Retention Time Correction Documentation: Code accompaniment to the report entitled 'Visualization, Quantification and Alignment of Spectral Drift in Population Scale Untargeted Metabolomics Data'

Mir Henglin et al.

December 23, 2016

citation: TBA

Retention Time Correction with R

Useful Libraries

```
### easy-read syntax and data-manipulation ###
library(magrittr)
library(dplyr)
library(purrr)

### Data import ###
library(readr)

### splines ###
library(gam)

### plotting ###
library(ggplot2)

### xml file manipulation ###
library(rvest)
library(xml2)
```

Sample Data Generation

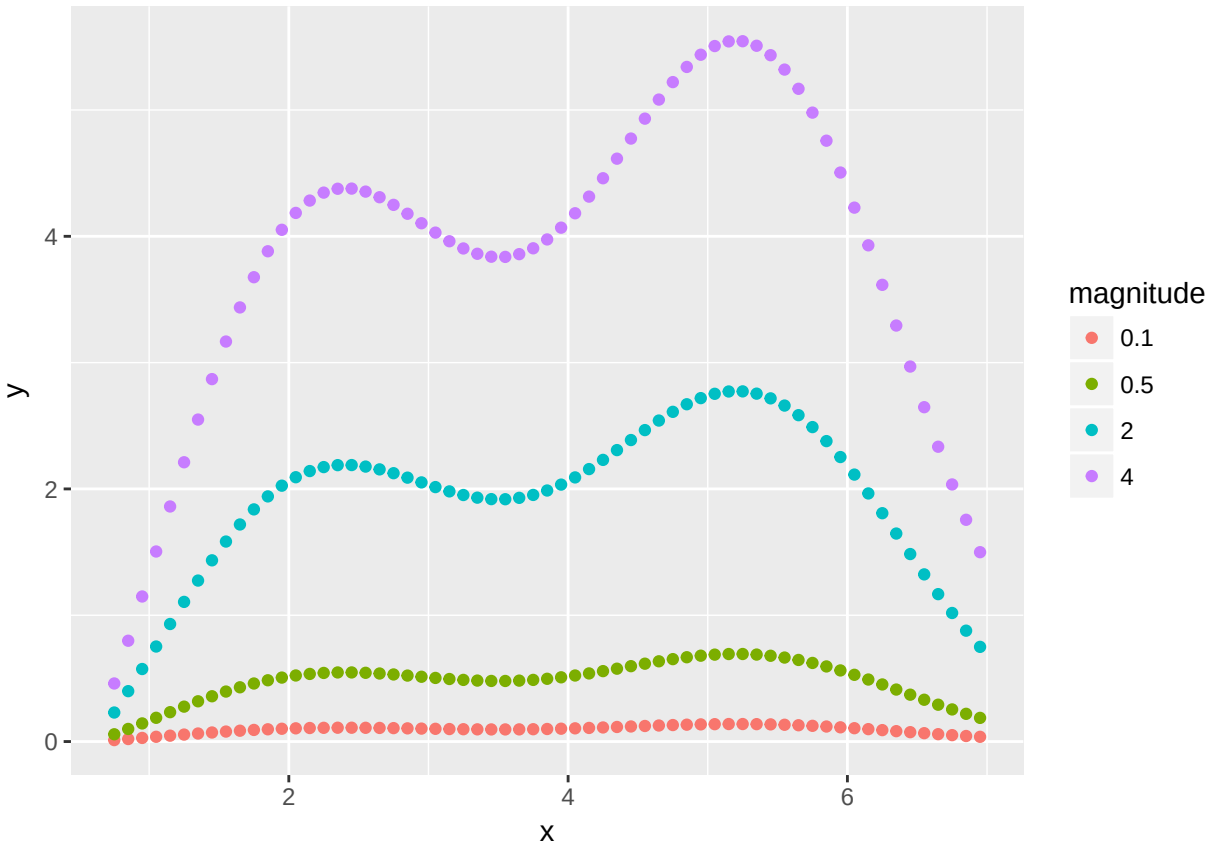
For demonstration, we will generate a dataset of random values that will be representative of the original referent dataset that was used to develop the method. Specifically, the generated dataset will represent a subset of measurements from the larger referent dataset.

```
nPlateWells <- 6 # Number of samples
nMetabolites <- 60 # Number of landmark metabolites per sample

# max and min values over which retention time was measures
rtMin <- 0.75
```

```
rtMax <- 7

# parameters controlling the shape and noise of the necessary corrections
diffRange <-
  0.25

diffNoise <-
  0.05

# We will generate samples that deviate from the target according to the function specified here.
diffShape <- function(x, magnitude) {
  #sign <- ifelse(rbernoulli(1), 1, -1)
  #magnitude <- runif(0.5, 2)
  shape <-
    (sin((x - rtMin) * pi / rtMax) + 0.5 * sin(2 * (x - rtMin) * pi / rtMax + 0.5) ^ 2)


  # sign * magnitude * shape
  magnitude * shape
}
```

Here, we plot examples of deviation from the target that will be generated.

```
xVals <-
  seq(rtMin, rtMax, by = 0.1)

data_frame(x = rep(xVals, each = 4),
           magnitude = rep(c(0.1, 0.5, 2, 4), times = length(xVals))
) %>%
  group_by(magnitude) %>%
  mutate(y = diffShape(x, unique(magnitude))) %>%
  ungroup() %>%
  mutate(magnitude = as.factor(magnitude)) %>%
  ggplot() +
  geom_point(aes(x = x, y = y, colour = magnitude))
```



As shown, deviation from the target follows curvilinear pattern, and this is similar to the pattern of deviations observed in the original referent data.

```
# Control the shape of the deviations
shapeMagnitudes <-
  seq(from = -diffRange, to = diffRange, length.out = nPlateWells + 1) %>%
  discard(~ .==0)

# Generate unqiue metabolite IDs
metaboliteID <- runif(nMetabolites, 100, 900)
while (length(unique(metaboliteID)) < nMetabolites) {
  metaboliteID <- runif(nMetabolites, 100, 900)
}

plateWellID <- seq_len(nPlateWells)

modelData <-
  expand.grid(metaboliteID, plateWellID) %>%
  as_data_frame() %>%
  setNames(c('metaboliteID', 'plateWellID')) %>%
  mutate_all(funs(as.factor))

modelData %<>%
  group_by(metaboliteID) %>%
  mutate(retentionTimeMean = runif(1, rtMin, rtMax),
         diffShape = diffShape(retentionTimeMean, shapeMagnitudes),
         diffNoise = rnorm(n(), sd = diffNoise)) %>% # Add noise to the shape of the differences
```

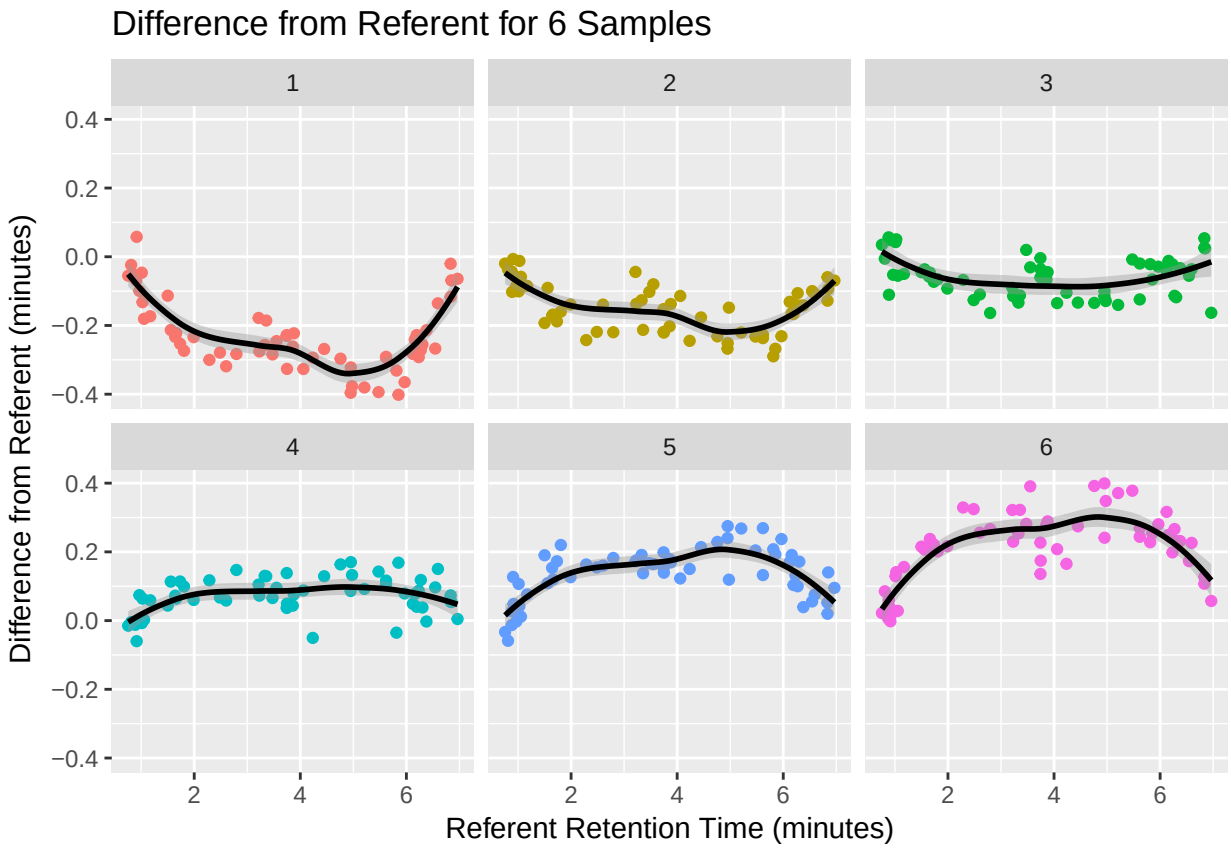
```
  ungroup()

modelData %<>%
  mutate(diff = diffShape + diffNoise,
         retentionTime = retentionTimeMean + diff) %>%
  # mutate(retentionTime = abs(retentionTime)) %>%
  select(-diffShape, -diffNoise) %>%
  arrange(plateWellID, metaboliteID)

modelData
```

```
## # A tibble: 360 × 5
##        metaboliteID plateWellID retentionTimeMean        diff
##               <fctr>      <fctr>             <dbl>       <dbl>
## 1  104.993790015578           1         0.9668951 -0.09940732
## 2  110.662939585745           1         1.9890770 -0.23390362
## 3  114.552807062864           1         5.6164380 -0.29132220
## 4  138.667912222445           1         6.2719815 -0.27158995
## 5   154.43371757865           1         6.8341694 -0.11657017
## 6  169.942902959883           1         0.8880220 -0.05384306
## 7  188.658790104091           1         1.0257058 -0.13210443
## 8  194.186932593584           1         0.8107903 -0.02390226
## 9  236.605985276401           1         2.7957456 -0.28306024
## 10 237.198729626834           1         3.2287698 -0.27558282
## # ... with 350 more rows, and 1 more variables: retentionTime <dbl>
```

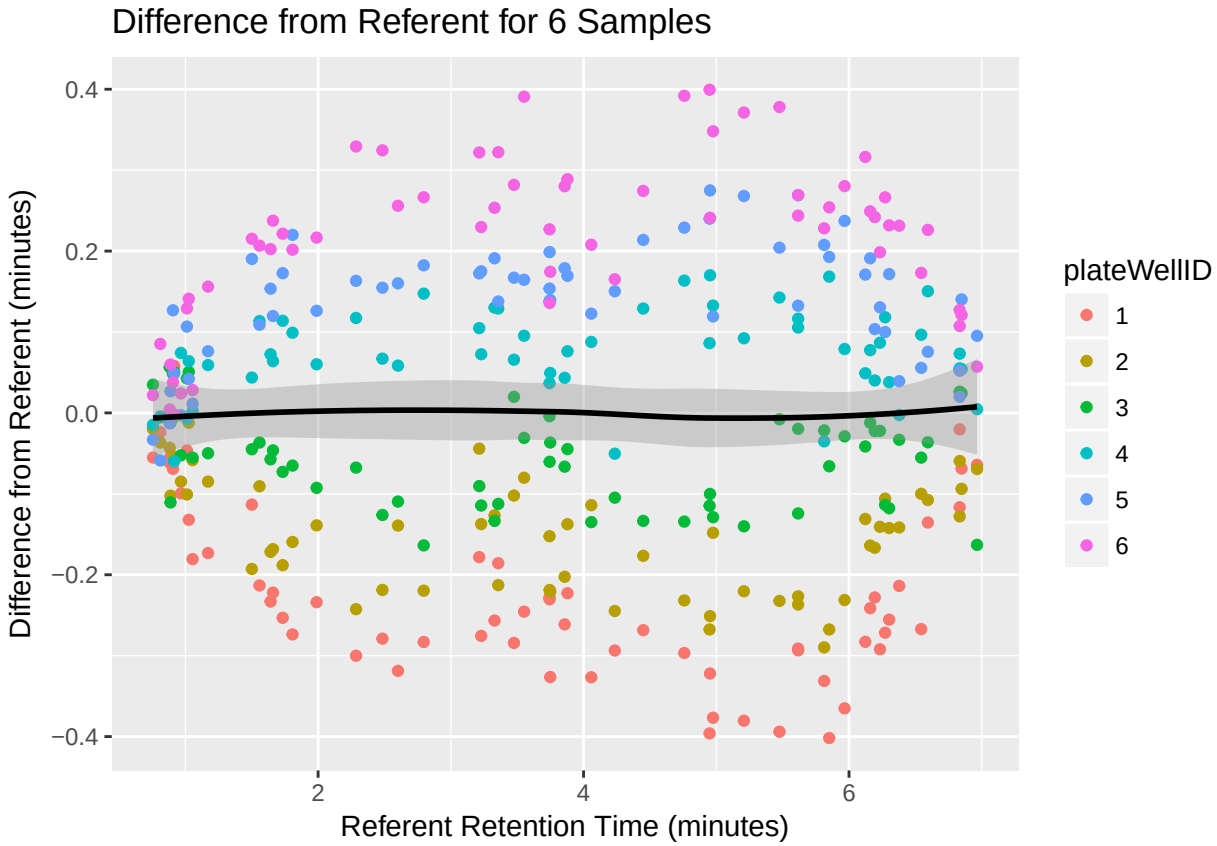
For our sample data, the columns are:

- metaboliteID
  - Numeric ID representing a unique measured eicosanoid metabolite.
- plateWellID
  - Numeric ID representing a sample in which metabolites were measured.
- retentionTimeMean
  - The mean retention time of metabolite across all samples. For our dataset, the mean is assumed to be calculated from the whole dataset, of which our sample data is a subset. The mean is the referent value that we attempt to adjust values to.
- retentionTime
  - The measured retention time of a metabolite in a given sample.
- diff
  - The difference between the referent value and the measured value. These differences are what we attempt to model as a function of retention time.

```
modelData %>%
  ggplot(aes(x = retentionTimeMean, y = diff, colour = plateWellID)) +
  geom_point() +
  scale_color_discrete(guide = FALSE) +
  stat_smooth(colour = 'black') +
  facet_wrap(~ plateWellID) +
  ggtitle('Difference from Referent for 6 Samples') +
  ylab('Difference from Referent (minutes)') +
  xlab('Referent Retention Time (minutes)')
```




```
# TODO remove colour guide
modelData %>%
  ggplot(aes(x = retentionTimeMean, y = diff, colour = plateWellID)) +
  geom_point() +
  stat_smooth(colour = 'black') +
  ggtitle('Difference from Referent for 6 Samples') +
  ylab('Difference from Referent (minutes)') +
  xlab('Referent Retention Time (minutes)')
```



Although not directly represented in our example data, it may be the case that based on domain knowledge of how the column/LCMS works, we expect that the behavior of the difference from the referent to behave differently in different time regions. For this example, we will say that we expect the beginning (<1 minute,) the end (>6 minute,) and the middle to behave differently from one another. This is similar to actual knowledge that we possessed about our data. To represent this, we create a factor variable.

```
time_region <- function(x) {
  ifelse(x <= 1, 1,
         ifelse(x >= 6, 3, 2))
}

modelData %<>%
  mutate(timeRegion = time_region(retentionTime)) %>%
  mutate(timeRegion = as.factor(timeRegion))
```

Modeling

We create a model by splitting our dataset into an individual dataset for each sample, and then fitting an additive model with terms for a smoothing spline based on retention time, a variable for the time region, and an interaction between these terms.

```
models <-
  modelData %>%
  split(.$plateWellID) %>%
  map(function(x) {
    gam(
```

```
      diff ~ s(retentionTime, 16) + timeRegion + timeRegion * s(retentionTime, 16) ,
      data = x
    )
  })

models
```

```
## $`1`
## Call:
## gam(formula = diff ~ s(retentionTime, 16) + timeRegion + timeRegion *
##     s(retentionTime, 16), data = x)
##
## Degrees of Freedom: 59 total; 39.0005 Residual
## Residual Deviance: 0.1011
##
## $`2`
## Call:
## gam(formula = diff ~ s(retentionTime, 16) + timeRegion + timeRegion *
##     s(retentionTime, 16), data = x)
##
## Degrees of Freedom: 59 total; 39.00086 Residual
## Residual Deviance: 0.07201
##
## $`3`
## Call:
## gam(formula = diff ~ s(retentionTime, 16) + timeRegion + timeRegion *
##     s(retentionTime, 16), data = x)
##
## Degrees of Freedom: 59 total; 39.00026 Residual
## Residual Deviance: 0.08239
##
## $`4`
## Call:
## gam(formula = diff ~ s(retentionTime, 16) + timeRegion + timeRegion *
##     s(retentionTime, 16), data = x)
##
## Degrees of Freedom: 59 total; 39.00042 Residual
## Residual Deviance: 0.06804
##
## $`5`
## Call:
## gam(formula = diff ~ s(retentionTime, 16) + timeRegion + timeRegion *
##     s(retentionTime, 16), data = x)
##
## Degrees of Freedom: 59 total; 39.0001 Residual
## Residual Deviance: 0.07041
##
## $`6`
## Call:
## gam(formula = diff ~ s(retentionTime, 16) + timeRegion + timeRegion *
##     s(retentionTime, 16), data = x)
##
## Degrees of Freedom: 59 total; 39.00093 Residual
## Residual Deviance: 0.10173
```

```
summary(models[[1]])
```

```
##
## Call: gam(formula = diff ~ s(retentionTime, 16) + timeRegion + timeRegion *
##     s(retentionTime, 16), data = x)
## Deviance Residuals:
##        Min         1Q     Median         3Q        Max
## -0.1001966 -0.0241879  0.0007236  0.0268328  0.1405877
##
## (Dispersion Parameter for gaussian family taken to be 0.0026)
##
##     Null Deviance: 0.6827 on 59 degrees of freedom
## Residual Deviance: 0.1011 on 39.0005 degrees of freedom
## AIC: -168.8887
##
## Number of Local Scoring Iterations: 30
##
## Anova for Parametric Effects
##                                 Df   Sum Sq  Mean Sq F value    Pr(>F)
## s(retentionTime, 16)             1 0.081914 0.081914 31.5997 1.738e-06 ***
## timeRegion                       2 0.049386 0.024693  9.5257  0.000428 ***
## s(retentionTime, 16):timeRegion  2 0.020577 0.010289  3.9691  0.026978 *
## Residuals                       39 0.101098 0.002592
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                                 Npar Df Npar F   Pr(F)
## (Intercept)
## s(retentionTime, 16)                 15 1.7382 0.08312 .
## timeRegion
## s(retentionTime, 16):timeRegion
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Specific to our real data, we observed after initial correction that many metabolites in the later retention times were over-corrected. Thus, we needed to regularize the function so that, in the later retention times, the predicted difference would accelerate to 0. We accomplished this manually generating data points with a difference of 0 in the later retention time range. An example of this is shown below.

```
models <-
  modelData %>%
  split(.$plateWellID) %>%
  map(function(x) {
    x %<>%
      bind_rows(data.frame(rt = rep(seq(6.5 , 7 , .1), 1), # add 0s to weigh down the function to 0 at
                           diff = 0,
                           timezone = 3))
  }) %>%
  map(function(x) {
    gam(
      diff ~ s(retentionTime, 16) + timeRegion + timeRegion * s(retentionTime, 16) ,
      data = x
    )
  })
```

mzXML Processing

After creating models that predict a difference based on a sampleID and a retention time, we need to process the mzMine outputs. In doing so, we will apply the correction models calculated using the metabolite standards to all metabolites measured in a sample. The code below shows an example of such a workflow. Note that much the code below is specific to our particular data format and workflow.

First we create a vector of mzXML documents to iterate over. Each mzXML file corresponds to the measurements of a single sample.

```
# ALL CODE BELOW IS NOT EVALUATED #
# SOME PSEUDOCODE #

dir <-
  'data/folderWithMZMINExmls/'

mzXMLdocs <- list.files(dir)

for (d in mzXMLdocs) {
```

For each file, it is important to use the model that corresponds to its sample. Here, we extract the sampleID from the file name, which we will use to grab the appropriate model. We also import the mzXML document, and extract and clean the retention times.

```
  # Get the platewell identifier from the fileName
  pwid <-
    stringr::str_extract(d, 'plateWellID-regex')

  doc <-
    read_xml(paste0(dir, d))

  # Extract Scan Elements and retention times from XML
  scanNodes <-
    xml_children(doc)[[1]] %>%
    xml_children() %>%
    .[xml_name(.) == "scan"]

  # cleaning and extraction
  retentionTimes <-
    xml_attr(scanNodes, 'retentionTime') %>%
    {
      gsub('^PT', '', .)
    } %>%
    {
      gsub('S$', '', .)
    } %>%
    as.numeric() %>%
    {
      . / 60 #  minute/second conversion
    }
```

Once we have extracted the full set of retention times, we need to calculate the predicted difference from the referent value. To do this, we first organize the extracted retention times into a data-frame and calculate the time-region factor values. We obtain the model corresponding to our mzXML file, predict the expected difference, and use the difference to calculate the corrected retention time.

```
#creating a dataframe of values to predict on
predDat <-
  data.frame(retentionTimes,
             time_region(retentionTimes)) %>%
  setNames(c('retentionTime', 'timeRegion'))

predDat$timeRegion %<>% factor(levels = levels(modelData$timeRegion))

# Grab the model corresponding to the plate and well combo of the data
model <-
  models[[pwid]]

predDiffs <-
  predict(model, newdata = predDat) %>%
  as.vector()

correctedRetentionTimes <-
  retentionTimes %>%
  `-`(predDiffs) %>%
  `*`(60)
```

Overcorrection

Note that this issue is possibly specific to our data and mzMine.

In mzXML files, the metabolites are ordered by retention time. After correction, these metabolites must maintain this same order. In other words, no metabolite can be corrected to have a retention time smaller than that of a metabolite that it was previously larger than, or larger than a metabolite that it was previously smaller than. This restriction is also not unreasonable in general, and might be a desirable restriction regardless of software limitations. We occasionally encountered this issue during our data processing. To account for this requirement, we had to apply a 'downstream' correction factor in every instance that such an error was detected. After applying a correction, we calculated the difference in retention times between adjacent metabolites according to the file order. If any difference was negative, this indicated that this issue had occurred. A correction factor was calculated by adding the difference and a small value 'delta', and then adding this correction to all 'downstream' retention times. This process was repeated until no errors were detected. An example of this workflow is below.

```
# Adjust for those that were over corrected
# Corrected final values must be constantly increasing (software constraint)
overcorrected <-
c(FALSE, diff(correctedRetentionTimes) <= 0)

while (any(overcorrected)) {
  # index of first overcorrected time
  oc <-
  which.max(overcorrected)

  # Overcorrected time, and all times after it, are shifted such that the overcorrected time is delta
  delta <- 0.0001

  correction <-
  (correctedRetentionTimes[(oc - 1)] + delta) - correctedRetentionTimes[(oc)]

  correctedRetentionTimes[oc:length(correctedRetentionTimes)] <-
```

```
    correctedRetentionTimes[oc:length(correctedRetentionTimes)] + correction

    overcorrected <-
    c(FALSE, diff(correctedRetentionTimes) <= 0)
  }

  stopifnot(all(correctedRetentionTimes > 0))
  stopifnot(all(diff(correctedRetentionTimes) > 0))
```

Finally, after calculating and correcting our new retention times, we reformatted them to be added to the mzXML file, and then exported the corrected mzXML document.

```
  correctedFormattedRetentionTimes <-
  correctedRetentionTimes %>%
  {
    paste0("PT", ., "S")
  }

  for (i in 1:length(newRetentionTimes2)) {
    xml_attr(scanNodes[i], 'retentionTime') <-
    correctedFormattedRetentionTimes[i]
  }

  stopifnot(all(xml_attr(scanNodes, 'retentionTime') == correctedFormattedRetentionTimes))

  write_xml(doc, paste0(outputpath, suffix, '-', d))
}
```