Phase noise to Allan deviation conversion

Salim Elzwawi

Thursday, August 20, 2015

This document describes the code for computing the Allan deviation (up to 1 second) from a phase noise measurement. The conversion is performed using numerical integration of the spectral density of the normalized frequency fluctuation $S_y(f)$ [1].

$$\sigma_y^2(\tau) = \int_0^{f_h} S_y(f) \frac{\sin^4(\pi\tau f)}{(\pi\tau f)^2}\, df$$

The procedure is programmed using the *R* language and **RStudio** as IDE (freely available from http://www.rstudio.com). Alternatively **RGui** (from https://www.r-project.org) can also be used. For plotting the data, the *ggplot2* and *scales* packages are required [2]. The program reads and outputs files in csv format.

Loading the required libraries

```
library(ggplot2)
library(scales)
```

Oscillator nominal frequency f_o (see comment [3])

```
fo <- 10.949297E6  # Osc. nominal frequency in Hz
```

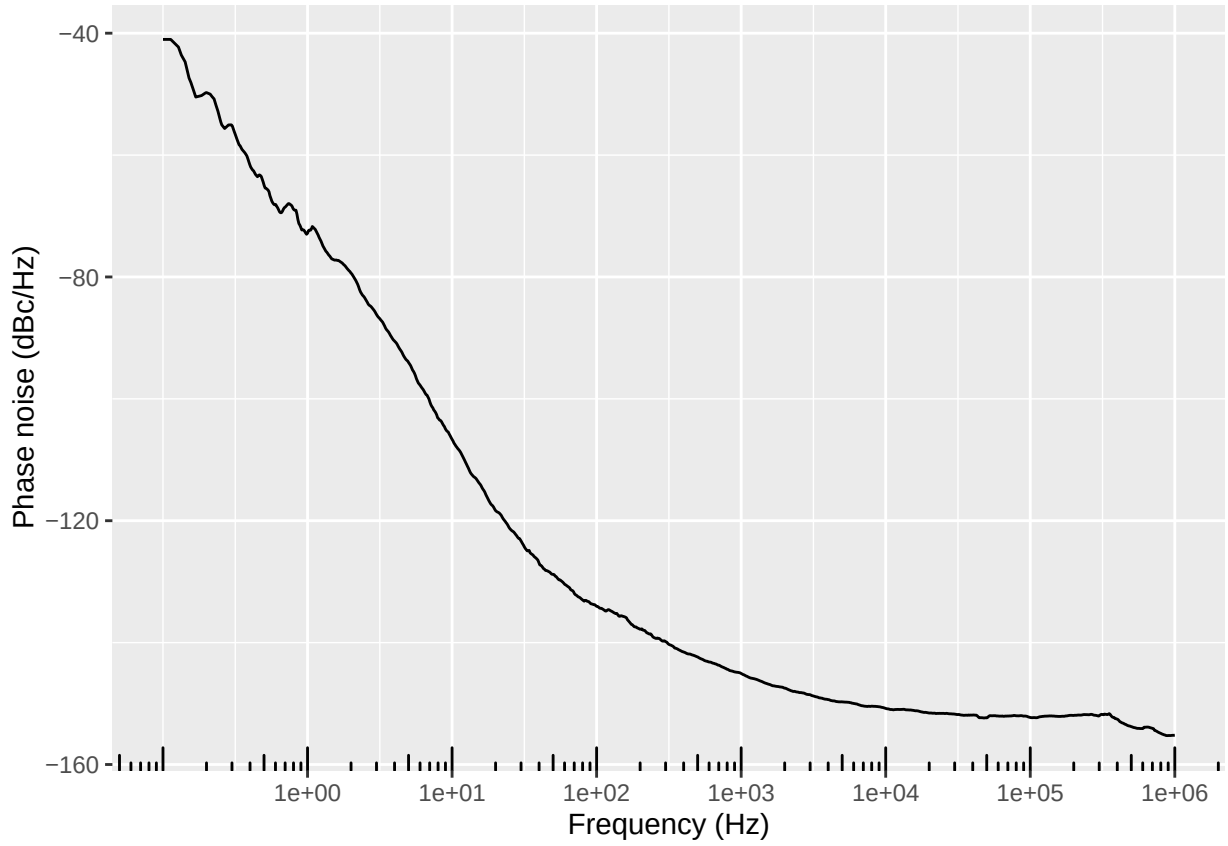
Reading and cleaning the data

The input file name is PN.csv where the first column is the frequency in Hz and the second is the measured single side band phase noise *L(f)* in dBc/Hz. The file is to be saved in the project's directory if using **RStudio** or in the working directory if using **RGui**.

```
Phasenoise <- read.csv("PN.csv")              # reading the file
Phasenoise <- Phasenoise[,1:2]                # Deleting any additional columns in the file
colnames(Phasenoise) <- c("Freq","PN")      # Assigning names to columns
```

Visualizing phase noise data

```
ggplot(Phasenoise, aes(x=Freq, y=PN)) + geom_line() +  scale_x_log10(breaks=10^(0:7)) +
  xlab("Frequency (Hz)") + ylab("Phase noise (dBc/Hz)") + annotation_logticks(sides="b")
```



Data transformation

The measured, logatithmic scale phase noise *L_(f)* is transformed to linear scale before processing, using

$$L(f) = 10^{L_{dBc(f)}/10}$$

.

```
Phasenoise$Lf <- 10^((Phasenoise$PN)/10)    # converting L(f) from dBz/Hz to normal scale
```

Calculating $S_y(f)$ from the measured phase noise *L(f)*

From the linear SSB phase noise trace, the spectral density of the fractional frequency is calculated using:

$$S_y(f) = 2\frac{f^2}{f_o^2}L(f)$$

.

```
Phasenoise$SyF <- (2*(Phasenoise$Freq/fo)^2)*Phasenoise$Lf    # L(f) to Sy(f) conversion
```

Discretizition: Calculating frequency intervals and mean *Sy(f)* in each interval

```
n <- nrow(Phasenoise)                 # number of data points
df <- NULL                            # Variable for number of freq. intervals
SyFm <- NULL                          # Variable for mean Sy(F) ineach interval
m <- n-1

for (i in 1:n-1) {
 df[i] <- Phasenoise$Freq[i+1] - Phasenoise$Freq[i]
}

for (j in 1:n-1) {
  SyFm[j] <- (Phasenoise$SyF[j+1] + Phasenoise$SyF[j])/2
}
```

Defining τ values at which Allan deviation is to be evaluated

```
tau <- c(seq(from=0.0001, to=0.0009, by=0.0001),
         seq(from=0.001, to=0.009, by=0.001),
         seq(from=0.01, to=0.09, by=0.01),
         seq(from=0.1, to=0.9, by=0.1),
         seq(from=1, to=9, by=1),
         seq(from=10, to=90, by=10),
         seq(from=100, to=900, by=100),
         seq(from=1000, to=9000, by=1000))
```

Calculating the integration muliplier

For each τ and f value, the integration multiplier

$$\frac{\sin^4(\pi\tau f)}{(\pi\tau f)^2}$$

is evaluated and the values are stored in a matrix. Each row holds the frequency dependent multiplier values for a specific τ.

```
multiplier <- matrix(1:(n*length(tau)),nrow=length(tau), ncol=n)       # Place holder for multiplier ma

for (j in 1:length(tau)) {
  for (i in 1:n) {
    multiplier[j,i] <- ((sin(pi/180*pi*tau[j]*Phasenoise$Freq[i]))^4)/((pi*tau[j]*Phasenoise$Freq[i])^2
  }
}
```

Integration

The Area under the curve over each frequency interval Δf and for each specific τ value is calculated and stored in a matrix (data frame).

```
integrand <- matrix(1:(m*length(tau)),nrow=length(tau), ncol=m)

for (j in 1:length(tau)) {
  for (i in 1:m) {
integrand[j,i] <- multiplier[j,i]*SyFm[i]*df[i]
}
}
```

Consequently, the discretized areas are summed ($\sum 2S_y(f).sin^4(\pi\tau f)/(\pi\tau f)^2\ \Delta f$):

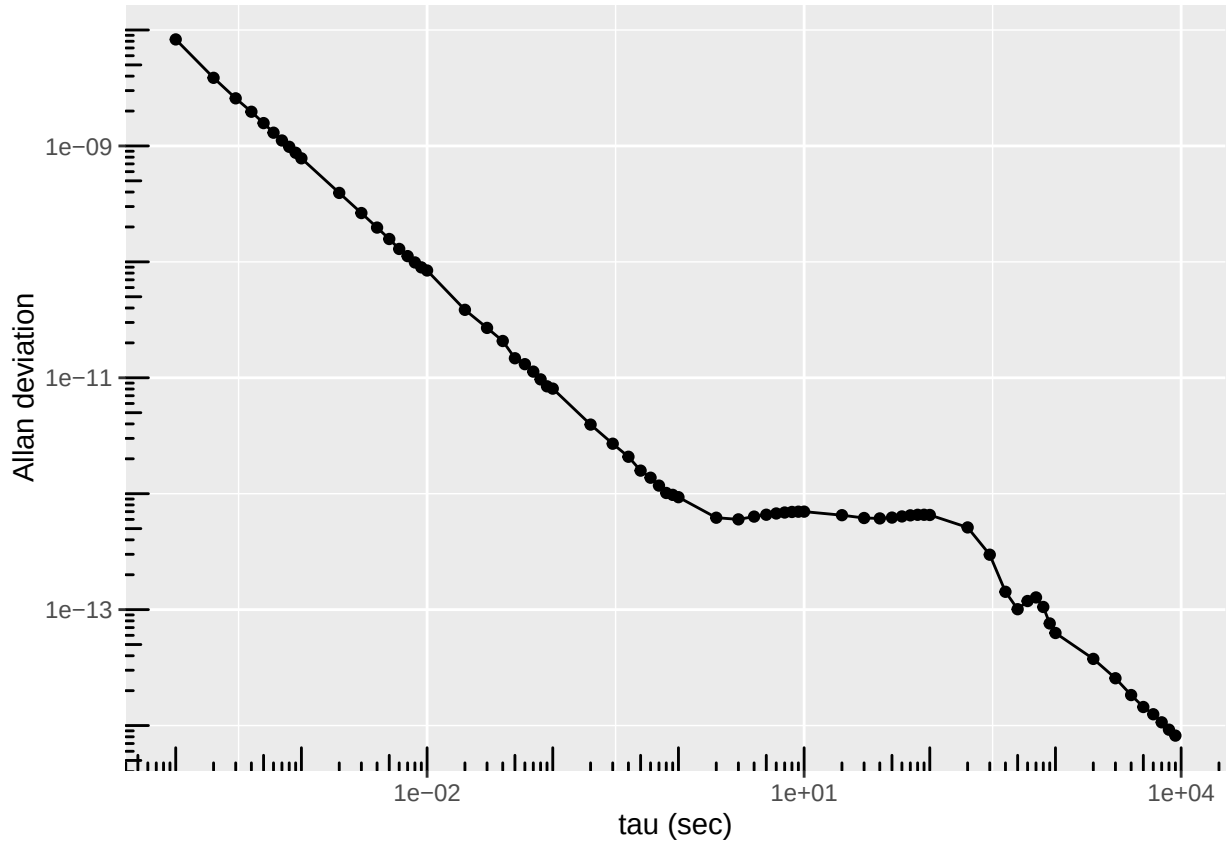
```
Allan_var <- 1:length(tau)                    # place holder for allan variance

for (j in 1:length(tau)) {
  Allan_var[j] <- 2*sum(integrand[j,])
}

Allan_dev <- sqrt(Allan_var)                    # Allan deviation
```

Plotting Allan deviation

```
data <- data.frame(cbind(tau,Allan_dev))
ggplot(data, aes(x=tau, y=Allan_dev)) + geom_line() + geom_point() +
  scale_x_log10() + scale_y_log10() + xlab("tau (sec)") + ylab("Allan deviation") +
  annotation_logticks(sides="bl")
```



Exporting Allan deviation data to a csv file

```
write.csv(data, file = "allan_dev.csv")
```

References and comments

[1] Characterization of Frequency and Phase Noise, Report 580 of the International Radio Consulative Committee (C.C.I.R), p. 142-425.
[2] Can be installed at command line by typing: *install.packages("ggplot2")* and *install.packages("scales")* .
[3] If the nominal frequency is to be input interactively in the console this code line should be replaced by the following line command:
fo <- as.numeric(readline("Input nom. oscillator frequency in Hz:\t"))