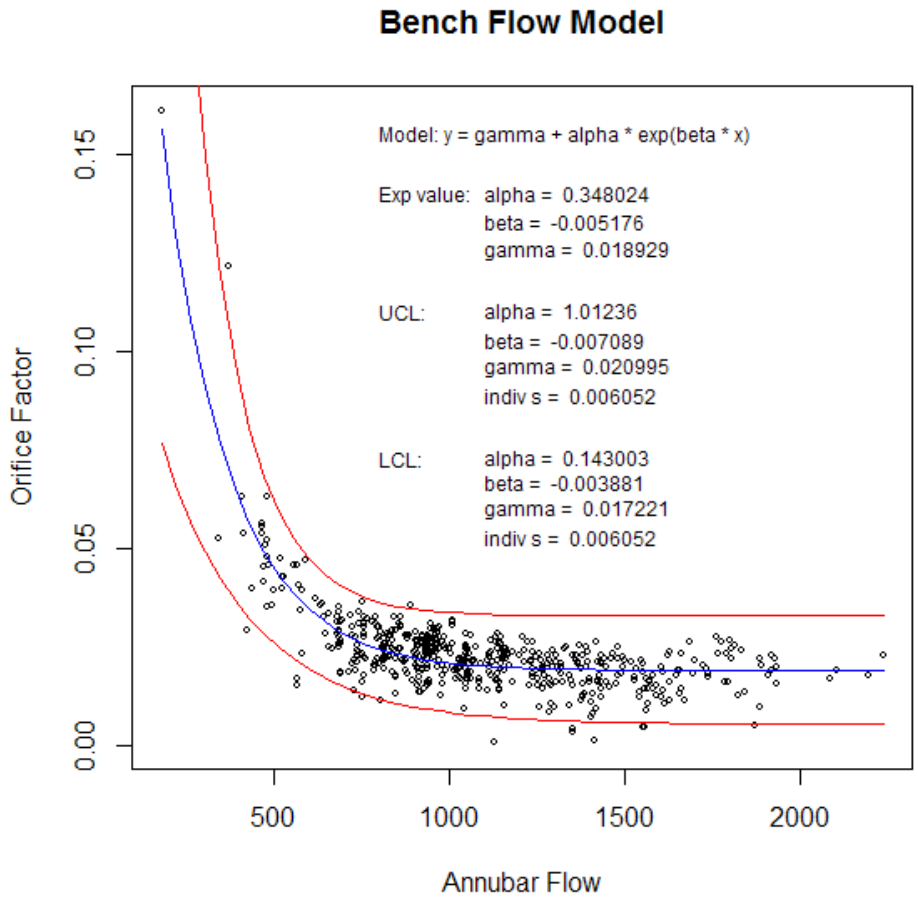
Cormetech, Inc - Bench Flow Model

Tom Balmat, March 2015

The Cormetech Testing Laboratory operates several chemical reactors to evaluate the effectiveness of catalytic reduction of undesirable compounds in various exhaust gases. Control limits were required for a critical parameter, Annubar Flow Orifice Factor, that is monitored during operation of the "Bench" reactor. Historical data were supplied by the responsible lab engineer, to which a three parameter exponential model, $OrificeFactor = \gamma + \alpha e^{\beta AnnubarFlow}$, was fit using non-linear least squares. The simultaneous least squares equations gave optimal estimates of α, β, and γ, but with no indication, empirical or theoretical, of their distributions, so a simulation of 200 α, β, and γ sets was conducted using partitions of the supplied data. From the simulated parameter sets, 10,000 Orifice Factor values were simulated for each observed Annubar Flow level. By partitioning these, a .95 confidence interval on Orifice Factor was defined at each Annubar Flow level, and final exponential distributions were fit to the upper and lower confidence points, giving upper and lower confidence models for the expected value of Orifice Factor. To the expected value models were added (upper) and subtracted (lower) an estimate of individual standard deviation, using deviations of actual Orifice Factor observations to the expected value exponential model fit to the entire data set (the first model mentioned above). Statistical programming and graph construction was accomplished in R. As seen in the plot, the expected value and control limit models track the observations quite well and the number of observations outside of the control limits (confidence band) approaches the number predicted at .95 confidence.



Program Listing

```
# Bench Reactor Annubar Flow Ratio Confidence Limit Model

nlReg <- function(x, y) {

  # Estimate regression parameters for the model
  # y = gamma + alpha*exp(beta*(x+delta))
  #   = gamma + alpha*exp(beta*delta)*exp(beta*x)
  #   = gamma + alpha'*exp(beta*x)
  # where y = Annubar Orifice Factor and x = Annubar Flow

  # Note that the least squares equations for alpha', beta, and gamma
  # are derived by minimizing the sum of squared errors of observed y values
  # to those estimated by the above model at a corresponding value of x

  # This is a non-linear regression problem

  # Source (x,y) pairs taken from input x and y vectors
  # Resulting parameters will be returned in a three element vector
  # alpha in pos 1, beta in pos 2, gamma in pos 3

  # Solve for optimal values of alpha, beta, and gamma (values for alpha, beta, and
  # gamma that minimize the sum of squared errors of actual to model deviations)

  # Sum of squared errors is minimized when the first partial derivatives of the sum,
  # with respect to each of alpha, beta, and gamma is are simultaneously 0

  # Use direct equations to solve optimal solutions for gamma and alpha (since, they can
  # be isolated on one side of their respective equation)

  # Use Newton's Method of Roots to solve for beta

  # Initial values of alpha and gamma are estimates from visual inspection of the x-y plot
  # of values being fit, along with known position, scale, and shape feature/parameter
  # relationships for the exponential pdf

  # alpha incorporates both the original alpha and exp(beta*delta)
  # The initial value for (combined) alpha is based on an estimate of .14 for the original alpha,
  # -0.005 for beta and -170 for delta
  alpha <- .3276
  beta <- -.005
  gamma <- .02

  # Following are the SSE derivative estimates by parameter
  # alpha in pos 1, beta in pos 2, gamma in pos 3
  # Begin with non-zero values to force initial evaluation
  dsse <- c(1,1,1)

  # iterationMax is used to terminate iterative search for intermediate parameter estimate
  # when Newton's Method appears to avoid convergence
  iterationMax <- 100

  # Calculate static values to be used later
  ymean <- mean(y)
  n <- NROW(y)
  nSolutionSets <- 0
  xx <- x*x

  # Continue while any derivative of sse remains "large"
  while(any(abs(dsse)>.000001)) {

    # Calculate optimal beta using current values of alpha and gamma
    nit <- 0
    while(abs(dsse[2])>.000001 & nit<iterationMax) {
      # Populate vector of values using currnt parameter estimates to be used in derivate calcs
      aebx <- alpha*exp(beta*x)
      # Calculate first derivative of the sum of squared errors with respect to beta
      # Note that, the constant factor of 2 (from the squaring of deviations) is omitted from the
      # derivative equation and the right-hand inequality parameter is correspondingly
```

```
      # divided by 2
      # (.000001/2=.0000005)
      # Note the retenton of sign, however, since this is critical to determining the
      # direction of beta adjustment
      dsse[2] <- -sum((y-gamma-aebx)*x*aebx)
      # Calculate the second derivative of the sum of squared errors
      # This determines the direction and amount to to adjust beta to bring the
      # sum of squared error derivative nearer to 0
      d2sse <- -sum((y-gamma-aebx)*xx*aebx - xx*aebx*aebx)
      nit <- nit+1
      if(abs(dsse[2])>.0000005)
        # Adjust along base of right triangle with vertical side parallel to y axis with
        # height equal
        # to distance to adjust y
        # Hypotenuse is at angle corresponding to derivative of (derivative of sum of
        # squared errors)
        # Second derivative positive (derivative of change in SSE) => adjust negative, otherwise
        # adjust positive
        # (objective is to move x in direction of first derivative = 0)
          if(d2sse>0)
            beta <- beta-dsse[2]/d2sse
          else
            beta <- beta+dsse[2]/d2sse
    }

    # Accumulate sums involving x and y once, to avoid repeating in individual
    # parameter equations that follow
    ebx <- exp(beta*x)
    Sebx <- sum(ebx)
    Sebxebx <- sum(ebx*ebx)
    Sxebx <- sum(x*ebx)
    Sxebxebx <- sum(x*ebx*ebx)
    Syebx <- sum(y*ebx)
    Sxyebx <- sum(x*y*ebx)

    # Simultaneously solve for zeroes of both the alpha and gamma derivative equations
    # Note that this causes the beta partial derivative to deviate from 0 (that's why
    # it must be re-evaluated)
    alpha <- n*(Syebx-Sebx*ymean)/(n*Sebxebx-Sebx*Sebx)
    gamma <- ymean-alpha*Sebx/n

    # Asynchronously recalculate alpha based on current values of beta and gamma
    # Note that this causes the beta and gamma partial derivatives to deviate from 0
    # (that's why they must be re-evaluated)
    # alpha <- (Syebx-gamma*Sebx)/Sebxebx
    # Recalculate gamma based on current values of alpha and beta
    # This causes the beta and alpha partial derivatives to deviate from 0
    # gamma <- ymean-alpha*Sebx/n

    # Recalculate all derivatives with new parameter estimates
    # Constant factors are retained for accurate evaluation of proximity to zero
    dsse[1] <- -2*(Syebx-gamma*Sebx-alpha*Sebxebx)
    dsse[2] <- -2*(alpha*Sxyebx-alpha*gamma*Sxebx-alpha*alpha*Sxebxebx)
    dsse[3] <- -2*(n*(ymean-gamma)-alpha*Sebx)

    # Calculate sum of squared errors with new parameter estimates
    sse <- sum((y-gamma-alpha*exp(beta*x))^2)
    nSolutionSets <- nSolutionSets+1

    # Intermediate status output
    # print(c(nSolutionSets, alpha, beta, gamma, sse, dsse))

  }

  par <- c(alpha, beta, gamma)
  names(par) <- c("alpha", "beta", "gamma")
  return(par)

}
```

```
#################################################################################################
# Main program section
# 1. Fit models to randomly selected halves of supplied data
# 2. Simulate Bench Flow response expected value at each recorded predictor level, using
#    randomly selected model parameters from step 1
# 3. Identify empirical (partitioned) expected value .95 upper and lower bounds for each
#    predictor level
# 4. Estimate overall expected value model upper and lower bounds of .95 confidence interval
#    by fitting model to upper and lower partitioned values from step 3
# 5. Estimate individual upper and lower bounds by calculating observed standard deviation
#    (deviations of observed response to model expected value) and adding/subtracting
#    to/from model upper/lower confidence limits
#################################################################################################

# Collect observed Bench annubar flow and orifice factor data
# Note the exclusion of negative (invalid) Orifice Factor (x,y) points
library(RODBC)
db <- odbcConnect('Devel',uid='sa',pw='')
flowdat <- sqlQuery(db,paste("select AnnubarFlow as x, AnnubarOrificeFactor as y ",
                             "from   BenchFlowData ",
                             "where  AnnubarOrificeFactor>0"))
odbcClose(db)

# Populate x and y vectors for regression function
flowx <- as.vector(flowdat$x)
flowy <- as.vector(flowdat$y)

# Constants
n <- NROW(flowy)
n2 <- as.integer(n/2)
nSetPairs <- 100              # number of parameters sets to generate
nSetPairs2 <- 2*nSetPairs
nSimulationPoints <- 10000    # number of points to simulate at each predictor level

# Generate nSetPairs pairs of model parameters, each based on two randomly selected sets
# of x-y pairs, each consisting of half of the supplied pairs (one is randomly
# generated, the other is its complement)
# Save results in parSet matrix
par <- matrix(nrow=nSetPairs2, ncol=4)
colnames(par) <- c("SetID", "alpha", "beta", "gamma")
for(set in 0:(nSetPairs-1)) {
  # Generate random indices for first half-sample
  # Note the "False" replacement, so that all indices are unique
  si <- sample(1:n,n2,replace=F)
  # Generate parameters for first half-sample
  par[set*2+1,] <- c(set*2+1, nlReg(flowx[si], flowy[si]))
  # Now, the other half
  # Generate complement of first-half indices
  # si <- which(pmatch(1:n,si,nomatch=0)==0) # works, thought I would keep it
  si <- which(is.na(pmatch(1:n,si))) # NA is the default val returned by pmatch on nomatch
  par[set*2+2,] <- c(set*2+2, nlReg(flowx[si], flowy[si]))
}

# Simulate nSimulationPoints at each predictor level
# Generate matrix containing predictor index and randomly assigned parameter set index
# Each predictor observation will have nSimulationPoints repeats
# The last col will contain the corresponding calculated response value
# Note that 1:n (the number of observed predictor observations) is repeated nSimulationPoints
# (once for each response to simulate) - the indices are then sorted to accelerate quantile calcs
M <- cbind(sort(rep(1:n,nSimulationPoints)),
           sample(1:nSetPairs2, n*nSimulationPoints, replace=T), NA)
colnames(M) <- c("xi", "pari", "y")
Quant <- matrix(nrow=n, ncol=2)
colnames(Quant) <- c(".025",".975")

# Calculate response at observed x using simulated parameter levels
M[,"y"] <- par[M[,"pari"],"gamma"] +
par[M[,"pari"],"alpha"]*exp(par[M[,"pari"],"beta"]*flowx[M[,"xi"]])
# Find confidence interval quantiles
# The following functions properly, but is very slow - sorting in initial M cfg remedied delay
# for(i in 1:n) Quant[i,] <- quantile(M[which(M[,"xi"]==i),"y"], probs=c(.025, .975))
```

```
for(i in 0:(n-1))
  Quant[i+1,] <- quantile(M[(i*nSimulationPoints+1):((i+1)*nSimulationPoints),"y"],
                          probs=c(.025, .975))

# Plot supplied x-y points
plot(flowy~flowx, main="Bench Flow Model", xlab="Annubar Flow", ylab="Orifice Factor", cex=.5)

# Fit model to supplied x-y points
par <- nlReg(flowx, flowy)

# Calculate std dev of observed Orifice Factors, using fitted model as expected value
# This is used for individual confidence interval adjustment above and below that of
# model expected value
ystd <- sqrt(sum((flowy-par["gamma"]-par["alpha"]*exp(par["beta"]*flowx))^2)/n)

# Model title
text(800, .155, "Model: y = gamma + alpha * exp(beta * x)", cex=.75, adj=0)

# Plot expected value model
#lines(par["gamma"]+par["alpha"]*exp(par["beta"]*flowy)~flowx, col="blue")
curve(par["gamma"]+par["alpha"]*exp(par["beta"]*x), add=T, col="blue", n=50)
text(800, .140, "Exp value:", cex=.75, adj=0)
text(1100, .140, paste("alpha = ", round(par["alpha"],6)), cex=.75, adj=0)
text(1100, .133, paste("beta = ", round(par["beta"],6)), cex=.75, adj=0)
text(1100, .126, paste("gamma = ", round(par["gamma"],6)), cex=.75, adj=0)

# Fit model to .975 CI points (exp val CI + individ CI) and plot
par <- nlReg(flowx, Quant[,".975"])
curve(par["gamma"]+par["alpha"]*exp(par["beta"]*x)+1.96*ystd, add=T, col="red", n=50)
text(800, .110, "UCL:", cex=.75, adj=0)
text(1100, .110, paste("alpha = ", round(par["alpha"],6)), cex=.75, adj=0)
text(1100, .103, paste("beta = ", round(par["beta"],6)), cex=.75, adj=0)
text(1100, .096, paste("gamma = ", round(par["gamma"],6)), cex=.75, adj=0)
text(1100, .089, paste("indiv s = ", round(ystd,6)), cex=.75, adj=0)

# Fit model to .025 CI points (exp val CI - individ CI) and plot
par <- nlReg(flowx, Quant[,".025"])
curve(par["gamma"]+par["alpha"]*exp(par["beta"]*x)-1.96*ystd, add=T, col="red", n=50)
text(800, .073, "LCL:", cex=.75, adj=0)
text(1100, .073, paste("alpha = ", round(par["alpha"],6)), cex=.75, adj=0)
text(1100, .067, paste("beta = ", round(par["beta"],6)), cex=.75, adj=0)
text(1100, .060, paste("gamma = ", round(par["gamma"],6)), cex=.75, adj=0)
text(1100, .053, paste("indiv s = ", round(ystd,6)), cex=.75, adj=0)

# Remove M (it has 5,260,000 rows)
```