Evaluating Statistical Bounding Methods for Semi-Quantitative Non-Targeted Analysis Using ENTACT Data (R Markdown)

Louis C. Groff II

2/25/2021

This markdown document shows the statistical methods performed for the ENTACT Mixtures LC Semi-Quant manuscript.

Load necessary libraries and shift directories as needed:

```
library(ggplot2)
library(readxl)
library(xlsx)
library(investr)
library(data.table)
library(gridExtra)
datadir <- 'C:/Users/lgroff/OneDrive - Environmental Protection Agency (EPA)/Profile/Documents/Data/ENTACT Semi Quant/Standard Mixtures/JRS Tracer Analysis'
```

Load In ESI+ data:

```
posdata_1 <- read_xlsx(paste0(datadir,'/Supp_Table1_Pos.xlsx'))
posdata_2 <- read_xlsx(paste0(datadir,'/Supp_Table2_Pos.xlsx'))
```

Method 1. Inverse Prediction Using Calibration Curves:

See Section 3.2.1 in manuscript

1. Preallocate empty arrays and lists:

```
chemunique_pos <- unique(posdata_1$New_ID) #unique chemical ID list
posdata_2$y0 <- NA
posdata_2$Known_Conc <- NA
posdata_2$Conc_CC <- NA
posdata_2$Conc_CC_Upper <- NA
lmlist_pos <- vector('list',length(chemunique_pos))
plotlist_pos <- lmlist_pos
predlist_pos <- lmlist_pos
```

2. set RNG seed and determine row indices to select y0 for each individual chemical (calculated as a random number between 0-1 * length of the individual chemical data subset):

```
set.seed(16384)
tmpidx_pos<-sample(1:100,nrow(posdata_2),size=nrow(posdata_2))/100
```

3. Run cal. curve for loop for each unique chemical (including isomers)

a. Store RNG-selected observed intensity (y0) and concentration at y0, store in posdata_2.

b. calculate linear model on log(Normalized_Intensity) vs. log(Conc.), store in lmlist_pos.
c. calculate Conc_CC (calibration curve estimate) at y0 from regression coefficients, stored in posdata_2.
d. calculate 99% prediction interval (99% PI) for the regression (need n > 3 data points), store in predlist_pos.
e. Use calibrate() function within investr package to calculate the upper concentration bound from the 99% PI (need n > 3 data points), encapsulated in a try() statement since it fails for some compounds, but don't want to break the loop on a failure, store in posdata_2.
f. store ggplots of each calibration curve in plotlist_pos:

```
for (i in 1:length(chemunique_pos)){
  tmpdf <- posdata_1[posdata_1$New_ID==chemunique_pos[[i]],]
  tmpdf <- tmpdf[complete.cases(tmpdf$Normalized_Intensity),] #remove NAs

  posdata_2$y0[[i]] <-
tmpdf[[ceiling(tmpidx_pos[[i]]*nrow(tmpdf)),'Normalized_Intensity']]
  posdata_2$Known_Conc[[i]] <-
tmpdf[[ceiling(tmpidx_pos[[i]]*nrow(tmpdf)),'Concentration']]

  #store regression model objects:
  lmlist_pos[[i]] <- lm(tmpdf,formula =
log10(Normalized_Intensity)~log10(Concentration))

  #store concentration predictions for individual chemical cal curves:
  posdata_2$Conc_CC[[i]] <- 10^((log10(posdata_2$y0[[i]])-
lmlist_pos[[i]]$coefficients[1])/lmlist_pos[[i]]$coefficients[2])

  #store 99% prediction interval data for each chemical cal curve:
  predlist_pos[[i]] <- try(predict(lmlist_pos[[i]],
                           level=0.99,
                           interval='prediction'), silent=T)
  #run calibrate on n > 3 intensities to get 99% upper bound:
  tmpgg <-
ggplot(tmpdf,aes(x=log10(Concentration),y=log10(Normalized_Intensity)))+
    geom_point()+
    geom_smooth(method='lm',formula=y~x,se=FALSE)
  labs(x='Log10 Concentration',
       y= 'Log10 Normalized Intensity',
       title = unique(tmpdf$Preferred_Name))
  if (length(tmpdf$Intensity) > 3){
    tmpcal<-try(calibrate(lmlist_pos[[i]],
                      y0=log10(posdata_2$y0[[i]]),
                      level=0.99,
                      interval='inversion'),silent=T)
    #store individual regression plots in a list:
    tmpgg <- tmpgg+
      geom_ribbon(aes(ymin=predlist_pos[[i]][,'lwr'],
```

```
                    ymax=predlist_pos[[i]][,'upr']),
                alpha=0.25)
    if (class(tmpcal) != 'try-error'){
      posdata_2$Conc_CC_Upper[[i]] <- 10^tmpcal$upper
    }
  }
  plotlist_pos[[i]] <- tmpgg
}

## Warning in predict.lm(lmlist_pos[[i]], level = 0.99, interval =
"prediction"): predictions on current data refer to _future_ responses

(551 Occurrences)
```

From the above calibration curve regressions, if regression slope is zero, set intercept to NA since that leads to underestimation of concentration estimate from RF=10^intercept, store RF = y0/concentration instead:

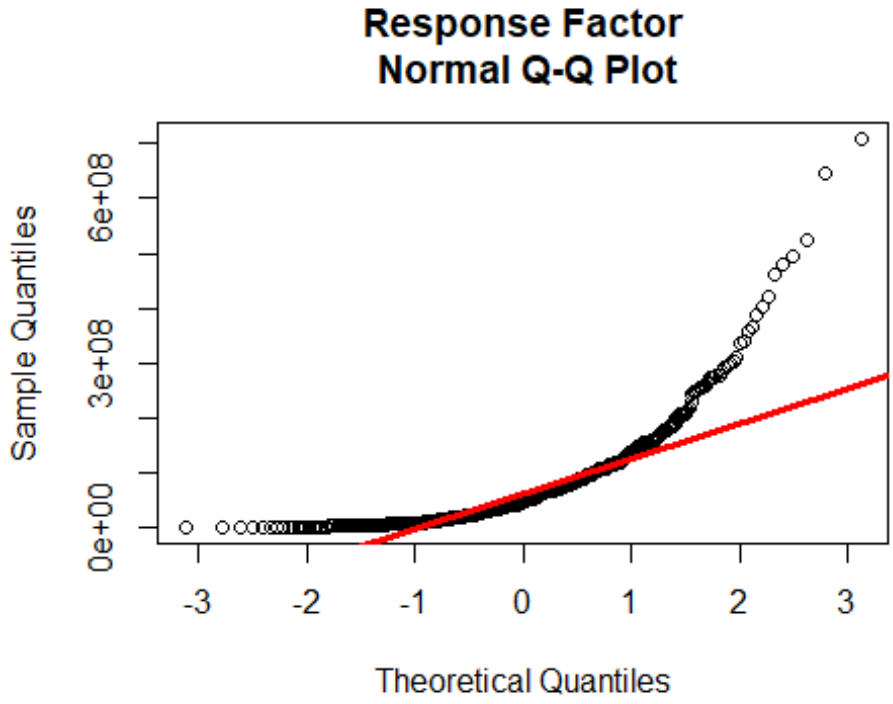
```
for(i in 1:length(posdata_2$Slope_Full)){
  if(posdata_2$Slope_Full[[i]]==0){
    posdata_2$Intercept_Full[[i]] <- NA
    posdata_2$RF[[i]] <- posdata_2$y0[[i]]/posdata_2$Known_Conc[[i]]
  }
}
```

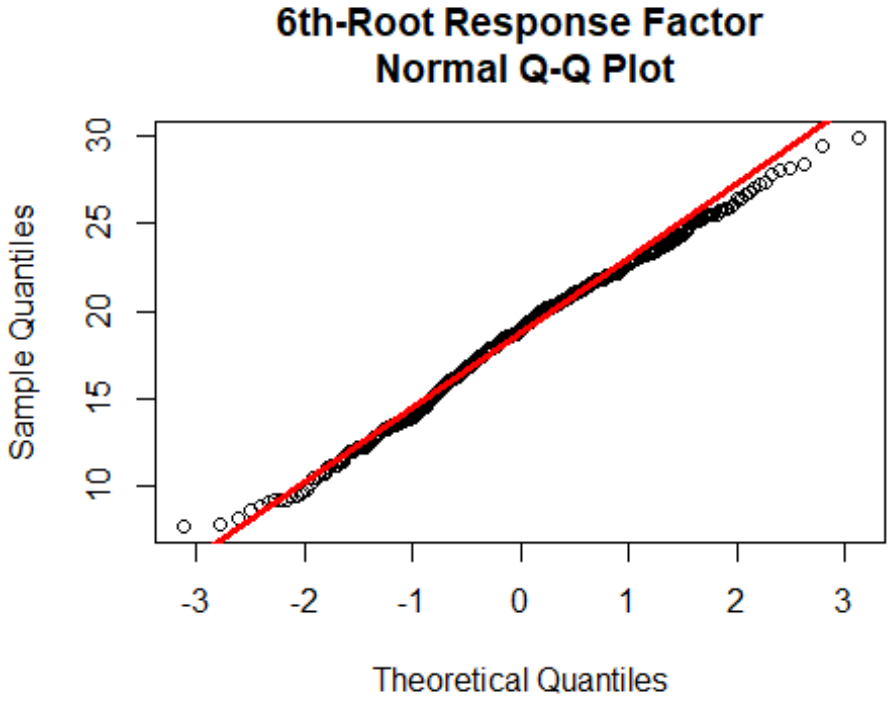
Store error quotients for calibration curve analysis in posdata_2:

```
#Conc_0.99_CC:
posdata_2$Err_CCUppervsCCEst <- posdata_2$Conc_CC_Upper/posdata_2$Conc_CC
```

Method 2: Inverse Prediction Using a Bounded Response Factor

See section 3.2.2. in manuscript. Show quantile-quantile plots for RF distribution and $RF^{(1/6)}$





Store tRF = $RF^{(1/6)}$ in posdata_2, and calculate tRF_0.01 and tRF_0.99 using Eq. 1 in manuscript.

```
alpha = 0.01 #0.01 for 99% pred. interval
level = 1-alpha/2
posdata_2$RF_6thRoot <- posdata_2$RF^(1/6) #6th-root transformation

#Upper prediction interval bound for a Normal 1D Distribution:
pos_RF6th_upper<-
mean(posdata_2$RF_6thRoot,na.rm=T)+qt(level,length(posdata_2$RF_6thRoot)-
1)*sd(posdata_2$RF_6thRoot,na.rm=T)*sqrt(1+1/length(posdata_2$RF_6thRoot))

#Lower prediction interval bound for a Normal 1D Distribution:
pos_RF6th_lower<- mean(posdata_2$RF_6thRoot,na.rm=T)-
qt(level,length(posdata_2$RF_6thRoot)-
1)*sd(posdata_2$RF_6thRoot,na.rm=T)*sqrt(1+1/length(posdata_2$RF_6thRoot))
```

Since concentration is inversely proportional to response factor (C=I/RF), use lower prediction interval bound (tRF_0.01) to calculate upper bound concentration predictions (Conc_0.99_RF). Store predictions and error quotients in posdata_2:

```
#Conc_0.99_RF:
posdata_2$Conc_DefaultRF_Upper <- posdata_2$y0/pos_RF6th_lower^6

#Error Quotients:
posdata_2$Err_RFUppervsCCEst<-
posdata_2$Conc_DefaultRF_Upper/posdata_2$Conc_CC
```

Method 3: Inverse Prediction Using Ionization Efficiency Estimation

See section 3.2.3 in manuscript. Load in data from Kruve, et al.:

```
datadir2<-'C:/Users/lgroff/OneDrive - Environmental Protection Agency
(EPA)/Profile/Documents/Data/ENTACT Semi Quant/Standard Mixtures/Anneli
Updates'
posdata_anneli <- read_xlsx(paste0(datadir2,'/ENTACT_pos_pred.xlsx'))
```

Pre-allocate empty columns, filter updated log(IE) predictions based on DTXSID matches:

```
posdata_2$logIE_Pred <- NA
posdata_anneli <- posdata_anneli[posdata_anneli$DTXSID %in%
posdata_2$DTXSID,]
for (i in 1:length(posdata_2$New_ID)){
  #If length of matched filtered subset is > 0, store the unique log(IE)
  #value in posdata_2.
  if (sum(posdata_anneli$DTXSID == posdata_2$New_ID[[i]])>0){
    posdata_2$logIE_Pred[[i]] <-
as.numeric(unique(posdata_anneli[posdata_anneli$DTXSID ==
posdata_2$New_ID[[i]],'logIE_pred_new'])[[1]][[1]])
  }
  #include log(IE) for isomers (store one value for all isomers):
  if (sum(posdata_anneli$DTXSID == posdata_2$DTXSID[[i]])>0 &
      sum(posdata_anneli$DTXSID == posdata_2$New_ID[[i]])==0){
    posdata_2$logIE_Pred[[i]] <-
as.numeric(unique(posdata_anneli[posdata_anneli$DTXSID ==
posdata_2$DTXSID[[i]],'logIE_pred_new'])[[1]][[1]])
  }
}

## Warning: NAs introduced by coercion
```

Compute linear regression on log(IE) vs. tRF:

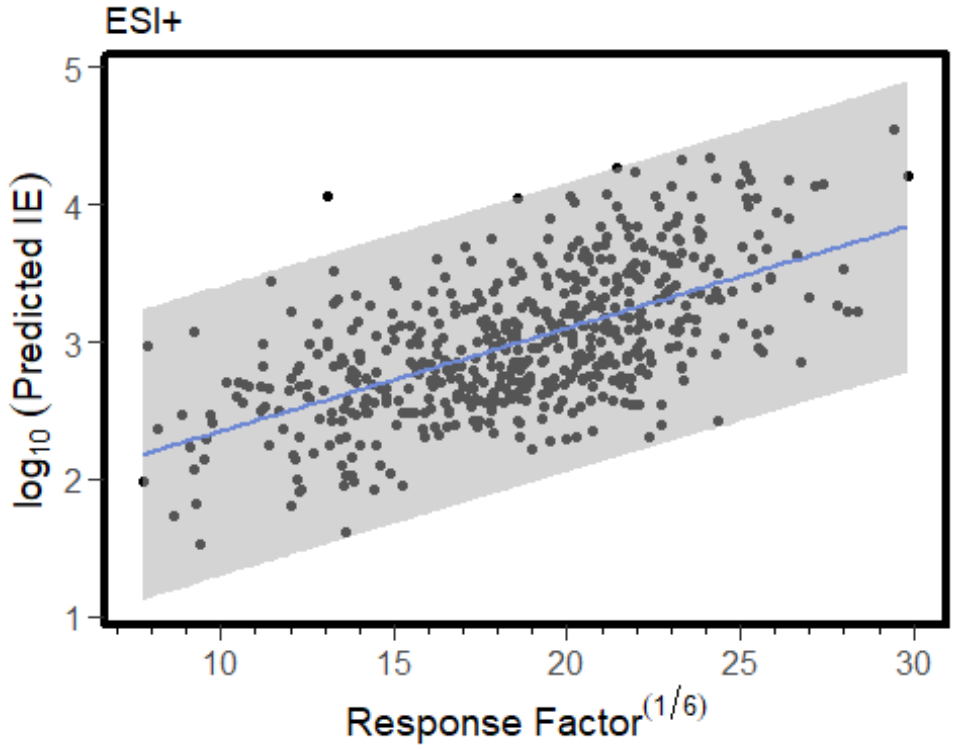
```
anneli_lm_pos<-lm(posdata_2,
                  formula=posdata_2$logIE_Pred~posdata_2$RF_6thRoot)
```

The slope is 0.0750841 and intercept is 1.5997875

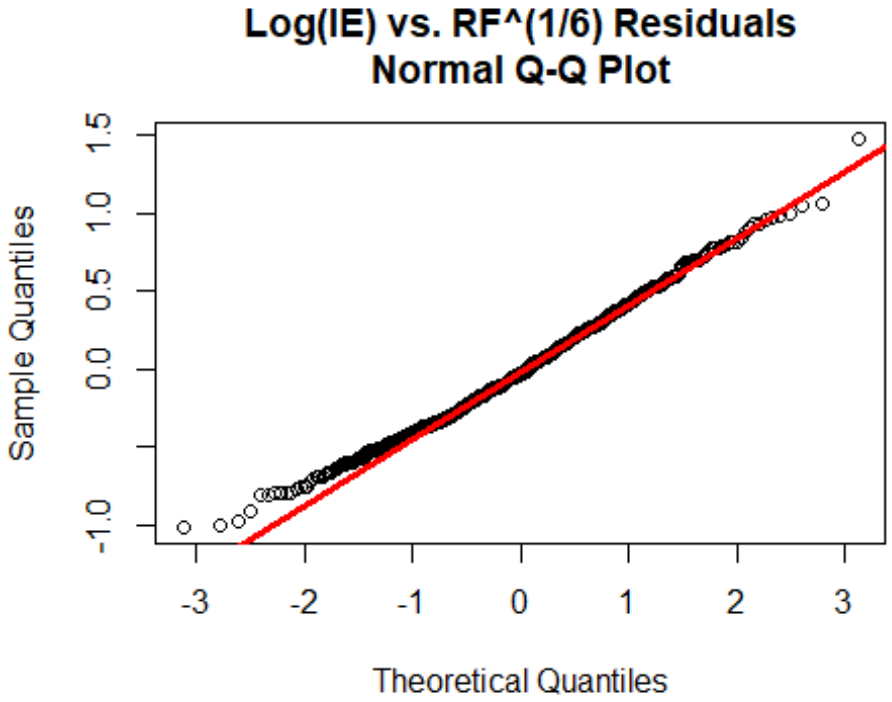
Calculate 99% Prediction Interval:

```
anneli_predint_pos<-predict(anneli_lm_pos,
                            newdata=posdata_2,
                            interval='prediction',
                            level=0.99)
```

Plot results:



Examine fit residuals:



Allocate empty columns for calibrate() analysis to produce IE-predicted RFs (particularly, tRF_0.01_IE):

```
posdata_2$logIE_RF_0.99 <- NA
posdata_2$logIE_RF_0.01 <- NA
posdata_2$logIE_RF_fit <- NA
```

Run inverse predictions using the log IE for each chemical:

```
for (i in 1:length(posdata_2$RF_6thRoot)){
  if (!is.na(posdata_2$logIE_Pred[[i]])){
    tmpcal<-calibrate(formula=posdata_2$logIE_Pred~posdata_2$RF_6thRoot,
                      y0=posdata_2$logIE_Pred[[i]],
                      interval='inversion',
                      level=0.99)
    posdata_2$logIE_RF_0.01[[i]]<-tmpcal$lower
    posdata_2$logIE_RF_0.99[[i]]<-tmpcal$upper
    posdata_2$logIE_RF_fit[[i]]<-tmpcal$estimate
  }
}
```

To ensure our error quotient is never larger than the largest error produced by the default RF method, we impute tRF_0.01 in for any tRF_0.01_IE < tRF_0.01 from the 1D tRF distribution:

```
for (i in 1:length(posdata_2$logIE_RF_0.01)){
  if (!is.na(posdata_2$logIE_RF_0.01[[i]]) &
      posdata_2$logIE_RF_0.01[[i]] < pos_RF6th_lower){
    posdata_2$logIE_RF_0.01[[i]]<-pos_RF6th_lower
  }
}
```

Following imputation, again, we use C=I/RF_0.01_IE^6 to determine Conc_0.99_IE, and calculate the error quotients:

```
#Conc_0.99_IE:
posdata_2$logIE_Conc_Upper <- posdata_2$y0/posdata_2$logIE_RF_0.01^6

#Error Quotients:
posdata_2$Err_IEUppervsCCEst <- posdata_2$logIE_Conc_Upper/posdata_2$Conc_CC
```