Some notes about detecting peaks in fast sampling CO2 concentration signals

M. Favaron

# Introduction

This note elaborates on a new peak detection algorithm, trying to reproduce results from the *Peak Sgamator 2.0,* presented by Roberto Nava in his thesis.

The algorithm is very specific, in the sense it is tailored specifically to CO2 data as found for example during the Fontanella campaign.

Its intended usefulness, given the *Peak Sgamator 2.0* already exists and works well, is in a faster execution time, with the possibility to apply it to other, similar, data coming in huge masses from data-driven pollutant dispersion model application experiments. Possibly, tolerating a lower detection power.

Requirements for the new peak detection algorithm include:

* Robustness
* Efficiency, in terms of execution time
* Ease of use and understanding
* Fitness for use with time series sharing statistical properties with those of Fontanella campaign CO2 concentration data.

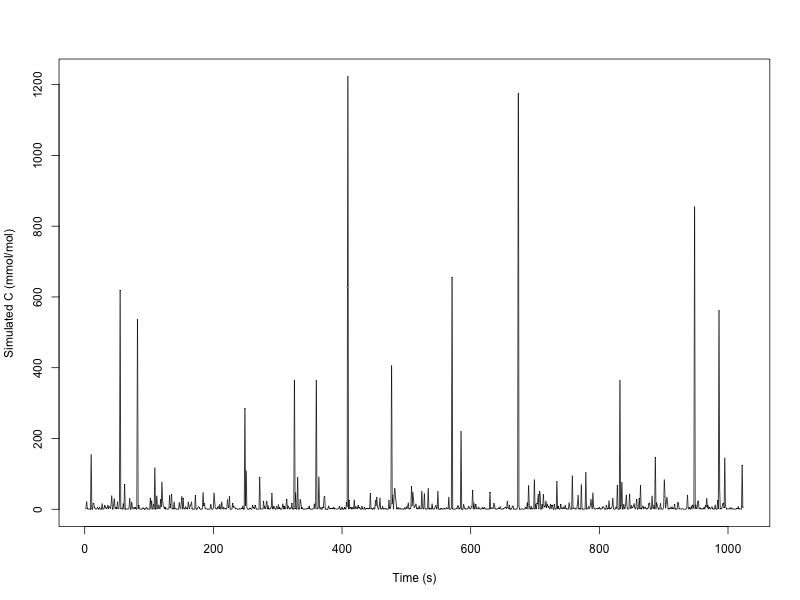
# CO2 concentration signals: known/assumed statistical properties, and generation method

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-normal 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 smoothed z-score detector

A popular peak detection algorithm, named “smoothed z-score”, first presented in M. C. Catalbas, T. Cegovnik, J. Sodnik and A. Gulten (2017). [**Driver fatigue detection based on saccadic eye movements**](https://ieeexplore.ieee.org/document/8266142/), 10th International Conference on Electrical and Electronics Engineering (ELECO), pp. 913-917, and described on StackOverflow with code, at URL <https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-data>.

The code presented here is a modern Fortran translation of original Matlab code:

! 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

We use this algorithm as starting point, given its closeness to the intuitive knowledge we have of the CO2 concentration series, of a locally stationary, on which an intermittent process is overlapped providing peaks, both positive and negative.

The smoothed z-score detector is based on the assumption that any value departing a significant number (typically 3) of standard deviations from the local mean can be considered an “outlier”. This is akin to say the data are locally distribute following a normal law, which ensures slim tails on both ends of density functions.

This is not our case, the log-normal distribution showing indeed a very fat upper tail. But, let’s ignore for the moment the depths of this realization, and proceed as if this assumption is satisfied.

The algorithm scans the input signal in time-increasing order, and a local moving mean and standard deviation are computed initially, using a fixed lag specified by the user, on the less recent part of the signal; notice peaks are not detected in this initial part, assuming the signal contains an initial “clean” part on which we users have full information, and no peaks can be found.

Moving average and standard deviation are then propagated to increasing time values, while detection of outliers is performed.

The rule actually used to perform detection is by comparing the signal value at index , that is, , with a “filtered” version of signal at preceding index, let’s name it . If

where is the moving standard deviation, then is labeled as an outlier, otherwise it is left unchanged. Positive and negative peaks are distinguished by checking whether is, respectively, larger or smaller than .

The filtered signal, , is defined as coinciding with the original signal in case this is not an outlier, otherwise it is updated using a linear smoother having form

where .

# Practical implementation aspects of the smoothed z-score detector

Modern Fortran has no statements to compute moving averages and standard deviations directly, but these may be easily implemented using existing instructions.

The way I’ve followed makes use of SIMD execution path available on most to-date processors, as supported by modern Fortran dedicated “whole array” statements.

I have *not* (for the moment) taken advantage of many-core CPUs like the Tegra I’ve on my desk (at home). This, for simplicity and to constrain development to a “simple and potentially slow” architecture, which encourages leveraging my neural circuitry finding decent-lightweight-readable-understandable code blocks.

# Issues of the basic smoothed z-score detector with CO2 concentration data

As I mentioned, an unspoken assumption of the basic smoothed z-score detector is the signal is normally distributed, at least locally.

By this, I mean:

1. The signal is an outcome of an almost second-order stationary random process having the form . Here indicates a log-normal random variable with log-normal mean and log-normal standard deviation . The symbol I’ve used should convey the meaning that if then are *independent* random variables – or at least, this is what I’m imagining of ☺. Don’t still know whether requiring statistical independence is paramount, or a may-safely-ignore: have still to think seriously on this – anyway, if you have an idea and/or experience already, I’d really glad learning it: maybe, it solves my philosophical problem in advance ☺.
2. change “slowly” with index . That is, they are allowed to change (as they would in any natural process), but, they do so quietly that the best predictor of are just, respectively, . (Does this requirement contradict my independence assumption? Have to check…)