Some notes about detecting peaks in fast sampling CO2 concentration signals

M. Favaron

# Introduction

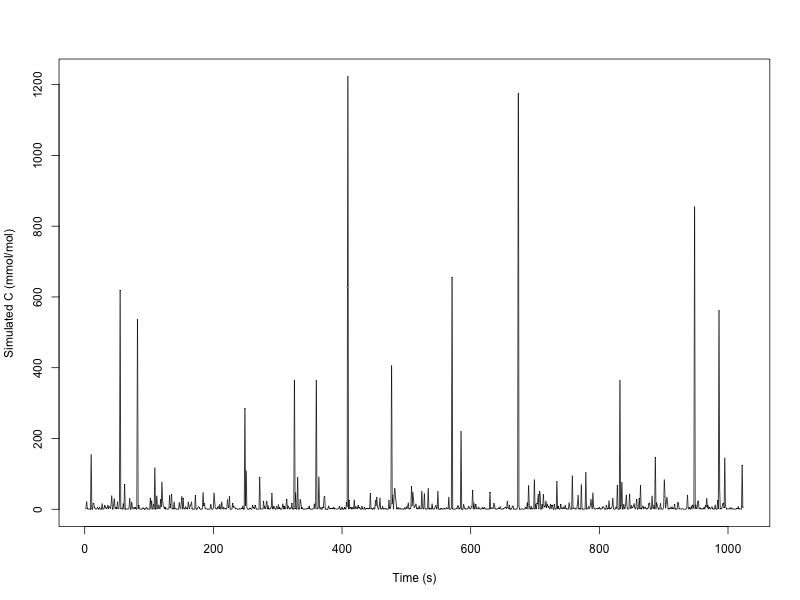
# CO2 concentration signals: known/assumed statistical properties, and generation method

As far as known from Fontanella campaign, we may assume CO2 concentration to be a mixture of a stationary, log-normally distributed background level, with intermittent peaks distribute randomly in time.

One possible model of such an observed behavior is a clipped log-normal distribution.

Another model is the explicit overlap of a log-normally distributed background, and single-sample peaks (in specified number), each having a random time position (with uniform distribution), and a level equal to a prescribed number of overall standard deviations computed on the log-normal part.

The following plot illustrates a realization of the latter type of process, with log-normal mean and standard deviations equal respectively to 0.5 and 2, and 3 peaks with amplitude equal to 5 times the log-nomal signal standard deviation.



Intermittency is, as we can see, well represented in simulated signal.

Also, the number of actual peaks is much larger than 3, due to the log-normal distribution’s fat tail.

The simulated series may be an oversimplification of actual measurements, but, at least visually reminds the CO2 signals not too far. A major departure from actual measurements may be in the latter’s lag-1 positive autocorrelation, which also regards peaks: usually more anomalous values do follow one another, with no “normality” gaps inside.

# The basic detector

Surfing the web I’ve found (on Stack Overflow) a simple detection algorithm.

The original is written in Matlab, but translating it to modern Fortran proved not that great deal, and here is the result (sorry so sloppy):

! Find peaks in a data vector, assuming a normal distribution. This program is the Fortran

! translation of "Smoothed z-score algo (very robust threshold algorithm)"; see

!

! https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-data

!

function FindPeaks\_Simple( &

rvX, &

lag, &

threshold, &

beta, &

signals, &

avgFilter, &

stdFilter &

) result(iRetCode)

! Routine arguments

real, dimension(:), intent(in) :: rvX

integer, intent(in) :: lag

real, intent(in) :: threshold

real, intent(in) :: beta

integer, dimension(:), allocatable, intent(out) :: signals

real, dimension(:), allocatable, intent(out) :: avgFilter

real, dimension(:), allocatable, intent(out) :: stdFilter

integer :: iRetCode

! Locals

integer :: n

integer :: i

real :: rAvg

real :: rStd

real :: rSumX

real :: rSumX2

real, dimension(:), allocatable :: filteredY

! Assume success (will falsify on failure)

iRetCode = 0

! Check something can be made

n = size(rvX)

if(n < 2) then

iRetCode = 1

return

end if

! Reserve workspace

if(allocated(signals)) deallocate(signals)

if(allocated(avgFilter)) deallocate(avgFilter)

if(allocated(stdFilter)) deallocate(stdFilter)

allocate(signals(n))

allocate(avgFilter(n))

allocate(stdFilter(n))

allocate(filteredY(n))

! Initialize data

signals = 0

filteredY(1:lag+1) = rvX(1:lag+1)

avgFilter(1:lag) = 0. ! Not used, really

stdFilter(1:lag) = 0. ! Not used, really

! Compute mean and standard deviation of signal beginning

rSumX = sum(rvX(1:lag+1))

rSumX2 = sum(rvX(1:lag+1)\*\*2)

rAvg = rSumX / (lag+1)

rStd = sqrt(rSumX2/(lag+1) - rAvg\*\*2)

avgFilter(lag+1) = rAvg

stdFilter(lag+1) = rStd

! Main loop: process all remaining time

do i = lag+2, n

! Locate peak

if(abs(rvX(i)-avgFilter(i-1)) > threshold\*stdFilter(i-1)) then

if(rvX(i) > avgFilter(i-1)) then

signals(i) = 1

else

signals(i) = -1

end if

filteredY(i) = beta\*rvX(i)+(1.-beta)\*filteredY(i-1)

else

signals(i) = 0

filteredY(i) = rvX(i)

end if

! Update comparison values

rSumX = sum(filteredY(i-lag:i))

rSumX2 = sum(filteredY(i-lag:i)\*\*2)

rAvg = rSumX / (lag+1)

rStd = sqrt(rSumX2/(lag+1) - rAvg\*\*2)

avgFilter(i) = rAvg

stdFilter(i) = rStd

end do

end function FindPeaks\_Simple

(Sorry I’ve not given an exact attribution; I will find it sooner or later: anyway, the algorithm seems to be one of the many “simple ideas” living around, extensively used in practice since many years, but never really published; sort of a shared trick.)

The idea behind the detector is of a locally stationary signal, on which an intermittent process is overlapped providing peaks, both positive and negative. The signal is then scanned in time-increasing order, and a local standard deviation computed using a local smoother with form

where , and the original signal.