Supplementary Material

July 15, 2008

This document contains the supplementary material for the document "Comparison of algorithms for pre-processing of SELDI-TOF mass spectrometry data". It includes:

- Settings used to perform peak quantification and peak detection for each of the algorithms included in the study: Ciphergen, PROcess, Cromwell, SpecAling and MassSpecWavelet.
- The analysis of the CV's using the detected peaks produced with MassSpecWavelet.
- The analysis of the CV's using different cutoffs for noise-definition.
- The mean FDR-sensitivity curves before joining the points with line segments.
- Table summarizing peak counts.
- R Code for generating the simulated data.

1 Settings for Peak Quantification

The parameters were modified when indications for doing it were provided, otherwise the defaults were used.

- **Ciphergen**. We use the default protocol to perform filtering, baseline subtraction and normalization. The settings used in the "Analysis Protocol Properties" dialog are the following. In the "Baseline" tab the options "Smooth before fitting baseline" and "Automatic" were selected and we used a "window size" equal to 25. In the "Filtering" tab the options "On","Average", "Signal enhance" and "Scale to 100" were marked and "Width" was set equal to 0.2. In the "Noise tab" we set the "Segment for measuring local noise from" to "1500 Da".

  To perform normalization we used the "Normalize spectra" dialog, the options "Intensity", "Total Ion Current", "End at smallest end" and "Baseline subtraction on" were selected and "Min M/Z" was set equal to 1500.

- **PROcess**. R code with the options used in our analysis appears below.

```
# Baseline subtraction.
# "ms.raw" is a matrix with the raw data
method = # "loess" or "approx"
# In our paper "approx" correspond to PROcess1
# and "loess" to PROcess2
bw = 0.1  # This is required when method = "loess"
ms.bseoff <- bslnoff(ms.raw, method = method, bw = bw, plot = FALSE)


# Normalization.
# bs.mat a matrix with the baseline-subtracted data
cutoff = 1500
renorm.mat <-renorm(bs.mat, cutoff = cutoff)
```

- **Cromwell**. To use this algorithm we need to set a threshold for wavelet denoising, t. By using SiZer plots we set $t = 4$. Matlab instructions used in our analysis are presented below.

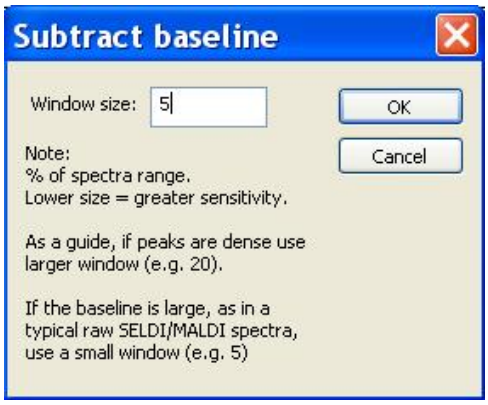
```
% Baseline subtraction and denoising.
% M is a matrix with mz values in the first column
%and the intensities in the second
t = 4;
saturate = sum(M(:,1) < 1500,1)+1;
[baselineCorrected, smoothSpec]  =
waveletSmoothAndBaselineCorrect(M(:,2)', t, saturate);
```

  Normalization is performed by dividing the intensities for each baseline subtracted and denoised spectrum by its mean.

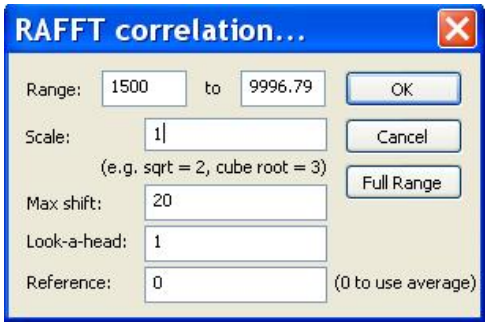
- **SpecAlign**. The following pre-processing steps were performed. They were applied as listed. The figure shows the dialogs in the SpecAlign software to perform the pre-processing with the settings we used.

  1. Filtering.

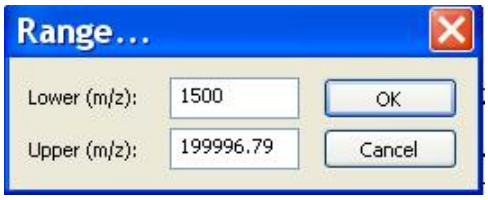
  2. Baseline



3. Align



4. Normalization



2 Settings for Peak Detection

We perform peak detection on the simulated data by using the following settings.

- **Ciphergen**. Filtering, baseline subtraction and normalization were performed as described in Section 5. Peak detection was performed using the additional tool Biomarker wizard. In the "Biomarker Wizard - Generate new clusters" dialog the options "Override threshold for user-detected peaks", "Second Pass" and "Add estimated peaks to complete clusters" were turned on. In addition, we set the "Cluster Mass Window" to 0.3. The combinations for the S/N associated to "First Pass" and "Second Pass", respectively, were $(2, 1)$, $(2.4, 1)$, $(2.6, 1)$, $(2.8, 1)$, $(3, 2)$, $(3.2, 2)$, $(3.4, 2)$, $(3.6, 2)$, $(3.8, 2)$, $(4, 2)$, $(4.5, 2)$, $(5, 2)$, $(5.5, 2)$, $(6, 2)$, $(6.5, 2)$ and $(7, 2)$. Finally, the results were exported from the "Summary" tab of the "Biomarker Properties" dialog.

- **PROcess**. We started computing the mean spectrum of the raw spectra. Then, the mean spectra was baseline subtracted using the instructions and settings described in Section 5, with the option "method" set to "approx". The default values to perform peak detection didn't work correctly with our simulated data sets, specifically the number of detected peaks was very small and even after changing the signal-to-noise ratio many noticeable, strong peaks were not detected. There is no systematic way for tunning the parameters so we adjusted them by-eye. We selected a combination that allowed us to detect the strongest peaks.

```
# Peak Detection
#mean.bsoff is the baseline-subtracted mean spectrum.
SoN = # We used each of the values
#in the vector seq(1.5,15,by=.3)

zerothrsh = 5
ratio = 0.001
prec = 0.003

peakMean <- isPeak(mean.bsoff ,SoN = SoN[k],span = 81,sm.span=11,
plot=FALSE, add = FALSE,
zerothrsh=zerothrsh,
area.w = prec,
ratio = ratio)
```

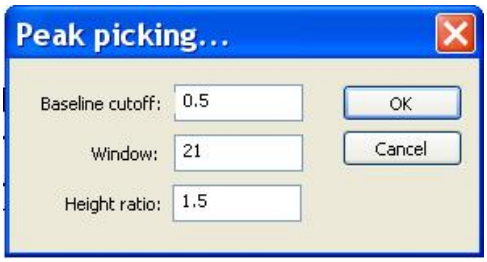
- **Cromwell** We started computing the mean spectrum from the raw spectra. By using SiZer plots we selected a wavelet threshold equal to 40. The pre-processing of the mean spectrum and the settings for peak detection appear below.

```
meanmz  % vector with the mz values for the mean spectrum
meanint_vec % intensities in the mean spectrum
SoN = 0:.1:10;

threshold = 40;  % Based on Sizer plots
L = 10;
saturate = sum(meanmz < 1500)+1;
% For smoothing and baseline correction
[baselineCorrected, smoothSpec] = waveletSmoothAndBaselineCorrect(meanint_vec, thres
bc = basecorr2(meanint_vec, meanmz, saturate, L, threshold);
load peakLocations  % load the vector of peaks
sn = signal2noise(meanint_vec, baselineCorrected, smoothSpec);


for i = 1:length(SoN)
% Compute the results for each SoN
peaksSoN = peaks(sn(1,peaks) > SoN(i))';  % (peaks>0)
peaksMZ = meanmz(peaksSoN);
dlmwrite(foundpeaks_name,peaksMZ, 'delimiter', ',', 'precision', 15);
end
```

- **SpecAlign**. This software performs peak detection on the individual spectra. The pre-processing steps prior peak detection coincide with those in Section 5. Peak detection was performed using the following settings.



- **MassSpecWavelet**. Peak detection is performed on the mean spectrum of the raw spectra. No additional pre-processing is required. The R code appears below.

```
SoN  # We used each value in the vector  seq(0.1,20,.5)
ampTh = 0

ms.mean.mat # matrix with the two columns.
# The first one contain the mz-values and the
# second the intensities of the mean spectrum.

int.vec <- ms.mean.mat[,2]
names(int.vec) <- c()
peakInfo <- peakDetectionCWT(int.vec, SNR.Th=SoN[k], amp.Th = ampTh)
majorPeakInfo = peakInfo$majorPeakInfo
peakIndex <- majorPeakInfo$peakIndex

peaks.estimated <- ms.mean.mat[peakIndex,1]
```

3 Analysis using MassSpecWavelet Peaks

To compare the algorithms in terms of the reproducibility of their quantified intensities we fixed a set of peak locations. In the manuscript such set of detected peaks was obtained by using the algorithms in the Ciphergen software. Here we present the results when the detected peaks are found with MassSpecWavelet. The results for both sets of detected peaks are similar. Figure 1 shows the coefficients of variation, CV, divided by algorithm and laser intensity.

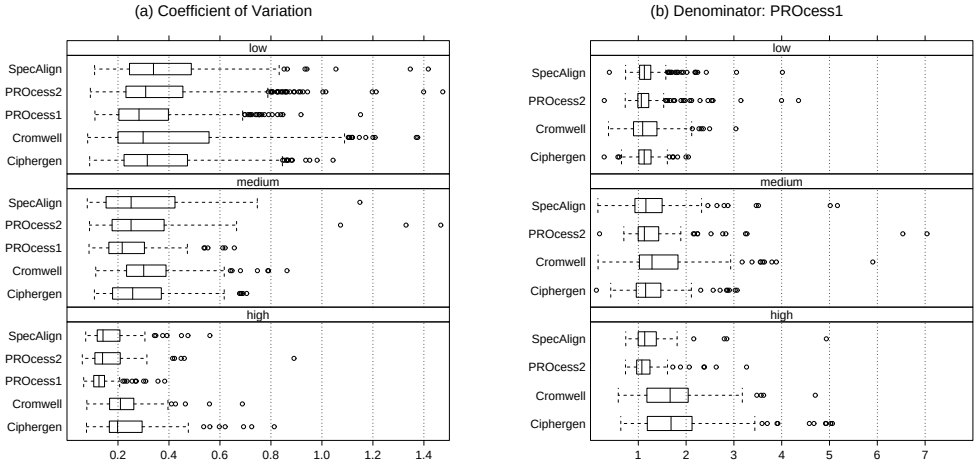


Figure 1: (a) Comparison of algorithms based on the coefficient of variation (CV) of peak intensities. (b) Ratio of CV's with PROcess1 in the denominator.

We log-transformed the data and fitted a repeated measures ANOVA model with between subjects factors, as described in the manuscript. The effects for "algorithm", "laser intensity" and their interaction are all significant. The p-values are smaller than 0.002.

4 Analysis using Different Cutoffs for "Noise-Definition"

To pre-process an spectrum one needs to specify a cutoff that defines a low-mass region where the spectrum is not reliable because is dominated by the matrix signal. The results reported in the manuscript were obtained by using a cutoff equal to 1500 Da. We repeated the analysis using the cutoffs 1500, 9000 and 28000 Da for low, medium and high laser intensity, respectively. For Ciphergen we used the cutoffs to set the parameters in the "Noise tab" of the "Analysis Protocol Properties" dialog by setting the "Segment for measuring local noise from" to "1500 Da", "9000 Da" and "28000 Da", respectively. For all the methods the low-mass region defined by the respective cutoff was not included to perform normalization. Using the Ciphergen software with the modified settings described above we performed peak detection tunning the parameters so that we obtained a set of "plausible" peaks. Such set of peaks was used to perform peak quantification with all the algorithms. Figure 2 shows the respective CV's, which are similar to those reported in the manuscript—where the cutoff of 1500 Da was used.

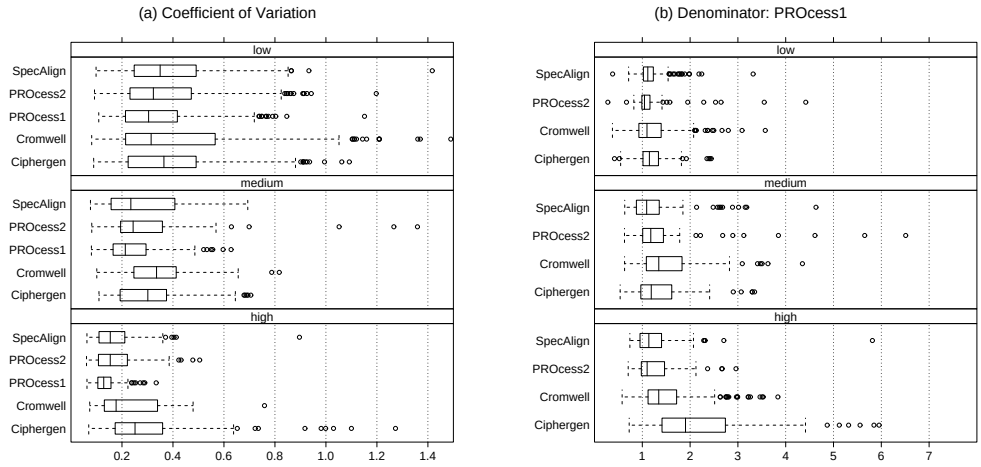


Figure 2: (a) Comparison of algorithms based on the coefficient of variation (CV) of peak intensities. (b) Ratio of CV's with PROcess1 in the denominator.

5 Mean FDR-Sensitivity Curves

In Figure 3 each point represents the mean FDR and sensitivity for a given signal-to-noise threshold.

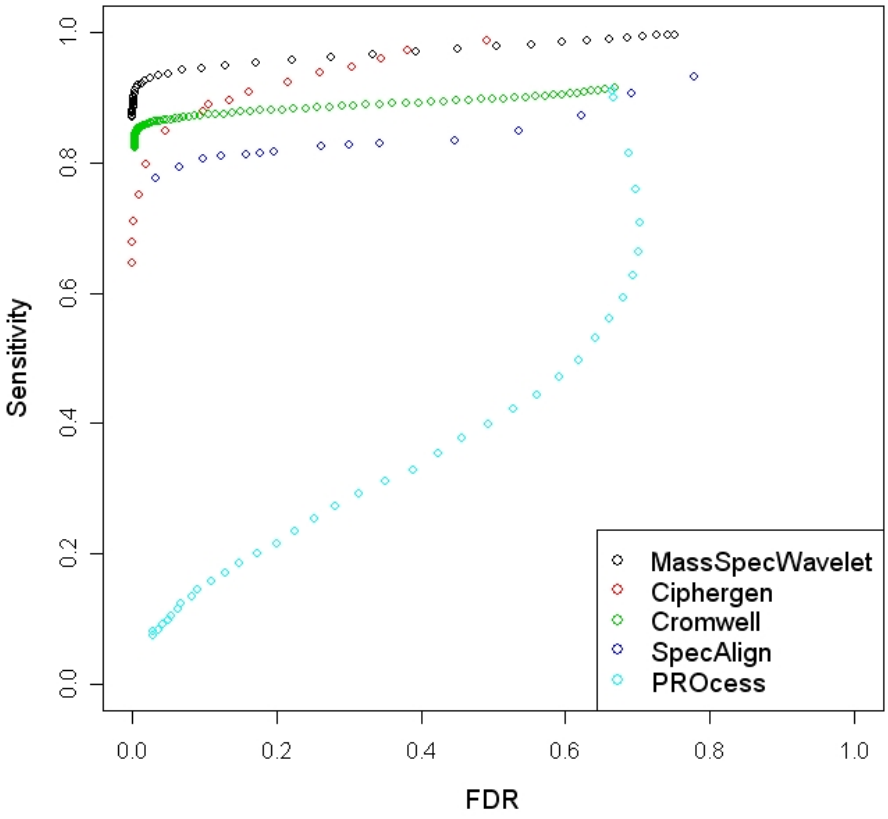


Figure 3: Mean FDR-sensitivity curve for peak detection algorithms. The mean was computed across 100 experiments for PROcess, MassSpecWavelet and Cromwell, and 10 for Ciphergen and SpecAlign. Each dot refers to the combinations of mean FDR and sensitivity corresponding to a given different signal-to-noise thresholds.

6 Table Summarizing Peak Counts

We are interested in comparing the count of missed peaks and falsely discovered peaks by algorithm. Those counts, however, are a function of the signal-to-noise threshold used to perform peak detection. To make sure that the results are comparable across the algorithms, we used signal-to-noise thresholds that produced the same level of FDR. Specifically, for each algorithm and simulated experiment we performed peak detection using a signal-to-noise threshold with FDR approximately equal to 0.1, and recorded the correspondent count of missed peaks as well as the count of falsely discovered peaks. We summarized the information across experiments by computing the mean and standard deviation of the peak counts obtained by algorithm. We repeated the procedure described above but considering signal-to-noise thresholds with FDR approximately equal to 0.2 and 0.3, respectively. The mean and standard deviation of the peak counts by algorithm and level of FDR appear in Table 1. For FDR 0.1, the software MassSpecWavelet produced a mean count of missed peaks equal to 8.08, the only algorithm different than MassSpecWavelet with a mean count of missed peaks smaller than 9 is Ciphergen when using signal-to-noise thresholds with FDR 0.3; however, its mean count of falsely discovered peaks is 50.50 as opposed to 15.94 produced by MassSpecWavelet. For all the considered FDR, the order of the algorithms according to the mean count of missed peaks is, in increasing order: MassSpecWavelet, Ciphergen, Cromwell, SpecAlign and PROcess.

Algorithm	FDR	Missed	Falsely Discovered
MassSpecWavelet	0.10 (0.01)	8.59 (2.94)	15.94 (2.53)
Ciphergen	0.10 (0.02)	18.10 (4.70)	11.10 (2.85)
Cromwell	0.10 (0.01)	19.30 (4.35)	13.33 (1.07)
SpecAlign	0.10 (0.01)	28.80 (5.39)	10.50 (0.85)
PROcess	0.10 (0.02)	124.68 (8.54)	2.02 (0.86)
MassSpecWavelet	0.20 (0.01)	6.73 (2.81)	38.37 (4.69)
Ciphergen	0.20 (0.02)	10.40 (4.77)	27.50 (3.34)
Cromwell	0.20 (0.01)	18.26 (4.28)	30.29 (2.18)
SpecAlign	0.20 (0.01)	26.00 (5.44)	25.00 (3.23)
PROcess	0.20 (0.02)	116.86 (10.57)	5.96 (2.08)
MassSpecWavelet	0.30 (0.02)	5.36 (2.49)	70.75 (9.02)
Ciphergen	0.30 (0.02)	8.30 (3.62)	50.50 (5.68)
Cromwell	0.30 (0.01)	17.31 (4.23)	52.39 (3.37)
SpecAlign	0.31 (0.02)	24.60 (5.06)	44.80 (3.77)
PROcess	0.30 (0.02)	108.43 (9.72)	13.33 (3.53)

Table 1: Mean and standard deviation, in parenthesis, of the peak counts of missed and falsely discovered peaks by algorithm and level of FDR. This results correspond to signal-to-noise thresholds with an approximate FDR equal to 0.1, 0.2 and 0.3, respectively, the column "FDR" shows the mean and standard deviation of the FDR's obtained across the analyzed experiments.

7 R Code for Generating Simulated Data

```
# This script shows how we simulated MS spectra for a given experiment.
# This script is not meant to be run, but includes a detailed description
# of how we simulated our data set.

# We will assume that the parameters for the experiment have been already
# determined. Remember that in our paper such parameters were chosen so that
# the generated spectra would have similar characteristics as
# our experimental data corresponding to fractionation 1 and low laser intensity.

# To summarize the characteristics of the experiment we used the following
# variables
# "mass.all"  # vector for the masses for the true peaks.
# "p" # prevalence for each mass in 'mass.all'
# "mu" # median for the intensity of each mass in 'mass.all' using the log2 scale
# "sig" # sd for the intensity of each mass in 'mass.all' using the log2 scale
# Here, we asssume that such information is provided in vectors
# with similar names.We also assume that the masses in the vector "mass.all"
# are sorted increasingly.
mass.all
mu
sig
p
# We also need a vector with the m/z range and specific values for which we will
# generate the respective intensities.
# For our simulated data, we obtained such vector, "mz.raw", from the
# experimental data.  Specifically, it had 60326 values in the m/z
# range from  83.25771 to 24998.48 DA, we called it
mz.raw

############ Generate the baseline  ##################################
baseline.log <- -log(5000:(length(mz.raw)+ 4999 )) +
                abs(min(-log(50:(length(mz.raw)+ 49 )))) + 6
baseline.final <- floor(exp(baseline.log) )
baseline.final <- matrix(baseline.final, ncol = 1)


############ Generate the signal  ####################################

# We add "extreme points" to the vector mass.all. They are useful to guarantee
# that the simulated signal will cover the m/z range specified by the
# vector 'mz.raw'.
mass.final <- c(1,mass.all,27000)

# Based on the prevalence, we determine which peaks will show up in
```

```
# the simulated spectra. We do that by considering a vector of 0's and
# 1's, a 1 indicates that the peak will be included.
# Furthermore, we use the vector 'mu' and 'sig' to generate
# the abundance that is associated to each mass
p.final <- p
ind= rep(0,length(p.final))
new.abun=rep(0,length(p.final))
for (j in 1:length(p.final))
{
ind[j]=rbinom(1,1,p.final[j])
new.abun[j]=rnorm(1,mu.final[j],sig.final[j])
}
# We add 1's for the extremes. We always want to include them but with a
# small abundance. We apply the inverse of the log2 transformation.
ind.final = c(1,ind,1)
new.final <- exp(c(0,new.abun,0)*log(2))

# Now, we use the simulation engine developed by Coombes et al.
# First, we load the required functions. Assuming the file is
# place in the working  directory.
source("ionFocusDelay.ssc")
# We use the following settings
mu0 <- 350
v0 <- 75
calvec <- c(1500,2000, 5000, 7000, 10000, 15000, 25000)

# The simulation engine produces:
# Experiment and Instrument parameters
expt <- expt.parameters(mu0,v0,4e-9)
default.instrument <- ifd.parameters()
# Generate the signal
mass <- mass.final[ind.final==1]
particles <- ceiling(new.final[ind.final ==1])
x <- simspec(mass,particles, expt)
# Get the m/z values
cal <- calibrator(calvec,expt, ifd=default.instrument)
y <- mass.spec(x, cal)
# Define final variables for:
# tof, intensity and mass
tof.sim =x$time
int.sim =x$intensity    # simulated intensity
mass.sim = y$mass       # simulated mass values

# Finally, we interpolate the data so that we can use the
# m/z values that correspond to our experimental data: 'mz.raw'
mz.interp <- mz.raw
```

```
int.interp <- 1:length(mz.raw)
for(i in 1:length(mz.raw)){
mz.value <- mz.raw[i]
# We look for the mz.value in the mass.sim vector
position <- sum( mass.sim <= mz.value)
# Interpolate the value using line
slope.int <- (int.sim[position + 1] - int.sim[position])/(mass.sim[position + 1]
                                        - mass.sim[position] )
int.interp[i] <- int.sim[position] + slope.int*(mz.value - mass.sim[position])
} # end for
int.interp <- floor(int.interp)

# Finally we notice that there is a difference between the scale
# of the intensities  in the simulated data and the experimental
# data of about 10. By adjusting for that factor we obtain the
# vector of intensities for the signal.
int.final <- int.interp*10
int.final <- as.matrix(int.final)

################## Generate the Noise  ############################
# As described in the manuscript, we randomly select one spectra
# from the experimental data and use it to generate the noise.
# Suppose that such spectrum is storaged in the vector
# "ms.raw"
# Such spectra was denoise using the Matlab script included in the
# Cromwell package. Using a threshold of 50 we obtained a deoised
# version of ms.raw and storaged it on the vector
# "ms.denoised"
# Therefore,the noise is given by the difference
noise.sample <- ms.raw - ms.denoised
# And we use that vector to simulate the component $\sigma(t)$
# as follows:
en <- length(noise.sample)
sd.noise.vec <- 1:en
# Compute standard deviation
for(i in (window.noise/2 + 1):(en-(window.noise/2 + 1))){
  sd.noise.vec[i] <- sd(noise.sample[(i-window.noise/2):(i+window.noise/2)])
}
sd.noise.vec[1:(window.noise/2+1)] <- sd.noise.vec[(window.noise/2 + 1)]
sd.noise.vec[(en-(window.noise/2+1)):en]<-sd.noise.vec[(en-(window.noise/2+1))]
# Extend the vector sd.noise.vec to the whole mz range.
sd.noise.vec.all <- 1:length(mz.raw)
start.noise <- which(mz.raw == noise.mz[1])
end.noise <- which(mz.raw == noise.mz[length(noise.mz)] )
sd.noise.vec.all[1:(start.noise-1)] <- sd.noise.vec[1]
sd.noise.vec.all[start.noise:end.noise] <- sd.noise.vec  # this is $\sigma(t)$
```

```
# Now we simulate the component $\gamma(t)$
# Suppose the parameters for the AR and MA part, respectively,
# are in the vectors
# ar_vec
# ma_vec
# Then the ARMA process was simulated as follows
noise.arma.vec <- arima.sim(list(ar=ar_vec,ma=ma_vec), n = length(mz.raw) )
# We rescale the ARMA process so it has variance 1
sd.ma <- sd(noise.ma.vec)
noise.arma.rescaled <- noise.ma.vec/sd.ma

# We combine the two components by multiplying the vectors
noise.ma.vec.scaled <- noise.arma.rescaled*sd.noise.vec.all
# Finally we make sure the all the values are positive.
noise.ma.vec.pos <- noise.ma.vec.scaled + max(abs(noise.ma.vec.scaled ))
noise.ma.final <- floor(noise.ma.vec.pos)  # this is $\epsilon(t)$

# Generate the final spectrum.
# Just add the components.
sim.all.arma <- int.final + noise.ma.final + baseline.final
```