[**3dREMLfit**](https://afni.nimh.nih.gov/pub/dist/doc/program_help/3dREMLfit.html)[**Notes**](https://docs.google.com/document/d/1zpujpZYuleB7HuIFjb2vC4sYXG5M97hJ655ceAj4vE0/edit?usp=sharing)

**RW Cox**

**[Aug 2019]**

A question came up:

*Is the way AFNI’s time series regression programs, 3dREMLfit and 3dDeconvolve, deal with time point censoring equivalent to the way that other programs (e.g., SPM, FSL) deal with such censoring*?

To understand the alternatives, consider the regression model for a single time series data vector **z** with *N* time points and *M* regressor (model) component:

**z = X β + ε**

where **X** is an *N*x*M* matrix, **β** is an *M*-vector, and **ε** is the *N*-vector of residuals (to be made as “small” as possible when solving for **β**). Suppose that element *zi* is to be censored out of the analysis for whatever silly reason (e.g., too much head motion).

* Row removal (AFNI approach): Remove the *i*th element from **z**and correspondingly remove the *i*th row from the matrix **X** (since it is that row which contains the model for *zi*).
* Column augmentation (SPM and FSL approach): Add a column to **X** that is all zero except for a single 1 in the *i*th element. The idea is that this extra regression component will exactly fit the data point in *zi* and so the β value for this extra component will be *zi*, and all the other components of beta will be devoted to fitting the “real” data in the rest of vector **z**. One column is added for each index *i* which is to be censored.

The advantages of the row removal method are (a) that it shrinks the **X** matrix, reducing the computational load, and (b) that it exactly accounts for the non-use of *zi* since the offending value is omitted entirely from the analysis. The disadvantage of the row removal method is that it breaks the regular time spacing of the data. The column augmentation method has the inverse characteristics.

If some non-AFNI pipeline wants to use 3dREMLfit, the developers are likely to want to censor using the column augmentation method, since that is what most neuroscience people are familiar with. The question arose in my mind about whether the two approaches give the same results in 3dREMLfit.

Regular time spacing is not important if ordinary least squares (OLSQ) is used to fit **β**. However, if a temporal correlation matrix **R** needs to be estimated from the data, and then applied to “pre-whiten” the problem, then the temporal spacing needs to be properly allowed for when model-fitting **R** and **β** together. Some algorithms for fitting models for **R** are much simpler with regular (unbroken Δ*t*=TR) time spacing; for example, the Yule-Walker equations for AR(p) models, or even more obviously, DFT-based approaches. AFNI’s 3dREMLfit was built to avoid the requirement for a regular TR, by using a *voxelwise* ARMA(1,1) model – see [3dREMLfit\_mathnotes](https://drive.google.com/open?id=1tD51_w9_lfVWLLg-Pt0hl57wE81s_Imc) for the details. Non-contiguous segments of data (“runs”) can be catenated and analyzed together, as well as allowing for censoring time points where bad things happened. The voxelwise computation of the ARMA(1,1) autocorrelation prewhitening model is meant to allow for different types of temporal correlation structure in different image regions and tissue types. (Is this useful? Opinions vary.)

Aside – Solution methods:

The OLSQ solution is **β** = [**XTX**]-1 **X**T **z**. The generalized least squares (GLSQ, or “pre-whitening” solution) is derived by pre-multiplying the matrix-vector equation by a symmetric matrix **W** such that **W**2=**R**-1, where **R** is the temporal autocorrelation matrix. Then the equation becomes **Wz = WXβ + Wε**, and under the assumption that E[**εε**T] = σ2**R**, E[**Wεε**T**W**] = σ2**I**, and so *this* equation is validly/optimally ([BLUE)](https://en.wikipedia.org/wiki/Gauss%E2%80%93Markov_theorem) solved by OLSQ, giving instead **β** = [**XTR**-1**X**]-1 **X**T**R**-1**z**. The actual calculations in 3dREMLfit are a little more intricate for the sake of efficiency and require estimating **R** (using **ε**) and **β** together to be self consistent – that’s the point of [REML](https://en.wikipedia.org/wiki/Restricted_maximum_likelihood). See the aforementioned math notes for more such “fun”.

Trying it out:

The simplest way to deal with the initial question was to run the program both ways. To aid in doing this, I modified 3dDeconvolve to allow the user (me) to generate the matrix file for 3dREMLfit with censoring handled by column augmentation, in addition to the matrix created with row removal. (3dDeconvolve creates the matrix file for 3dREMLfit, from the user’s time series model components.) 3dREMLfit could then be run twice, once with each censoring method. I used a study which had already been run with afni\_proc.py, and started from that results directory.

Script 1: create the two .xmat.1D files (**X** matrices) in 3dDeconvolve

This command was edited from the script generated by afni\_proc.py

 3dDeconvolve

 -input pb01.sub-10697.r01.tshift+orig.HEAD

 -censor censor\_sub-10697\_combined\_2.1D

 -polort 4 -num\_stimts 8

 -stim\_times 1 stimuli/pamenc.times.CONTROL.txt 'BLOCK(2)'

 -stim\_label 1 CONTROL

 -stim\_times 2 stimuli/pamenc.times.TASK.txt 'BLOCK(4)'

 -stim\_label 2 TASK

 -stim\_file 3 'motion\_demean.1D[0]' -stim\_base 3 -stim\_label 3 roll

 -stim\_file 4 'motion\_demean.1D[1]' -stim\_base 4 -stim\_label 4 pitch

 -stim\_file 5 'motion\_demean.1D[2]' -stim\_base 5 -stim\_label 5 yaw

 -stim\_file 6 'motion\_demean.1D[3]' -stim\_base 6 -stim\_label 6 dS

 -stim\_file 7 'motion\_demean.1D[4]' -stim\_base 7 -stim\_label 7 dL

 -stim\_file 8 'motion\_demean.1D[5]' -stim\_base 8 -stim\_label 8 dP

 **-x1D XQ.xmat.1D**

 **-x1D\_regcensored XQ.regcensor.xmat.1D**

 -x1D\_stop

The two bolded lines output the two .xmat.1D files for input to 3dREMLfit (row-censored and column-augmented).

Script 2: run 3dREMLfit twice, using the two matrix files, on the time-shifted input data

 3dREMLfit

 -matrix **XQ.xmat.1D**

 -input pb01.sub-10697.r01.tshift+orig.HEAD

 -fout -tout -verb -Grid 5

 -Rbuck QQstats.sub-10697\_REML

 -Rvar QQstats.sub-10697\_REMLvar

 3dREMLfit

 -matrix **XQ.regcensor.xmat.1D**

 -input pb01.sub-10697.r01.tshift+orig.HEAD

 -fout -tout -verb -Grid 5

 -Rbuck QQRstats.sub-10697\_REML

 -Rvar QQRstats.sub-10697\_REMLvar

Then the stats datasets from the two runs can be compared (visually and by subtraction).

It turned out that the results were exactly the same, *except* in a few voxels – about 10 out of more than 300,000. This outcome was peculiar, but a few moments of inspection showed that the differences occurred precisely in those (non-brain) voxels which were identically 0 except at one or more of the censored time points. When I realized this, the explanation was obvious.

With row removal, the censored data points are fully removed from the analysis. In these exceptional voxels, that removal resulted in the data time series **z** being identically zero. When this happens, 3dREMLfit skips all analysis in that voxel, and fills in the corresponding voxel results as being all zeros. In column augmentation, normal linear solving will take place, as the data is not exactly zero. In exact arithmetic solution, the augmented columns would zero out the nonzero elements of **z**; however, with inexact computer arithmetic, the linear regression leaves a nonzero residual vector **ε**, which in turn is analyzed for the ARMA(1,1) parameters, and then **β** and all the voxel-level statistics are calculated. **Question answered**: *3dREMLfit works the same for either censoring method***.**

But … there’s always a “but”:

In looking at the results from Script 2, I saw something peculiar:



This is an image of the λ parameter = correlation at lag=1 from the ARMA(1,1) model. A little thought shows that this is due to the time-shifting operation. By default, the necessary temporal interpolation is done with 5th order (quintic) Lagrange polynomials, which uses ±2 points in time for interpolation (via AFNI program 3dTshift). I re-ran the time shifting with the various options for interpolation method, and found that the Fourier (FFT) interpolation completely eliminated the stripes. To further investigate, I added ±5 and ±9 point weighted sinc interpolation methods to 3dTshift. The striping artifact is reduced with the “wsinc5” method, and almost completely gone with the “wsinc9” method.

How important is this artifact? If one is using 3dREMLfit, then the voxelwise ARMA(1,1) model should deal with it. The alternative cure, using a broader-based temporal interpolation, gets rid of the artifact, but has the downside that more distant time points will leak into the interpolated output values. In turn, this could bias the **β** estimation – probably not much, but that is another line for investigation.

Conclusion: [**The Rabbit Hole Has No Bottom**](https://en.wikipedia.org/wiki/Red_Queen%27s_race)