid_case1[j,i] <- ((0.01*network_numgates_gcid[j,i])^2 + (sd(latlvl[latlvl$LATID=="48-
    network_hsd_gcid_case2[j,i] <- ((0.01*network_numgates_gcid[j,i])^2 + (sd(latlvl[latlvl$LATID=="22-
  }
}

# construct a run for each case, and each scenario. scenario A is for 6% error (0.02 coefofvar) and sce
networkresult_gcid_case1A <- networkresult_gcid_case2A <- networkresult_gcid_case1B <- networkresult_gc
for(l in 1:maxlaterals){
  for(g in 1:maxgates){
    dd <- 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=dd, coefofvar=0.02, optflowerr
    # CASE II (not well-behaved lateral), SCENARIO A (accurate)
    hsdn <- network_hsd_gcid_case2[g, l]
    networkresult_gcid_case2A[[l]][[g]] <- operationalerrorsim(dataindex=dd, coefofvar=0.02, optflowerr
    # CASE I (well-behaved lateral), SCENARIO B (less accurate)
    hsdn <- network_hsd_gcid_case1[g, l]
    networkresult_gcid_case1B[[l]][[g]] <- operationalerrorsim(dataindex=dd, coefofvar=0.05, optflowerr
    # CASE II (not well-behaved lateral), SCENARIO B (less accurate)
    hsdn <- network_hsd_gcid_case2[g, l]
    networkresult_gcid_case2B[[l]][[g]] <- operationalerrorsim(dataindex=dd, coefofvar=0.05, optflowerr
  }
}
```

```r
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$LATNUM <- rep(1:maxlaterals, times=1, each=maxgates*nsim) # for plotting on the x axis
  gcid_results$GATENUM <- rep(1:maxgates, times=maxlaterals, each=nsim) # for plotting on the y axis
  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.
  gcid_mean_flowerror <- aggregate(FLOWERROR~LATNUM+LATID+GATENUM+GATEID, data=gcid_results, FUN=mean,
  colnames(gcid_mean_flowerror)[5] <- "MEANFLOWERROR"
  gcid_abs_mean_flowerror <- aggregate(abs(FLOWERROR)~LATNUM+LATID+GATENUM+GATEID, data=gcid_results, FU
  colnames(gcid_abs_mean_flowerror)[5] <- "MEANABSFLOWERROR"
  gcid_min_flowerror <- aggregate(abs(FLOWERROR)~LATNUM+LATID+GATENUM+GATEID, data=gcid_results, FUN=mi
  colnames(gcid_min_flowerror)[5] <- "MINABSFLOWERROR"
  gcid_max_flowerror <- aggregate(abs(FLOWERROR)~LATNUM+LATID+GATENUM+GATEID, data=gcid_results, FUN=ma
  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_result
  gcid_abs_mean_errorcondition$ERRORCONDITIONNOTMET <- 1-gcid_abs_mean_errorcondition$ERRORCONDITION

  # plot min, max, mean, ...
  gg1 <- ggplot(gcid_abs_mean_flowerror, aes(x=LATNUM, y=GATENUM, size=MEANABSFLOWERROR, fill=MEANABSFL
    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, uni
    plot(gg1)
  dev.off()

  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 \n(cfs
    xlab("Lateral No.") +
    ylab("Gate No.")

  png(paste0('outputs/gcid_min_flowerror_C', casenum, 'S', scenario , '.png'), width=20, height=8, units
    plot(gg2)
  dev.off()

  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 \n(cfs
    xlab("Lateral No.") +
    ylab("Gate No.")
```

```
  png(paste0('outputs/gcid_max_flowerror_C', casenum, 'S', scenario , '.png'), width=20, height=8, units
    plot(gg3)
  dev.off()

  gg4 <- ggplot(gcid_abs_mean_errorcondition, aes(x=LATNUM, y=GATENUM, size=ERRORCONDITIONNOTMET, fill=
    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=1
    xlab("Lateral No.") +
    ylab("Gate No.")

  png(paste0('outputs/gcid_mean_noncompliancepercentage_flowerror_C', casenum, 'S', scenario , '.png'),
    plot(gg4)
  dev.off()
}

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")
```

8.0 Uncertainty at the Lateral Level Disaggregated

8.1 Disaggregated Uniformily

```
lateralflow <- read.csv("inputs/lateral_acre_flow.csv")
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$LATNUM <- rep(1:maxlaterals, times=1, each=maxgates*nsim) # for plotting on the x axis
  gcid_results$GATENUM <- rep(1:maxgates, times=maxlaterals, each=nsim) # for plotting on the y axis
  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
  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
  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")
```

```
# Now that flow through the lateral and into the gates are disaggregated, calcualte the upstream level

# network_hsd_gcid_case1 <- network_hsd_gcid_case2 <- network_gatetype_gcid
# network_hsd_gcid_case1[,] <- NA
#
# library(reshape2)
# network_disaggregated_flow_1A_wide <- dcast(network_disaggregated_flow_1A, GATENUM ~ LATID, value.var
```

This is really bad code, but to do it quickly I am copy-pasting a lot here.

```
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$LATNUM <- rep(1:maxlaterals, times=1, each=maxgates*nsim) # for plotting on the x axis
  gcid_results$GATENUM <- rep(1:maxgates, times=maxlaterals, each=nsim) # for plotting on the y axis
  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
  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_C"
  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$LATNUM <- rep(1:maxlaterals, times=1, each=maxgates*nsim) # for plotting on the x axis
  gcid_results$GATENUM <- rep(1:maxgates, times=maxlaterals, each=nsim) # for plotting on the y axis
  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.
  gcid_mean_flowerror <- aggregate(FLOWERROR~LATNUM+LATID+GATENUM+GATEID, data=gcid_results, FUN=mean,
  colnames(gcid_mean_flowerror)[5] <- "MEANFLOWERROR"
  gcid_abs_mean_flowerror <- aggregate(abs(FLOWERROR)~LATNUM+LATID+GATENUM+GATEID, data=gcid_results, FU
  colnames(gcid_abs_mean_flowerror)[5] <- "MEANABSFLOWERROR"
  gcid_min_flowerror <- aggregate(abs(FLOWERROR)~LATNUM+LATID+GATENUM+GATEID, data=gcid_results, FUN=mi
  colnames(gcid_min_flowerror)[5] <- "MINABSFLOWERROR"
  gcid_max_flowerror <- aggregate(abs(FLOWERROR)~LATNUM+LATID+GATENUM+GATEID, data=gcid_results, FUN=ma
  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_resul
  gcid_abs_mean_errorcondition$ERRORCONDITIONNOTMET <- 1-gcid_abs_mean_errorcondition$ERRORCONDITION

  # plot min, max, mean, ...
  gg1 <- ggplot(gcid_abs_mean_flowerror, aes(x=LATNUM, y=GATENUM, size=MEANABSFLOWERROR, fill=MEANABSFL
    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/network_alpha_beta_runs/gcid_mean_flowerror_C', casenum, 'S', scenario, "A1", alp
    plot(gg1)
  dev.off()

  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 \n(cfs
    xlab("Lateral No.") +
    ylab("Gate No.")

  png(paste0('outputs/network_alpha_beta_runs/gcid_min_flowerror_C', casenum, 'S', scenario, "A1", alpha
    plot(gg2)
  dev.off()

  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 \n(cfs
    xlab("Lateral No.") +
    ylab("Gate No.")

  png(paste0('outputs/network_alpha_beta_runs/gcid_max_flowerror_C', casenum, 'S', scenario, "A1", alpha
    plot(gg3)
  dev.off()

  gg4 <- ggplot(gcid_abs_mean_errorcondition, aes(x=LATNUM, y=GATENUM, size=ERRORCONDITIONNOTMET, fill=
    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=1
    xlab("Lateral No.") +
    ylab("Gate No.")

  png(paste0('outputs/network_alpha_beta_runs/gcid_mean_noncompliancepercentage_flowerror_C', casenum,
    plot(gg4)
  dev.off()
}

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 squa
      network_hsd_gcid_case1[j,i] <- ((alpha1*network_numgates_gcid[j,i])^2 + (sd(latlvl[latlvl$LATID==
      network_hsd_gcid_case2[j,i] <- ((alpha2*network_numgates_gcid[j,i])^2 + (sd(latlvl[latlvl$LATID==
    }
```

```
  }

  networkresult_gcid_case1A <- networkresult_gcid_case2A <- networkresult_gcid_case1B <- networkresult_g
  for(l in 1:maxlaterals){
    for(g in 1:maxgates){
      dd <- 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=dd, coefofvar=0.02, optflowe
      # CASE II (not well-behaved lateral), SCENARIO A (accurate)
      hsdn <- network_hsd_gcid_case2[g, l]
      networkresult_gcid_case2A[[l]][[g]] <- operationalerrorsim(dataindex=dd, coefofvar=0.02, optflowe
      # CASE I (well-behaved lateral), SCENARIO B (less accurate)
      hsdn <- network_hsd_gcid_case1[g, l]
      networkresult_gcid_case1B[[l]][[g]] <- operationalerrorsim(dataindex=dd, coefofvar=0.05, optflowe
      # CASE II (not well-behaved lateral), SCENARIO B (less accurate)
      hsdn <- network_hsd_gcid_case2[g, l]
      networkresult_gcid_case2B[[l]][[g]] <- operationalerrorsim(dataindex=dd, coefofvar=0.05, optflowe
    }
  }

  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)
}

networkoperationalerrorsim(alpha1=0.01, alpha2=0.01, beta1=1, beta2=1)
networkoperationalerrorsim(alpha1=0.01, alpha2=0.01, beta1=1, beta2=2)
networkoperationalerrorsim(alpha1=0.01, alpha2=0.01, beta1=1, beta2=3.5)
networkoperationalerrorsim(alpha1=0.01, alpha2=0.01, beta1=1, beta2=5)
networkoperationalerrorsim(alpha1=0.05, alpha2=0.05, beta1=1, beta2=1)
networkoperationalerrorsim(alpha1=0.05, alpha2=0.05, beta1=1, beta2=2)
networkoperationalerrorsim(alpha1=0.05, alpha2=0.05, beta1=1, beta2=3.5)
networkoperationalerrorsim(alpha1=0.05, alpha2=0.05, beta1=1, beta2=5)
```

8.1 Disaggregated By Service Area