**About TDR data downloaded from VORTEX-SE 2018 archive**

The TDR data can be processed with a combination of an automated Python script, that Conrad Ziegler is finalizing, and semi-automated editing with the NCAR/EOL Solo software.



In the TDR data files, the acronym “MA” means “MASTER” which refers to the forward-pointing antenna/radar.  The acronym “SL” means “SLAVE” which refers to the aft-pointing antenna/radar.  The file name of the tarred, gzip-compressed MA and SL data for a single IOP includes the characters “product\_raw”, which is the file type that you need download. Ignore the other file types (if any).

**CIMMS Radar Quality Control Software**

First things first, it needs to locate a ***Py-ART*** source file in the install directory and replace it with the version attached. The ‘simulated\_vel.py’ file can properly handle TDR data when creating simulated radial velocities from an input sounding. It can be found in the following location if you installed Py-ART in your root conda environment:

*<path to anaconda3 directory>*/lib/python3.7/site-packages/pyart/util/simulated\_vel.py

or here if you installed into an environment:

*<path to anaconda3 directory>*/envs/*<environment name>*/lib/python3.7/site-packages/pyart/util/simulated\_vel.py

Then, you need to transfer TDR raw data to Dorade form by Radx. Because Soloii/solo3 does not support TDR raw data. Next, you’ll need to do some preliminary editing of the TDR data in ***soloii/solo3***. The first step is to compute ground-relative velocities. In a solo editing window, issue the following commands for each sweep (press the ‘First Sweep’ and ‘Last Sweep’ buttons at the bottom of the window to apply these commands to every sweep file in the current directory:

duplicate VEL in VG

remove-aircraft-motion in VG

Once you have VG computed, you will need to follow the directions in the attached ‘preEdit\_CRQC\_README.pdf’ (showing below) which will guide you through using the surface removal routine in solo. I suspect that you will be fine using an “optimal-beamwidth” of 3.0 degrees, but feel free to experiment with this. The goal with this step is to remove the majority of the echoes near and below ground-level. You will notice in the example screenshot that I copied ‘DBZ’ and ‘VG’ to ‘D0’, and ‘V0’, respectively. Feel free to use whichever names you prefer, just be sure to adjust the input YAML (\*.yml) file accordingly.



Once you have completed the preliminary editing in solo, you will need to convert the dorade files to CfRadial files using RadxConvert. The command to do this is (if you’re currently in the same directory as the dorade files):

RadxConvert -f swp.\* -outdir <path/to/output-dir>

The YAML file (‘input\_radQC\_example.yml’) is how you will control which QC routines to apply to your data along with the parameters associated with each. I have done my best to include plenty of comments in the input file, but please let me know if anything remains unclear. The YAML file I have included should require very little editing, as the default parameters *should* work fairly well for VSE 2018 TDR data. As is, the following routines will be applied to the data:

* rmvSpokes: Loop through all input files and check for any “spokes” (e.g., rays or parts of rays with notably higher reflectivity than in adjacent rays). In addition to spoke removal, this step also tends to do a good job “defreckling” the data (removing gates or small collections of gates with higher reflectivity than neighboring gates). This step will produce output CfRadial files ending in “\*\_preCorr.nc”.
* agg: Loop through all files within each volume and generate a single “aggCfrad.\*” file for each volume. This aggregate file contains several statistical quantities used later on for surface identification and masking (e.g., volume median reflectivity at each gate).
* One final loop over all the files where each of the following is executed. An output CfRadial file will be saved (ending in “\*\_corr.nc”) and RadxConvert will be called to make a dorade copy of each (unless you set ‘frmat’ to ‘None’ on line 37):
  + makeSimVel: Generate a simulated radial velocity field based on an input environmental sounding. This step is optional, but is currently required if you want to utilize the 4DD velocity dealiasing routine and/or the clear air velocity checks later on. This currently only supports soundings in the NCAR EOL Sounding Composite format (see the example file attached – “example\_sounding.txt”)
  + deal: Dealias radial velocities. There are a few options here, but if you run it as-is velocities will be dealiased using the aforementioned 4DD method.
  + dual: Identify and correct any dual-PRF processor mistakes.
  + sfcRmv: Identify and mask gates potentially affected by the intersection of the beam with the surface.
  + g2dFilt: Generate a smoothed reflectivity field (optionally any other field) which is then used to help mitigate noise
  + clrVelChk: Compare observed radial velocity field to a simulated radial velocity field generated from an input environmental sounding to identify and mask errant observations in areas of low reflectivity (i.e., clear air).

To get started, you should only need to check and adjust the following parameters (you can experiment with adjusting anything else as you see fit):

radar\_folder (line 25) – Path to input (edited) CfRadial files

velIn/refIn/velOut/refOut (lines 41, 42, 45, 46) – Names of input and output velocity and reflectivity variables

vols (line 122+) – Used for specifying start and end times for each P-3 dual-Doppler leg, along with the location of the storm relative to the P-3 direction of flight

soundF (line 146) – Path to input representative environmental sounding

Once you’re ready to run the code, issue the following command from within the directory containing the QC code:

python run\_radQC.py input\_radQC.yml

**Single-Radar Objective Analysis Code**

It needs the Gfortran compiler and a C compiler on machine.

Here I attach.  Please review this Makefile, which indicates that in addition to Gfortran and C compilers, your machine needs to have netcdf libraries installed.

Firstly, download the single-radar objective analysis code: *mpbarnes.tar.gz*. A Makefile is included for single radar objective code that interpolates radar data to a regular Cartesian grid at chosen analysis times. You can either edit that accordingly or use the version of Gfortran and Gcc you already have. Once you have the directory unzipped/untarred, you can compile the code by executing the following command within that directory:

make -f <name-of-your-makefile>

It is normal to see a number of *warnings* during compilation (these are mostly due to differences in Fortran versions, and can generally be safely ignored). If you end up with an executable (“x.oban\_multi\_v3\_gf”) at the end of compilation, you should be good to go. Regardless, be sure to review the make output to ensure there were no errors.

Then run the code by command:

./x.oban\_multi\_v3 ob\_20180413\_tfor.input >&! oban.output &

***We suggest you take some time to review the file and the embedded comments, as well as to review the code itself (especially “oban\_multi\_v3.f”) to see how the input parameters are used.***