



– *Technical Report* –

Registration No.

OPSEC#:4793

Date of Report

2020-11-17

VEHICLE SHAKER (N-POST) DRIVE FILES – FROM COURSE PROFILE TO CHANNEL TIME HISTORIES

W. Bylsma

Structures

U.S. Army Combat Capabilities Development Command (DEVCOM)

Ground Vehicle Systems Center (GVSC)

Detroit Arsenal

Modeling, Simulation & Software

ATTN: FCDD-GVR-MSS/MS 157

6501 East 11 Mile Road

Warren, Michigan 48397

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited.

OPSEC#: 4793

Disclaimer: Reference herein to any specific commercial company, product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the Department of the Army (DoA). The opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or the DoA, and shall not be used for advertising or product endorsement purposes.

U.S. Army Combat Capabilities Development
Command Ground Vehicle Systems Center
Detroit Arsenal
Warren, Michigan 48397-5000

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.				
1. REPORT DATE (DD-MM-YYYY) 11-17-2020		2. REPORT TYPE TECHNICAL		3. DATES COVERED (From - To) 2007-2020
VEHICLE SHAKER (N-POST) DRIVE FILES - FROM COURSE PROFILE TO CHANNEL TIME HISTORIES		5a. CONTRACT NUMBER		
		5b. GRANT NUMBER		
		5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) WESLEY BYLSMA		5d. PROJECT NUMBER		
		5e. TASK NUMBER		
		5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) STRUCTURES-US ARMY DEVCOM/GVSC ATTN: FCDD-GVR-MSS/MS157 6501 E 11 MILE RD WARREN, MI 48397-5000		8. PERFORMING ORGANIZATION REPORT NUMBER OPSEC#: 4793		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)		
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION / AVAILABILITY STATEMENT Distribution Statement A: Approved for public release; distribution is unlimited.				
13. SUPPLEMENTARY NOTES				
14. ABSTRACT This report documents the steps required to create vehicle N-Post shaker drive files for use in a laboratory setting for dynamic vehicle testing. Consideration is given to preparing course profile data into a standard format and its use in generating actuator position time history data. These steps include unit conversion, detrending, mean removal, travel and velocity limit checks, time versus spatial frequency comparison, peak scaling, tire patch filtering, and zero column insertion which are encapsulated in two Powershell scripts. The Course Profile Conversion (CPC) PowerShell script reads course profile data in multiple file formats and outputs detrended single track data (left and/or right) after unit conversion. The Drive File Creation (DFC) PowerShell script creates time history channel data for each set of parameters specified (e.g. vehicle parameters, speeds, course profile, processing options) that define simulator actuator positions versus time for laboratory testing. The time histories are based on detrended left and right track course profiles output from CPC.				
15. SUBJECT TERMS vehicle shaker, N-Post, drive files, course profile conversion, channel time histories, actuator position time history data				
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Unclassified	18. NUMBER OF PAGES 71
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified		
			19a. NAME OF RESPONSIBLE PERSON Wesley Bylsma	
			19b. TELEPHONE NUMBER (include area code) 586-282-4104	

CONTENTS

1.0 INTRODUCTION	1
2.0 COURSE PROFILE CONVERSION.....	3
2.1 COURSE PROFILE FILE FORMATS	3
2.2 UNIT CONVERSION.....	5
2.3 DETRENDING	5
2.4 OUTPUT FILE FORMAT	7
2.5 EXECUTION	7
3.0 DRIVE FILE CREATION	10
3.1 PARAMETERS	10
3.1.1 Comments	10
3.1.2 Track Data Input.....	10
3.1.3 Output Channels.....	10
3.1.4 Channel Delay	11
3.1.5 Speed	12
3.1.6 Sample Frequency	12
3.1.7 Ramp Time	12
3.1.8 Wait Time	13
3.1.9 Travel Limit.....	13
3.1.10 Velocity Limit	14
3.1.11 Tire Patch Filter	15
3.1.12 Channel Data Output Name	16
3.1.13 Unit Length Symbol	18
3.1.14 Unit Velocity Symbol	18
3.1.15 Velocity to Length Conversion Factor	18
3.1.16 Scale Peak	19
3.1.17 Insert Columns	19
3.2 OUTPUT FILE FORMAT	20
3.3 EXECUTION	20
3.4 DRIVE FILE CREATION PROCESS SUMMARY	22
APPENDIX A.1 – CPC TESTS.....	27
APPENDIX A.2 – DFC TESTS.....	31
APPENDIX A.3 – CPC SCRIPT	45
APPENDIX A.4 – DFC SCRIPT	55
APPENDIX A.5 – POWERSHELL NOTES	71

FIGURES

FIGURE 1 – GVSC N-POST TEST EQUIPMENT (SEE [1]).....	1
FIGURE 2 – PROFILE SECTION.....	3
FIGURE 3A – TYPE 1 FILE FORMAT (MULTIPLE ROWS)	3
FIGURE 3B – TYPE 1 FILE FORMAT (SINGLE COLUMN)	4
FIGURE 4 – TYPE 2 FILE FORMAT.....	4
FIGURE 5 – TYPE 3 FILE FORMAT.....	4
FIGURE 6 – DOUBLE-SIDED EXPONENTIAL FILTER [13]	6

FIGURE 7 – COURSE OUTPUT LENGTH W/O PADDING	7
FIGURE 8 – EXAMPLE CPC EXECUTION WITH PARAMETERS	8
FIGURE 9 – CHANNEL OUTPUT 2-AXLE EXAMPLE	11
FIGURE 10A – CHANNEL DELAY 2-AXLE EXAMPLE	11
FIGURE 10B – CHANNEL DELAY 3-AXLE EXAMPLE	12
FIGURE 11– RAMP TIME	13
FIGURE 12– WAIT TIME.....	13
FIGURE 13– TRAVEL LIMIT	14
FIGURE 14– VELOCITY LIMIT	14
FIGURE 15– TIRE PATCH FILTER PARAMETERS.....	15
FIGURE 16– HANN FUNCTION [15].....	16
FIGURE 17– SCALE PEAK EXAMPLE.....	19
FIGURE 18 – DFC FILE FORMAT.....	20
FIGURE 19 – EXAMPLE DFC EXECUTION WITH “.DEF” PARAMETER FILE	21
FIGURE 20 – CONVERSION FROM DISPLACEMENT TO TIME HISTORY PROCESS	24
FIGURE A1.1 – COMPARISON OF DETRENDING (W/O PADDING) WITH MATLAB VS CPC	27
FIGURE A1.2 – COMPARISON OF “IN TO M” UNITS W/O DETRENDING	28
FIGURE A1.3 – COMPARISON OF “FT TO IN” UNITS W/O DETRENDING.....	29
FIGURE A1.4 – COMPARISON OF LEFT/RIGHT TRACK “FT TO MM” UNITS W/O DETRENDING	30
FIGURE A2.1 – COMPARISON HMMWV, CHURCHVILE B SMOOTH, 10 MPH.....	31
FIGURE A2.2 – COMPARISON HMMWV, CHURCHVILE B SMOOTH, 15 MPH.....	32
FIGURE A2.3 – COMPARISON HMMWV, CHURCHVILE B SMOOTH, 20 MPH.....	33
FIGURE A2.4 – COMPARISON HMMWV, CHURCHVILE B SMOOTH, 25 MPH.....	34
FIGURE A2.5 – COMPARISON M1085, BELGIAN BLOCK, 15 MPH	35
FIGURE A2.6 – COMPARISON M1085, BELGIAN BLOCK, 20 MPH	36
FIGURE A2.7 – COMPARISON M1085, BELGIAN BLOCK, 25 MPH	37
FIGURE A2.8 – COMPARISON M1085, BELGIAN BLOCK, 30 MPH	38
FIGURE A2.9 – COMPARISON M1085, PERRYMAN 3, 10 MPH.....	39
FIGURE A2.10 – COMPARISON M1085, PERRYMAN 3, 15 MPH.....	40
FIGURE A2.11 – COMPARISON M1085, PERRYMAN 3, 20 MPH.....	41
FIGURE A2.12 – COMPARISON M1085, PERRYMAN 3, 25 MPH.....	42
FIGURE A2.13 – COMPARISON BETWEEN PEAK SCALING FOR M1085_P3_TF_025.....	43
FIGURE A2.14 – ZOOMED COMPARISON BETWEEN PEAK SCALING FOR M1085_P3_TF_025..	43
FIGURE A2.15 – EXAMPLE #INSERT 0 1 0 3 0 0 0	44

TABLES

TABLE 1 – REVISION TIMELINE.....	2
TABLE 2 – FILTER DISTANCE (ONE SIDE)	6
TABLE 3 – CPC TYPE 1 OR 2 FILE FORMAT PARAMETERS.....	7
TABLE 4 – CPC TYPE 3 FILE FORMAT PARAMETERS	8
TABLE 5 – HANN FUNCTION CALCULATION.....	16
TABLE 6 – DFC PARAMETERS	20
TABLE 7 – SPATIAL SAMPLING DATA	24
TABLE 8 – SPEED SAMPLING DATA.....	25

VEHICLE SHAKER (N-POST) DRIVE FILES – FROM COURSE PROFILE TO CHANNEL TIME HISTORIES

TARDEC Technical Report No. OPSEC 4793

November 2020

1.0 INTRODUCTION

Testing vehicles in their operating environment is essential to verify safety requirements and design decisions. While full vehicle testing is usually completed at a proving ground [4-6], it is often more cost and time effective to perform laboratory testing if the capability exists and can produce the desired event that allows performance to be measured. An example of this is shown in Figure 1 for the Ground Vehicle Systems Center at the Detroit Arsenal in Warren Michigan [2-3].



Figure 1 – GVSC N-Post Test Equipment (see [1])

The benefits outlined in [1] include

- Observation of test specimens in motion allows the test engineer to evaluate the vehicle negotiating difficult or problem-causing terrain

- Controlled environment for repeatability of previous test conditions, eliminating changes in motion due to time, the driver, and weather or test parameters

Replication of the terrain input to the vehicle is key to providing results that closely match proving ground testing. This is accomplished with the use of profilometers to measure course elevations as a function of distance traveled along the course [7, Section 4]. The data from this process is used by specialized software [8] to control test equipment actuators that replicate proving ground inputs to the vehicle. Besides allowing the play out of drive files, it acquires response data and manages the test lab information. However, the software requires the drive files to be in a particular format. Due to the various course profile data formats (e.g. multiple or single track) and units (e.g. feet or inches), proper processing is required before a drive file can be created.

In 2007, at the request of the Physical Simulation Team (PST), the creation of drive files from course profile data began with the implementation of a Perl [9] script. The course profile data was assumed to be detrended to stay within actuator limits and provide separate left and right tracks, otherwise they were assumed to be symmetrical (left = right). A tire patch filter was added to account for the reduction in course severity imparted to the vehicle due to tire filtering. In 2011, the Perl script was converted to a Python [10,11] script in order to maintain the capability (Python was becoming more popular and Perl was losing). In 2020, the Python script was converted to a Powershell [12] script (Drive File Creation-DFC) with several revisions and corrections. The previous implementations ran on Unix/Linux operating systems and it was thought that using a Microsoft Windows based platform would allow easier usage and availability for the Physical Simulation Team (PST). Updates included a corrected tire patch filter (minimal impact), shifted mean to show after removing the mean from the track data, and requiring speed values to be integer only (easier to format in output file name). Additional functionality includes the scaling of peaks to be within defined limits, insertion of zero columns (unused channels), and the handling of any units provided consistent scale factors are included. An additional Powershell script (Course Profile Conversion-CPC) was added to address converting the course profile data into a consistent format and units.

Table 1 – Revision Timeline

2007-07	original Perl script created
2007-09	tire patch filter (average over) added
2011-11	convert to Python script
2020-08	convert to Powershell (Drive File Creation-DFC) corrected Average Patch Filter (size, loop counter) updated Shift mean to show after removing it from track data updated speed values to only integers added Peak scaling added insert zero column added handling of units (conversion factor)
2020-09	created Course Profile Conversion (CPC) script added detrending to unit conversion

In the following sections the objective is to fully document the details and process for the creation of drive files from course profile data. Some details are obscure and one always has to remember that

The complexity arises when the principles are reduced to practice, for then the realities of implementing logic, storing information, and transferring data have to be addressed.

---Robert F. Stengel, Optimal Control and Estimation (ISBN 0-486-68200-5)

2.0 COURSE PROFILE CONVERSION

The Course Profile Conversion (CPC) PowerShell script reads course profile data in multiple file formats and outputs detrended single track data (left and/or right) after unit conversion using a double-sided exponential filter.

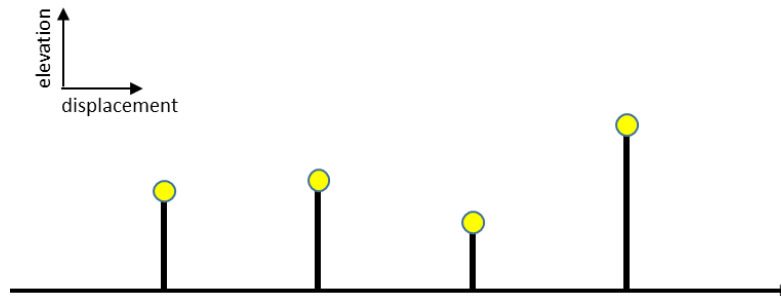


Figure 2 – Profile Section

All of the file formats represent profile data (each track) for all or sections of a course as depicted in Figure 2. The purpose of CPC is to provide a consistent course profile format for the Drive File Creation (DFC) script. The output from CPC is usually designated as a “.cpc” file and used as input into DFC.

2.1 Course Profile File Formats

CPC has the ability to read three different types of file formats. The file format must be specified when CPC is called. The format for one column single track data (Type 1) is given with examples in Figures 3a and 3b.

- ✓ Type 1 (single track, one column)
 - [comment lines]
 - [number of elevation samples, displacement per sample]
 - [elevation samples]

In this format, the elevation data is given in multiple rows or a single column. The spacing is assumed to be constant and provided in the header information along with the number of samples.

```
===BOF
apg_IV 2.1942 RMS
APGIV 468 FEET LONG, PROFILE 4 ( 2.194 in. rms ) L=468 ft
  469 12.
    .00 .84 1.68 2.52 3.36 4.08 4.92 5.76 6.60 7.44
    8.28 9.12 9.96 10.80 11.64 11.40 10.32 9.12 8.04 6.84
    5.76 4.56 3.48 2.28 1.20 .00 .00 .00 .00 .00
    .
    .
    .
===EOF
```

Figure 3a – Type 1 File Format (multiple rows)

```

===BOF
RMS1.30-23.txt /magnitude,dx in inches
4000, 3.000000
0.427780
0.779344
0.938570
.
.
.
===EOF

```

Figure 3b – Type 1 File Format (single column)

The format for two column single track data (Type 2) is given with an example in Figure 4.

- ✓ Type 2 (single track, two columns)
- [comment lines]
- [displacement samples, elevation samples]

In this format, the elevation data is given in a single column after the displacement data column. The sample spacing could theoretically be variable.

```

===BOF
0.0000      0.0834
0.2500      0.0836
0.5000      0.0808
.
.
.
===EOF

```

Figure 4 – Type 2 File Format

The format for multiple column multiple track data (Type 3) is given with an example in Figure 5.

- ✓ Type 3 (multiple track, multiple columns)
- [comment lines]
- [displacement samples, elevation 1 samples, elevation 2 samples, ...]

In this format, multiple elevation data columns are given after the displacement data column. The sample spacing of the directly measured data is variable due to the spatial variation of the sensors across the width of the vehicle. For example, while turning a corner the inner sensor will move slower than the outer sensor ($s=r\theta$). Processing is required to interpolate each measurement back to a center path to achieve equal spacing. One advantage of multiple track data is the assurance of correlation and the accuracy which you can achieve for a given track width. The profilometer that collects this type of data is described in Appendix A. ATC Laser Profiler [7].

```

===BOF
Distance (ft),Laser 1 (ft),Laser 2 (ft),Laser 3 (ft),Laser 4 (ft),Laser 5 (ft)
0.00000000,0.03423493,0.10571433,0.08979320,-0.24436195,-0.48479081
0.25000000,0.03943786,0.08687522,0.06247506,-0.20190297,-0.52107400
0.50000000,0.01217997,0.07665324,0.02443129,-0.27996569,-0.55725897
.
.
.
===EOF

```

Figure 5 – Type 3 File Format

File formats not defined here can be read by modifying CPC to accommodate them or by manually processing them to one of the formats in this section or section 2.4 Output File Format. If this is done, it is incumbent upon the creator to ensure all requirements are met (e.g. consistent units, detrended). Many scripting tools are available to accomplish this---Powershell and MATLAB [14] are a few examples.

2.2 Unit Conversion

The purpose for unit conversion is to ensure consistent units for drive file creation. DFC converts the profile data in the spatial domain into actuator position in the time domain, given a specified speed ($T=D/V$). This process is best served if units are the same for displacement and elevation. Conversion factors must be specified when CPC is called. **The supplied unit conversion factors for each type of file format should ensure that the resulting units for DISPLACEMENT and ELEVATION are the SAME.**

For Type 1 and 2 formats two conversion factors are required.

- cf1 conversion factor1 for displacement <per sample (Type 1) | column 1 (Type 2)>
- cf2 conversion factor2 for elevation <column 1 (Type 1) | column 2 (Type 2)>

For Type 1 formats, the displacement column is created based on the sample distance and number of samples.

For Type 3 formats three conversion factors are required.

- cf1 conversion factor1 for displacement column 1
- cf2 conversion factor2 for elevation left track
- cf3 conversion factor3 for elevation right track

After unit conversion and detrending the left and right track data are output separately.

2.3 Detrending

The purpose for detrending is to retain the profile roughness characteristics (high frequency) while removing elevation trends (low frequency) that would restrict usage in laboratory testing due to actuator travel limits. These low frequency trends generally do not excite the vehicle for operational speeds. DFC converts the profile data in the spatial domain into actuator positions in the time domain, given a specified speed ($T=D/V$). This information is useless if it exceeds the actuators travel limits. This is addressed in DFC through peak scaling, but is expected to be used only when a few peaks exceed the actuator travel limits as it does change the course profile in those areas.

Detrending is achieved with a double-sided exponential filter as shown in Figure 6. A detailed explanation of this filter and detrending is contained in [13]. Filter distance must be specified when CPC is called. If no detrending filter is specified (detrending is not desired), a warning message is given as CPC is intended to prepare data to create drive files for test equipment which requires the data be detrended (limited actuator displacement). Not detrending in CPC will allow an external detrending method to be used.

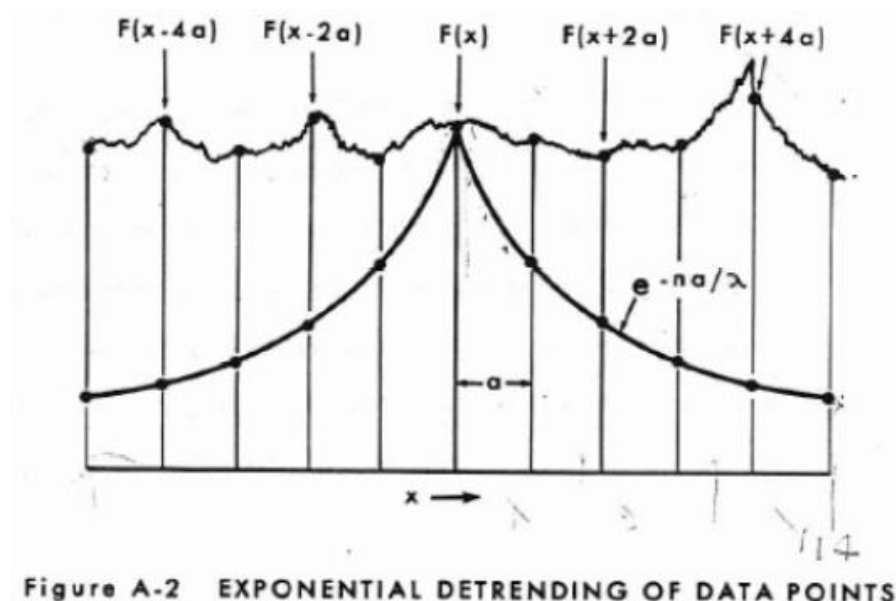


Figure 3 – Double-sided Exponential Filter (see [12])

Figure 6 – Double-Sided Exponential Filter [13]

The "filter_distance_one_side (fd1s)" parameter defines the one-sided distance of the exponential filter. This filter is also used before calculating root-mean-square (RMS) roughness values. The standard distance is 30 feet [13] on each side of the sample point, giving a total filter distance of 60 feet (+/-30 feet). The appropriate value to use for fd1s is to convert 30 feet into the desired units which should be the same units of length (used for displacement) **after the conversion factors are applied** (see cf1). Table 2 shows example values for fd1s given specific units of length.

Table 2 – Filter Distance (one side)

UNITS	fd1s (30 feet)
ft	30
in	360
m	9.144
mm	9144

The sample distance is also required for the detrending filter but CPC determines it from the course profile data provided (assumes is equally spaced). Detrending of each single track data file (left or right) is done separately, except for file format type 3 where both left and right track data is provided.

There is an internal flag in CPC that by default (\$pad = \$true) indicates padding is to be done before detrending. Padding includes extending the beginning of the profile data with the initial value for the number of samples for half the filter length (one-sided distance) and the ending of the profile data with the final value for the same number of samples. This allows detrending to begin at the initial profile value, otherwise, the center point of the filter must be advanced by the number of samples for half the filter length so that the left side (or right side) of the filter has data to use for filtering. Without padding the course profile length is reduced by the filter length as shown in Figure 7. If no padding is desired then set \$pad = \$false in CPC.

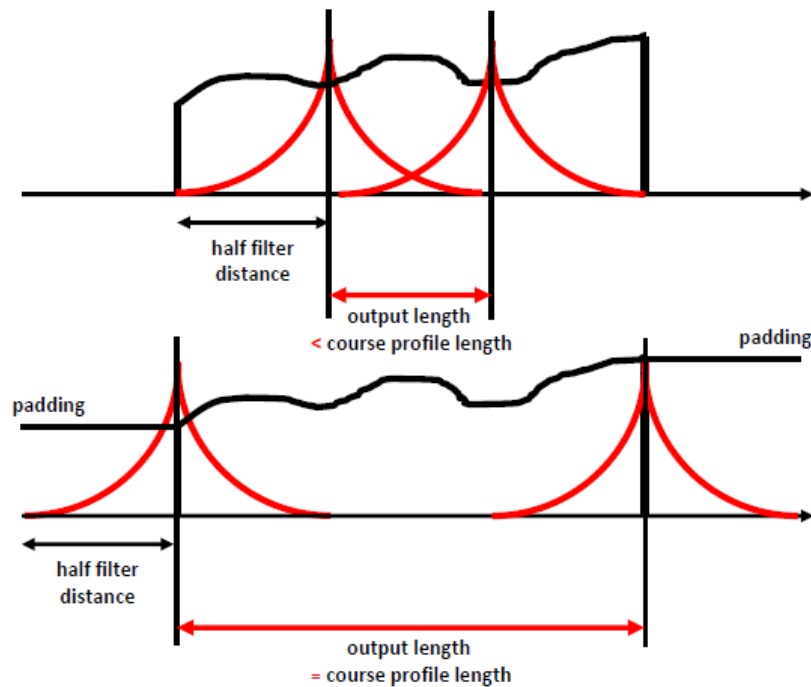


Figure 7 – Course Output Length w/wo Padding

2.4 Output File Format

The output from CPC is a detrended single track data file whose format is two columns, displacement (equally spaced) and elevation data, respectively (Type 2 file format). Depending on the input file format type, separate left and right detrended single track data files may be generated (see Type 3). The supplied unit conversion factors for each type of file format should ensure that the resulting units for DISPLACEMENT and ELEVATION are the SAME---as this is the assumption for the output from CPC. The output is usually designated as a “.cpc” file (with the name including the units of length and whether detrended or not (e.g. crse1_in_d.cpc) and used as input into DFC. If no detrending filter was specified, the output will not be detrended and it is assumed that an external detrending method will be applied to the data before using it as input to DFC (or that the data has been verified to be within the actuator travel limits).

2.5 Execution

The CPC script is executed where the input file name and format type are specified along with a variable number of parameters depending upon the file format type. Tables 3 and 4 show the required and optional parameters (“[]” brackets indicate optional parameters). The green highlighted areas in Table 4 show the differences from Table 3. Figure 8 shows execution examples from the Powershell prompt or from within another Powershell script.

Table 3 – CPC Type 1 or 2 File Format Parameters

Type 1 or 2	PS> .lcpc <ifn> <ft> (<nocl>, <cf1>, <cf2>, <ofn1>, [fd1s])
ifn	input file name
ft	format type
nocl	number of comment lines
cf1	conversion factor1 for displacement <per sample (Type 1) column 1 (Type 2)>

cf2	conversion factor2 for elevation <column 1 (Type 1) column 2 (Type 2)>
ofn1	output file name
fd1s	filter distance one side

Table 4 – CPC Type 3 File Format Parameters

Type 3	PS> .lcpc <ifn> <ft> (<nocl>, <cf1>, <cf2>, <cf3>, <lcn>, <rcn>, <ofn1>, <ofn2>, [fd1s])
ifn	input file name
ft	format type
nocl	number of comment lines
cf1	conversion factor1 for displacement column 1
cf2	conversion factor2 for elevation left track
cf3	conversion factor3 for elevation right track
lcn	left track column number (not zero based)
rcn	right track column number (not zero based)
ofn1	left track output file name
ofn2	right track output file name
fd1s	filter distance one side

```

=====Type 1 (in to mm)

PS> $params = @(1, 25.4, 24.5, "crse1_mm_d.cpc", 9144)
PS> .\cpc .\data\crse1.txt 1 @params

=====Type 2 (ft to in)

PS> $params = @(0, 12.0, 12.0, "crse2_in_d.cpc", 360)
PS> .\cpc .\data\crse2.txt 2 @params

=====Type 3 (ft to mm)

PS> $params = @(1, 304.8, 304.8, 304.8, 6, 2, "crse3L_mm_d.cpc", "crse3R_mm_d.cpc")
PS> .\cpc .\data\crse3.txt 3 @params

```

Figure 8 – Example CPC Execution with Parameters

Input parameters after the input file name and format type are passed to CPC in an array that varies in size depending upon the file type and detrending filter option. These parameters are accessed using the \$args variable (e.g. \$args.length, \$args[0], \$args[1], ...)

In Table 4, the column numbers for course profile data begin with the number one (1) (they are zero-based only internally to CPC).

Comparisons of detrending, with and without padding, with MATLAB versus CPC are shown in Appendix A1. Appendix A1 also includes comparisons of unit conversions, with and without detrending. In these cases, the converted units are scaled back to the original units manually to show any differences. In all cases the differences are less than 1e-8.

In Figure 8, the Type 1 example has conversions factors cf1 and cf2 set to 25.4 with the filter distance fd1s set to 9144. These values relate to converting inches to mm (25.4 mm/1 inch) and to filter distance in the units after the conversion is applied ($30 \text{ ft} * 304.8 \text{ mm/ft} = 9144 \text{ mm}$). The other examples in Figure 8 follow a similar pattern. In the Type 3 example (see Figure 5 also) the measurements are taken as the laser scans from right to left. Because of this, the left track column number is greater than the right track column number.

3.0 DRIVE FILE CREATION

The Drive File Creation (DFC) PowerShell script creates time history channel data for specified speeds and course profiles that define simulator actuator positions versus time for laboratory testing. The time histories are based on detrended left and right track course profiles output from CPC (".cpc" files, see 2.4 Output File Format). The terms speed and velocity are used interchangeably. Parameters are sometimes referred to as commands or functions if their values represent some action to be taken. The output from DFC is usually designated as a ".dfc" file.

3.1 Parameters

Many parameter values are required to create the ".dfc" drive file. A definition file ".def" is required that defines the necessary parameter values and is used as input by DFC. The following sections describe individual parameters defined in the ".def" file. The units of time are always in seconds. It is assumed that the left and right course profile tracks are available and detrended.

3.1.1 Comments

The comment line allows comments to be added to the ".def" file for clarity, meaning, and documentation of the parameter value selection. A comment line begins with a pound sign "#". Parameters also begin with this character. In order to prevent parameter execution include a space after the "#" character if the following characters are a parameter name. Blank lines are acceptable.

```
# This and the following lines are comment lines. The third line is a command line.
# SAMPLEFREQ <sampling_freq>
#SAMPLEFREQ 204.8
```

3.1.2 Track Data Input

Track input data allows the course profile to be specified for individual left and right tracks. Separate file names for the left and right track course profile data are used. The format required for each track file is two columns, with column 1 holding the displacement samples (equal spaced) and column 2 holding the associated elevation samples---which is the Type 2 format specified in 2.4 Output File Format for CFC. **BOTH columns must be in the SAME units of length.**

```
# Define left and right track inputs
# TRACKS <l_trk_name> <r_trk_name>
#TRACKS ./ter/3courL_in_d.cpc ./ter/3courR_in_d.cpc
```

3.1.3 Output Channels

The output channels defines which track will be used to create each channel of actuator position (elevation vs. time) time history data. A track data index value is used to define an output channel (1=left, 2=right). The number of channels should match the number of vehicle wheels/spindles being actuated (see #TRACKS in 3.1.2 Track Data Input and #DELAY in 3.1.4 Channel Delay).

```
# Select track data for output channels
# OUTPUTCH <ch1_trk_indx> <ch2_trk_indx> [<ch3_trk_indx> <ch4_trk_indx> ...]

# For a 2-axle vehicle where the 2nd and 4th channels are for actuating the front RIGHT
# and rear RIGHT wheels/spindles and the 1st and 3rd channels are for actuating the front
# LEFT and rear LEFT wheels/spindles
#OUTPUTCH 1 2 1 2

# For a 3-axle vehicle
#OUTPUTCH 1 2 1 2 1 2
```

The ordering of channels is left to the user, but the ordering scheme must be followed consistently for the other parameters (see #DELAY in 3.1.4 Channel Delay). For a 2-axle vehicle example in Figure 9, the

output channels could alternate between left and right from front to rear (LF, RF, LR, RR = “#OUTPUTCH 1 2 1 2”) or from front to rear for the left and then right sides (LF, LR, RF, RR = “#OUTPUTCH 1 1 2 2”). The selected ordering of output channels should be the one that best fits with the test equipment actuator control.

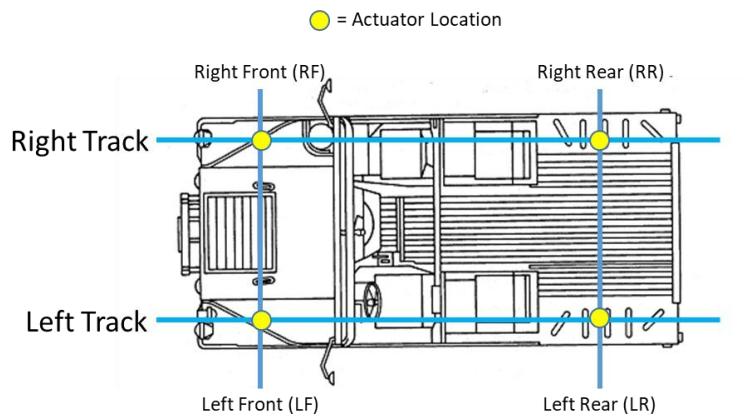


Figure 9 – Channel Output 2-Axle Example

3.1.4 Channel Delay

The channel delay allows the output channel time history data to reflect the effect axle spacing has on delaying the terrain input. The delay is specified in displacement (units of length) for each channel. The number of delay values should match the number of output channels **and order** specified in #OUTPUTCH (see 3.1.3 Output Channels) and represents the distance behind the front (first) axle---see Figures 10a and 10b.

```
# Specify delay length for output channels
# DELAY <ch1_delay_dis> <ch2_delay_dis> [<ch3_delay_dis> <ch4_delay_dis> ...]

# For a 2-axle vehicle, zero distance for the front left/right channels and 130.0 wheelbase
# distance for the rear left/right channels in the units of length in the track data file
# OUTPUTCH 1 2 1 2
# DELAY 0 0 130.0 130.0

# For a 3-axle vehicle
# OUTPUTCH 1 2 1 2 1 2
# DELAY 0 0 130.0 130.0 260.0 260.0
```

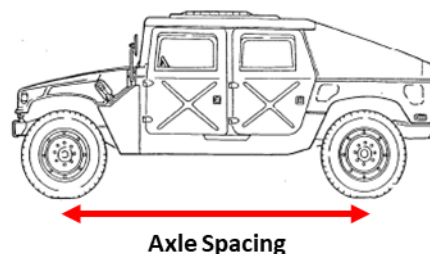


Figure 10a – Channel Delay 2-Axle Example

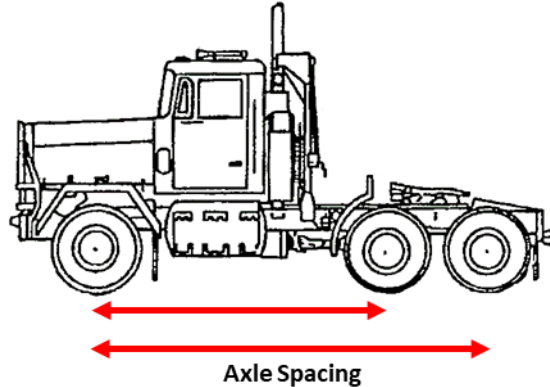


Figure 10b – Channel Delay 3-Axle Example

3.1.5 Speed

The speed values determine the course profile distance traveled for a given time interval. Generally higher speeds would reflect more severe course inputs. Specify vehicle speeds as a single or range of speeds with increments, in units of speed, which the time history output is to represent. **Only INTEGER speed values are allowed.** The units of speed used here when multiplied by #VEL2LEN (3.1.15 Velocity to Length Conversion Factor) and unit time [sec] should match the units of length in the track data files (see 3.2 Track Data Input) for displacement and elevation (e.g. "#VEL2LEN 17.6" would convert velocity in units of [MPH] times 1 [sec] to length in units of [in] or "#VEL2LEN 0.277" would convert velocity in units of [kph] times 1 [sec] to length in units of [m]). Single speeds and ranges are separated by commas ",". A speed range is specified with a beginning and ending speed separated by colons ":" with an increment value in between (e.g. 1:10:100 – a range from 1 to 100 in increments of 10).

```
# Specify vehicle speeds the output channel time histories represent
# SPEED <spd1>, <spd2> [,<spd_range>]

# For a single speed
#SPEED 45

# For a range of speeds (from 10 to 25 in increments of 5)
#SPEED 10:5:25

# For a combination of single/range of speeds
#SPEED 5,20:5:45,70
```

3.1.6 Sample Frequency

The sampling frequency determines the resolution of the output channel time history data. Channel output sampling frequency is in units of [Hz]. (Currently, the value of 204.8 Hz is required for specific equipment in the GVSC laboratory)

```
# Specify sampling frequency of time history output.
# SAMPLEFREQ <sampling_freq>

# Frequency in units of Hz
#SAMPLEFREQ 204.8
```

3.1.7 Ramp Time

The ramp time allows the output channel time history data to command the actuators to come to their beginning positions gradually. Specify ramp time in units of [sec] to ramp "up" the time history signal from zero to its beginning elevation value and ramp "down" its ending elevation value to zero. See Figure 11 for an example. The beginning and ending elevation values are determined after the mean of the track profile

data (both left and right) is subtracted. This means that these elevation values could be negative and ramp “down” instead of “up” would more appropriately describe the action.

```
# Specify ramp time
# RAMPTIME <ramp_time>

# 5 second ramp time
#RAMPTIME 5
```

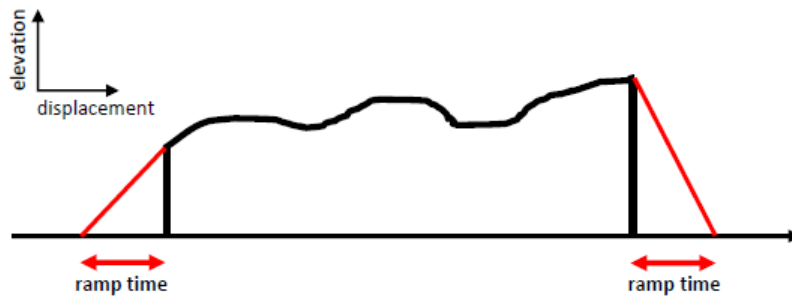


Figure 11– Ramp Time

3.1.8 Wait Time

The wait time allows the output channel time history data to command the actuators to wait or “rest” for a period of time before actuator position commands (elevation vs. time) are given after ramp time (see 3.1.7 Ramp Time). This wait period allows motion to settle out so that measurement data starts clean. Specify wait time in units of [sec] before the channel data should start after ramp time. See Figure 12 for an example.

```
# Specify wait time
# WAITTIME <wait_time>

# 5 second wait time
#WAITTIME 5
```



Figure 12– Wait Time

3.1.9 Travel Limit

The travel limit allows the drive file creation process to determine when the output channel time history data exceeds the actuators capability. It is assumed that all actuators have the same travel limit. The beginning and ending elevation values are determined after the mean of the track profile data (both left and right) is

subtracted (see 3.1.7 Ramp Time)---it is centered about zero. Specify the absolute value of the travel limit from zero, which should match the units of length in the track data files for displacement and elevation. This value is used only to indicate if the track file data requires the actuator to exceed this amount and in #SCALEPK (see 3.1.16 Scale Peaks) to resolve this issue (if peak scaling is requested). See Figure 13 for an example.

```
# Specify travel limit
# LIMITTRAV <travel_limit>

# 5 second wait time
#LIMITTRAV 10
```

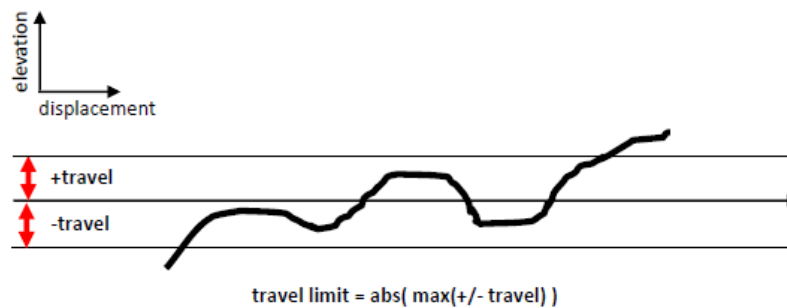


Figure 13– Travel Limit

3.1.10 Velocity Limit

The velocity limit allows the drive file creation process to determine when the output channel time history data exceeds the actuators capability. It is assumed that all actuators have the same velocity limit. Specify the absolute value of the velocity limit, which should match the units of length in the track data files for displacement and elevation (see 3.1.13 Unit Length Symbol) divided by time (1 [sec]). **This is an exception to the velocity units defined elsewhere because of the way it is calculated---a difference in the elevation divided by sample time (reciprocal of the sampling frequency) and is generally expressed in smaller units than vehicle speed (see 3.1.5 Speed).** This value is used only to indicate if the track file data requires the actuator to exceed this amount. See Figure 14 for an example.

```
# Specify velocity limit
# LIMITVEL <velocity_limit>

#UNITLEN in
# 75 [UNITLEN/sec] = 75 in/sec velocity limit
#LIMITVEL 75
```

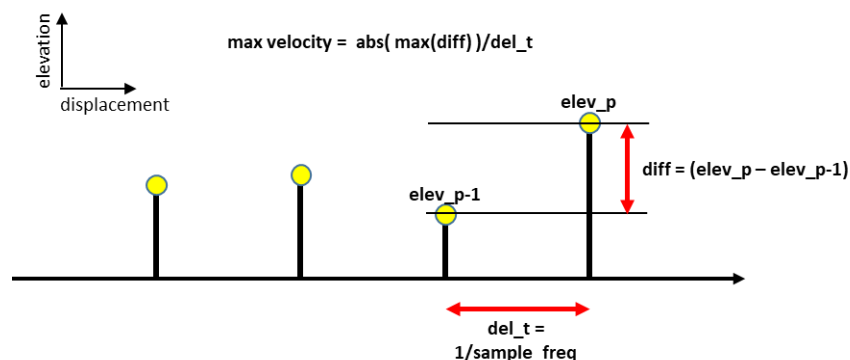


Figure 14– Velocity Limit

3.1.11 Tire Patch Filter

The tire patch filter smooths the track data elevation points **before processing** to approximate the effects tires would have on softening the impact of the course profile into the vehicle. Specify tire patch filter in units of length (0=no filtering). The length units should match those in the track data files for displacement and elevation. This filter uses a Hann Function [15] or Raised Cosine Window to perform low-pass filtering. The filter length is usually set to 1.5 times (1.5x) the tire radius (there is no theoretical basis for using this value, only observation from limited test data for a specific vehicle).

```
# Specify tire filter length
# PATCH <patch_filter_len>

# Filter length for a tire with radius of 17 units of length (17 x 1.5 = 25.5)
#PATCH 25.5
```

This value is used to determine the actual length of the Hann Function. The green highlighted area of Equation (1) shows how the tire patch filter value (1.5R) determines the actual one-sided length (L) of the Hann Function

$$L = \text{int} \left[\frac{1.5R}{2\Delta x} + 1 \right] \quad (1)$$

where R is the tire radius and Δx is the sample spacing of the course profile as shown in Figure 15. If the tire patch filter value was the tire diameter (1.5R -> 2R) then Equation (1) would effectively give the number of sample points in the tire radius. The 1.5 scale factor reduces this slightly as the tire patch in contact with the ground does not extend to the outer edge of the tire. The suggested scale factor of 1.5 can be changed at the user's discretion.

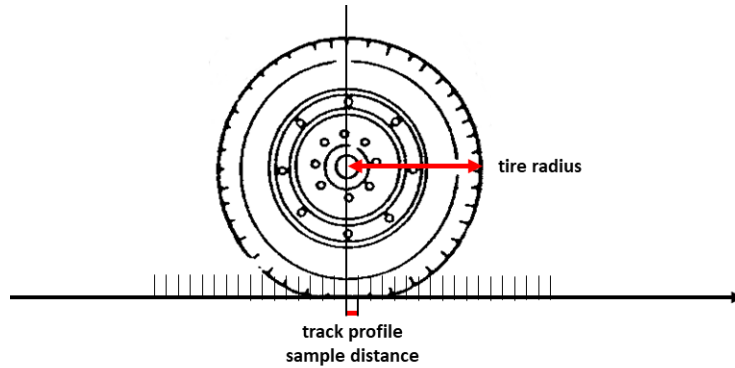


Figure 15– Tire Patch Filter Parameters

Assuming the units of length are inches [in] and the sampling distance is 1 [in], then from (1)

$$L = \text{int} \left[\frac{25.5}{2(1)} + 1 \right] = 13 \quad (2)$$

and the actual Hann Function length is $2*L+1 = 27 = N+1$, which is always odd so it can be centered on the data point being filtered. Padding is used so profile data is not lost (see 2.3 Detrending). Figure 16 shows the un-normalized function for $L=13$ while Table 5 details the calculation. Column 3 in Table 5 is the function shown in Figure 16 while column 4 (last) is the normalized function used for filtering the data. The filtering process is the same as in section 2.3 Detrending but with different filter values.

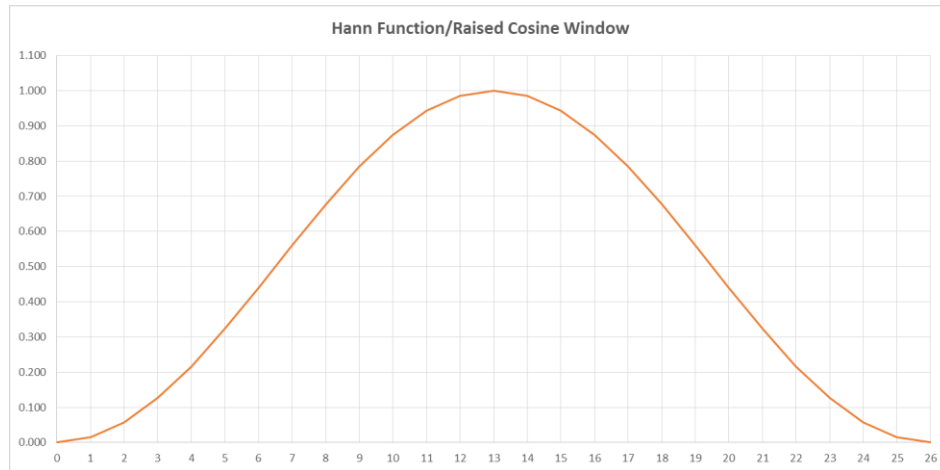


Figure 16– Hann Function [15]

Table 5 – Hann Function Calculation

L=13			
N=26			
$F=0.5-0.5*\cos(2*\pi()*i/N)$			
i = 0..N	i/N	F	F/SUM(F)
0	0.000	0.000	0.000
1	0.038	0.015	0.001
2	0.077	0.057	0.004
3	0.115	0.126	0.010
4	0.154	0.216	0.017
5	0.192	0.323	0.025
6	0.231	0.440	0.034
7	0.269	0.560	0.043
8	0.308	0.677	0.052
9	0.346	0.784	0.060
10	0.385	0.874	0.067
11	0.423	0.943	0.073
12	0.462	0.985	0.076
13	0.500	1.000	0.077
14	0.538	0.985	0.076
15	0.577	0.943	0.073
16	0.615	0.874	0.067
17	0.654	0.784	0.060
18	0.692	0.677	0.052
19	0.731	0.560	0.043
20	0.769	0.440	0.034
21	0.808	0.323	0.025
22	0.846	0.216	0.017
23	0.885	0.126	0.010
24	0.923	0.057	0.004
25	0.962	0.015	0.001
26	1.000	0.000	0.000
SUM		13.000	1.000

3.1.12 Channel Data Output Name

The channel data output name allows parameter substitution so output can be clearly identified. Because the time histories are speed dependent, the following composite formatting syntax "[SPEED={index [,alignment] [:formatString]]]" can be included as part of the output name to have the speed value substituted into it (e.g. "#SPEED 3,1:10:100,55" then "#OUTPUT veh1_crse1_[SPEED={0,-3:D3}].dfc" would output channel time history data for each speed specified in #SPEED as a zero padded 3 digit integer, "veh1_crse1_003.dfc", "veh1_crse1_001.dfc", "veh1_crse1_011.dfc", ..., "veh1_crse1_100.dfc", "veh1_crse1_055.dfc"). An excerpt from [16] and [17] on formatting details follows.

The Decimal ("D") Format Specifier

The "D" (or decimal) format specifier converts a number to a string of decimal digits (0-9), prefixed by a minus sign if the number is negative. This format is supported only for integral types.

The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier. If no precision specifier is specified, the default is the minimum value required to represent the integer without leading zeros.

The result string is affected by the formatting information of the current NumberFormatInfo object. As the following table shows, a single property affects the formatting of the result string.

NumberFormatInfo property	Description
NegativeSign	Defines the string that indicates that a number is negative.

The following example formats an Int32 value with the decimal format specifier.

C#

```
int value;

value = 12345;
Console.WriteLine(value.ToString("D"));
// Displays 12345
Console.WriteLine(value.ToString("D8"));
// Displays 00012345

value = -12345;
Console.WriteLine(value.ToString("D"));
// Displays -12345
Console.WriteLine(value.ToString("D8"));
// Displays -00012345
```

Standard numeric format strings are used to format common numeric types. A standard numeric format string takes the form Axx, where:

A is a single alphabetic character called the format specifier. Any numeric format string that contains more than one alphabetic character, including white space, is interpreted as a custom numeric format string. For more information, see Custom Numeric Format Strings.

xx is an optional integer called the precision specifier. The precision specifier ranges from 0 to 99 and affects the number of digits in the result. Note that the precision specifier controls the number of digits in the string representation of a number. It does not round the number itself. To perform a rounding operation, use the Math.Ceiling, Math.Floor, or Math.Round method.

When precision specifier controls the number of fractional digits in the result string, the result string reflects a number that is rounded to a representable result nearest to the infinitely precise result. If there are two equally near representable results:

```
# Specify output name with SPEED substitution
# OUTPUT <ch_output_name>

# SPEED as a zero padded 3 digit integer
#OUTPUT ./veh/veh1_crse1_[SPEED={0,-3:D3}].dfc
```

This is a special parameter. As soon as it is encountered the drive file creation process begins after checking to ensure all required parameter values have been defined. Only parameter values that change need to be updated before using this parameter again. With this feature, multiple course profile, vehicle, and speed dependent drive files can be created from one ".def" file.

As an example, the following ".def" file creates a drive file ("vehONE_crseONE_010.dfc") for vehicle "vehONE" over course "crseONE" at a speed of 10 MPH, then assumes all the parameters are the same except it updates the course profile ("crseTWO" from "crseONE"), redefines the speed range (from 10 MPH to a range of 25, 30, 35 MPH), and then updates the output name to create drive files ("vehONE_crseTWO_025.dfc", "vehONE_crseTWO_030.dfc", "vehONE_crseTWO_035.dfc"). The changes are highlighted in yellow.

```
# Example .def
.
.
.
#UNITVEL MPH

#SPEED 10
#TRACKS ./ter/crseONEleft_in_d.cpc ./ter/crseONERight_in_d.cpc
#OUTPUT ./veh/vehONE_crseONE_[SPEED={0,-3:D3}].dfc

#SPEED 25:5:35
#TRACKS ./ter/crseTWOleft_in_d.cpc ./ter/crseTWOright_in_d.cpc
#OUTPUT ./veh/vehONE_crseTWO_[SPEED={0,-3:D3}].dfc
```

3.1.13 Unit Length Symbol

The unit length symbol is used only to document the required length units that are to be used. Specify the symbol that represents the length units being used (e.g. "#UNITLEN in" for units of length in inches, "mm" for units of length in millimeters, etc.). All units of length are assumed to be the same and of this type. The length units should match those in the track data files for displacement and elevation.

```
# Specify unit length symbol
# UNITLEN <len_symbol>

# Symbol "in" for units of length in inches
#UNITLEN in
```

3.1.14 Unit Velocity Symbol

The unit velocity symbol is used only to document the required velocity units that are to be used. Specify the symbol that represents the velocity units being used (e.g. "#UNITVEL MPH" for units of velocity in miles per hour, "kph" for units of velocity in kilometers per hour, etc.). All units of velocity are assumed to be the same and of this type **with the exception of the units in 3.1.10 Velocity Limit**. The velocity units should match those in 3.1.5 Speeds.

```
# Specify unit velocity symbol
# UNITVEL <vel_symbol>

# Symbol "MPH" for units of velocity in miles per hour,
#UNITVEL MPH
```

3.1.15 Velocity to Length Conversion Factor

The velocity to length conversion factor is used to determine the speed based sampling distance using the sampling frequency as in Equation (3) and the speed based run distance in Equation (4) for the output channel time history data.

$$\Delta x[UNITLEN] = \frac{speed[UNITVEL] * VEL2LEN}{sample_freq[Hz]} \quad (3)$$

$$run_distance[UNITLEN] = speed[UNITVEL] * VEL2LEN * wait_time[sec] + delay_distance[UNITLEN] \quad (4)$$

Specify the conversion factor between velocity and length. This value should correspond to converting the units in #UNITVEL times unit time (1 [sec]) to the units in #UNITLEN (**with the exception of the units in 3.1.10 Velocity Limit**). The basis for this is the relation $D=VT$ where $D[\text{length}] = (V[\text{length/time}] * <vel_to_len>) * T[\text{sec}]$ (e.g. "#VEL2LEN 17.6" would convert velocity in units of [MPH] times 1 [sec] to length in units of [in], "#VEL2LEN 0.277 would convert velocity in units of [kph] times 1 [sec] to length in

units of [m], "#VEL2LEN 277.78 would convert velocity in units of [kph] times 1 [sec] to length in units of [mm]).

```
# Specify velocity to length conversion factor
# VEL2LEN <vel_to_len>

# [MPH]*[sec] to [in], 1[MPH]*17.6 = 17.6[in/sec]*1[sec] = 17.6[in]
#VEL2LEN 17.6
```

3.1.16 Scale Peak

The scale peak parameter scales the channel output data to a percentage of the #LIMITTRAV value to prevent actuators from exceeding that value (see 3.1.9 Travel Limit). Specify the percentage of #LIMITTRAV to scale peak values greater than #LIMITTRAV to (0=no scaling) (e.g. "#SCALEPK 0.95" would scale peaks greater than #LIMITTRAV to 95% of #LIMITTRAV). This function assumes the signal has been centered about zero and that the scaled peaks have values below #LIMITTRAV before and after the peak value excluding the ramp and wait time sections. Figure 17 demonstrates the scaling effect.

```
# Specify scale peak percentage
# SCALEPK <percent_pk_scale>

# scale peaks greater than LIMITTRAV to 95% of LIMITTRAV
#SCALEPK 0.95
```

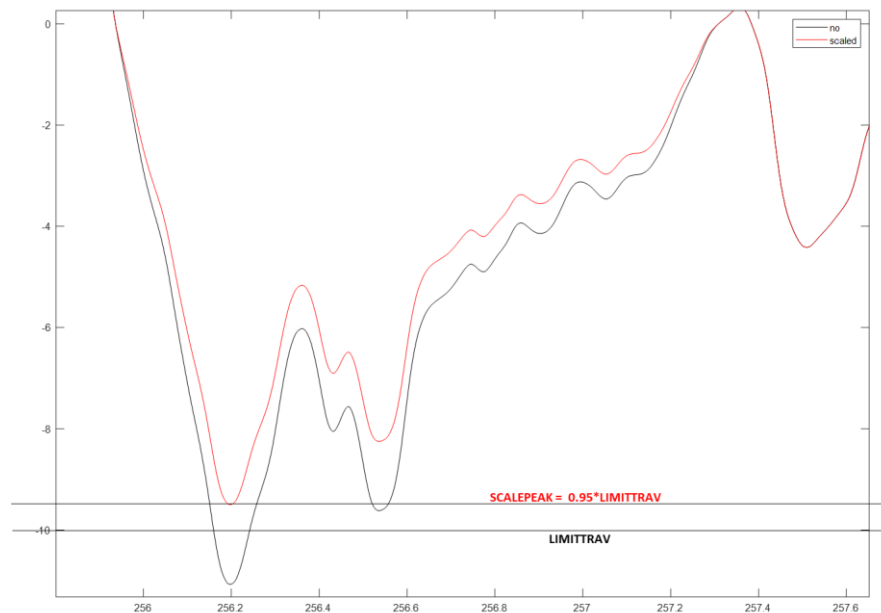


Figure 17– Scale Peak Example

3.1.17 Insert Columns

The insert columns function inserts zero value channels into the channel output time history data to account for existing channels that do not need to be actuated. Specify the number of zero value columns to insert before, between, and after defined channels (0=no insertion, max=10) (see 3.1.3 Output Channels). The number of values should be one more than the length of #OUTPUTCH as shown in the example.

```

# Specify number of zero columns to insert
# INSERT <ncol> <ncol> [<ncol> <ncol> ...]

# For a 2-axle vehicle
# insert 3 before ch1, 1 between ch1/ch2, 1 between ch2/ch3, none between ch3/ch4,
# and 2 after ch4
# OUTPUTCH 1 2 1 2
#INSERT    3 1 1 0 2

# For a 3-axle vehicle
# OUTPUTCH 1 2 1 2 1 2
#INSERT    3 1 1 0 2 0 0

# For a 2-axle vehicle, actuate front left only (quarter car)
# OUTPUTCH 1
#INSERT    0 3

```

3.2 Output File Format

The output from DFC is a multiple column data file for each set of parameters specified before each #OUTPUTCH command (e.g. vehicle parameters, speeds, course profile, processing options) whose columns represent channels of actuator position time history data (see 3.1.12 Channel Data Output Name). The format is given with an example in Figure 18.

- ✓ DFC Output (multiple columns, '{0,-13:F8}')
[time, ch1 position, ch2 position, ch3 position, ...]

The output is usually designated as a “.dfc” file (with the name including the units of length).

```

===BOF
0.00000000    0.00000000    0.00000000    0.00000000    0.00000000    0.00000000    0.00000000
0.00488281   -0.00509685   -0.00520018   -0.00509685   -0.00520018   -0.00509685   -0.00520018
0.00976563   -0.01019371   -0.01040036   -0.01019371   -0.01040036   -0.01019371   -0.01040036
0.01464844   -0.01529056   -0.01560055   -0.01529056   -0.01560055   -0.01529056   -0.01560055
0.01953125   -0.02038742   -0.02080073   -0.02038742   -0.02080073   -0.02038742   -0.02080073
.
.
.
===EOF

```

Figure 18 – DFC File Format

3.3 Execution

The DFC script is executed where the input file name (see “.def” in 3.1.Parameters) is specified. Table 6 shows the required parameters even if their values indicate no action. Figure 19 shows an execution example from the Powershell prompt or from within another Powershell script.

Table 6 – DFC Parameters

PARAMETER	REFERENCE
#	3.1.1 Comment Line
#TRACKS	3.1.2 Track Data Input
#OUTPUTCH	3.1.3 Output Channels
#DELAY	3.1.4 Channel Delay
#SPEED	3.1.5 Speed
#SAMPLEFREQ	3.1.6 Sample Frequency
#RAMPTIME	3.1.7 Ramp Time
#WAITTIME	3.1.8 Wait Time
#LIMITTRAV	3.1.9 Travel Limit

#LIMITVEL	3.1.10 Velocity Limit
#PATCH	3.1.11 Tire Patch Filter
#OUTPUT	3.1.12 Channel Data Output Name
#UNITLEN	3.1.13 Unit Length Symbol
#UNITVEL	3.1.14 Unit Velocity Symbol
#VEL2LEN	3.1.15 Velocity to Length Conversion Factor
#SCALEPK	3.1.16 Scale Peak
#INSERT	3.1.17 Insert Column

```

#=====Drive File Creation

PS> .\dfc hmmwv.def

#=====“.def” Example

PS> cat hmmwv.def

# HMMWV, 2 axles, wheel base 130 inches
#OUTPUTCH 1 2 1 2
#DELAY 0 0 130.0 130.0
#SPEED 45
#SAMPLEFREQ 204.8
#RAMPTIME 5
#WAITTIME 5
#LIMITTRAV 10
# Units of UNITLEN/sec
#LIMITVEL 75
# tire radius = 17, 1.5x = 25.5
#PATCH 25.5
#UNITLEN in
#UNITVEL MPH
#VEL2LEN 17.6
#INSERT 0

#-----COURSES (SET 1)
#SPEED 10:5:25
# scale just for P3
#SCALEPK 0.95
#TRACKS ./ter/3courL_in_d.cpc ./ter/3courR_in_d.cpc
#OUTPUT ./veh/hmmwv/hmmwv_p3_[SPEED={0,-3:D3}].dfc
#SCALEPK 0

#TRACKS ./ter/BBL_in_d.cpc ./ter/BBR_in_d.cpc
#OUTPUT ./veh/hmmwv/hmmwv_bb_[SPEED={0,-3:D3}].dfc

#TRACKS ./ter/CHBSML_in_d.cpc ./ter/CHBSMR_in_d.cpc
#OUTPUT ./veh/hmmwv/hmmwv_chBsm_[SPEED={0,-3:D3}].dfc

#-----COURSES (SET 2)
#SPEED 20:5:40
#TRACKS ./ter/MUNGL_in_d.cpc ./ter/MUNGR_in_d.cpc
#OUTPUT ./veh/hmmwv/hmmwv_mg_[SPEED={0,-3:D3}].dfc

#TRACKS ./ter/PRMAL_in_d.cpc ./ter/PRMAR_in_d.cpc
#OUTPUT ./veh/hmmwv/hmmwv_pa_[SPEED={0,-3:D3}].dfc
#-----END

```

Figure 19 – Example DFC Execution with “.def” Parameter File

Comparisons of channel output, 2011 Python script (Linux) versus DFC PowerShell script, are shown in Appendix A2. In all cases the differences are less than 0.84 (Tire Patch Filter was updated in DFC).

3.4 Drive File Creation Process Summary

With the parameters being described in 3.2 Parameters, a summary of the drive file creation process in DFC is outlined. It closely follows the DFC script in Appendix A.4.

- **Read Parameters** (".def" file)
 - foreach ".def" file line
 - set parameter value
 - if the parameter is OUTPUT, then execute **Output Channel Data**, and continue
- **Output Channel Data**
 - check that all parameter values have been defined, if not give error and exit
 - read track data (TRACKS)
 - foreach track file
 - read displacement and elevation data
 - if PATCH -ne 0,
 - create Hann Function coefficients
 - pad track data
 - filter padded track data with Hann Function
 - shift track data (remove mean to center about zero)
 - **forall** track data files
 - find min/max elevation
 - mean = (min + max) / 2
 - **forall** track data files
 - subtract mean
 - find min/max (after mean subtracted)
 - max = abs(max(min,max))
 - travel limit check (LIMITTRAV)
 - create channel data
 - del_t = (1/ SAMPLEFREQ) = sampling time
 - get speed values (SPEED)
 - parse for single value or range
 - foreach speed value
 - foreach output channel
 - ds = track data displacement delta = sampling distance
 - dsf = speed * VEL2LEN * del_t = SPEED,SAMPLEFREQ based sampling distance
 - spatial frequency check
 - $(1/dsf) > 2*(1/ds) \implies ds > 2*dsf$
 - yb = beginning track data elevation
 - ramp beginning
 - foreach RAMPTIME(step del_t),
 - output (yb/RAMPTIME) * RAMPTIME(step del_t)
 - wait beginning
 - run = speed * VEL2LEN * WAITTIME+ DELAY
 - foreach run(step dsf), output yb
 - track data (TRACKS, PATCH)
 - disp = 0..track data end displacement value
 - foreach disp(step dsf), output h
 - h1 = truncate(disp(step dsf) / ds) = first sample point < disp(step dsf)
 - h2 = (disp(step dsf) / ds) - h1 = fraction part between sample points
 - h3 = h1 + 1 = next sample after h1
 - if h3 > channel length, h3 = channel length
 - hslp = (elevation@h3 - elevation@h1) / ds = slope of track data
 - h = hslp * (h2 * ds) + elevation@h1 = interpolated elevation@disp(step dsf)
 - if SCALEPK -ne 0

- find positive peaks > LIMITTRAV, excluding ramp and wait
 - find start/end at zero
- foreach i in (start..end)
 - pmax = value of max peak
 - pidx = index of max peak value
- foreach i in (start..end), output elevation@i * LIMITTRAV/pmax * SCALEPK
- find negative peaks < -LIMITTRAV, excluding ramp and wait
 - find start/end at zero
- foreach i in (start..end)
 - pmin = value of min peak
 - pidx = index of min peak value
- foreach i in (start..end), output elevation@i * LIMITTRAV/abs(pmin) * SCALEPK
- foreach channel elevation data, excluding ramp and wait
 - find min/max of diff elevation
 - max = abs (min/max diff elevation)
 - absvel = max/del_t
- velocity limit check (LIMITVEL)
- ye – ending track data elevation
- wait ending
 - foreach run(step dsf), output ye
- ramp ending (-\$ye/\$ramp_t) * \$k + \$ye
 - foreach RAMPTIME(step del_t),
 - output (-ye/RAMPTIME) * RAMPTIME(step del_t) + ye
- foreach channel
 - find largest ending index (longest channel)
 - lindx = largest ending index
- if INSERT –ne 0
 - set zero column same length as longest channel
 - foreach INSERT number
 - insert zero column into new channel data
 - if –ne last INSERT number
 - add current channel
 - foreach channel
 - find largest ending index (longest channel)
 - lindx = largest ending index
- output data
 - output name substitution (OUTPUT)
 - if can't find “[...]” return
 - parse out parameter name (SPEED) and format, if error warn then exit
 - substitute value into output name for “[...]”
 - check file path, warn if already exists else create
 - foreach i in (0..lindx)
 - foreach channel
 - if i > channel length, write channel[end] value (pad to longest channel)
 - write channel[i] value

The key part of drive file creation is the conversion between track course profile data (elevation versus displacement) into speed and sampling frequency dependent channel time history data (elevation/actuator position versus time) as illustrated in Figure 20. Linear interpolation is used to determine the elevation between profile data samples. Using length units of inches ([in]) and speed units of miles per hour ([MPH]), for dsf to satisfy the Nyquist frequency criteria [18] in Equation (5), larger spatial sampling distances will be required at higher speeds as shown in Table 7.

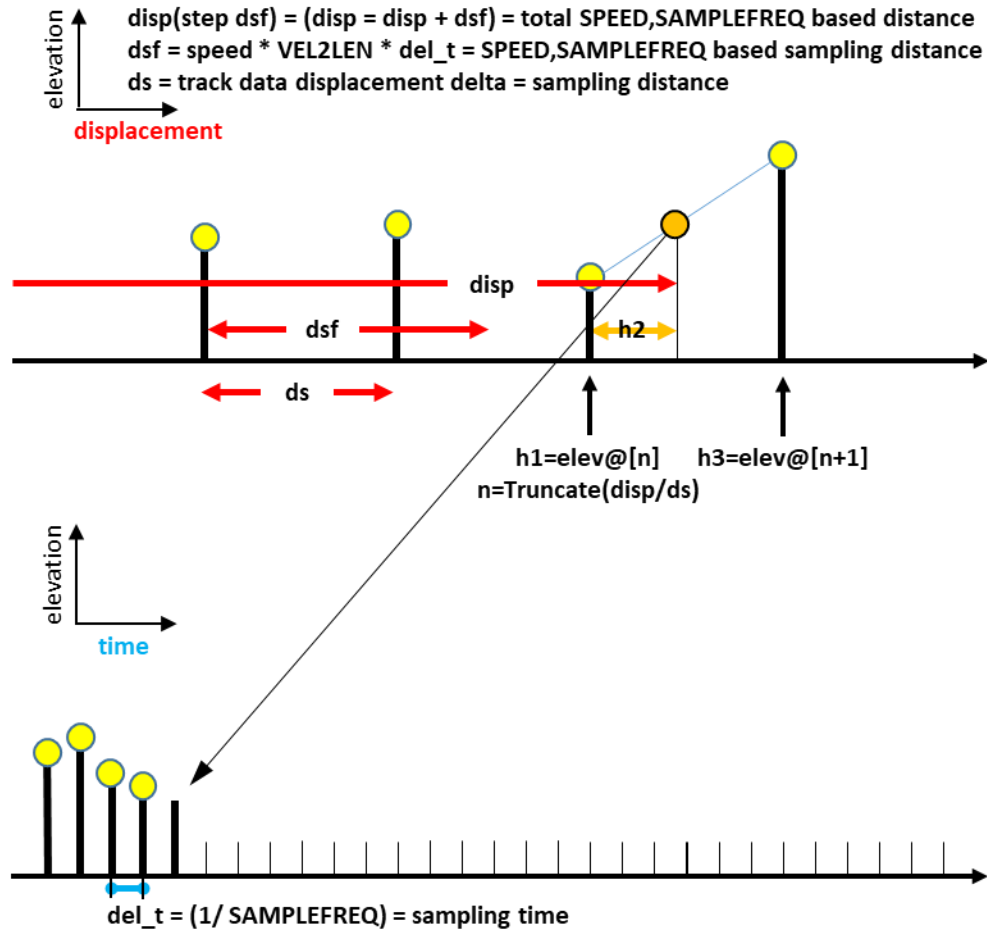


Figure 20 – Conversion from Displacement to Time History Process

$$\frac{1}{dsf} \geq 2 \frac{1}{ds} \rightarrow ds \geq 2dsf$$

(5)

Table 7 – Spatial Sampling Data

Spatial Sampling Distance	Spatial Sampling Frequency	Nyquist Frequency
ds	1/ds	2*(1/ds)
[in]	[1/[in]]	[1/[in]]
0.50	2.000	4.000
1.00	1.000	2.000
3.00	0.333	0.667
12.00	0.083	0.167

Using the alternative Nyquist criteria in Equation (5), Table 8 also illustrates this condition.

Table 8 – Speed Sampling Data

Speed	Speed Based Sampling Distance	Speed Based Sampling Frequency	Nyquist Criterion ($ds > 2 \cdot dsf$)
speedvals	dsf	1/dsf	2*dsf
[MPH]	[in]	[1/[in]]	[in]
5	0.430	2.327	0.859
10	0.859	1.164	1.719
15	1.289	0.776	2.578
20	1.719	0.582	3.438
25	2.148	0.465	4.297
30	2.578	0.388	5.156
35	3.008	0.332	6.016
40	3.438	0.291	6.875
45	3.867	0.259	7.734
50	4.297	0.233	8.594
55	4.727	0.212	9.453
NOTE	SAMPLEFREQ = 204.8 Hz		

This does not mean that the resolution of the course profile data should be reduced. It only illustrates that when this criteria is not met that sample points in the course profile will be skipped as the speed increases (assuming a fixed sampling frequency). It is desirable to have a reasonable resolution for the profile data so when linear interpolation is used to determine the elevation between samples it is an accurate representation. If the profile data sampling resolution is too low, the linear interpolation between samples may not reflect the actual elevation between them.

In conclusion, it should be acknowledged that the scripts included in the Appendix are not considered to be optimized. Different approaches could be taken. Overall, however, it is believed that the essence of drive file creation is captured---particularly in Figure 20.

REFERENCE

- [1] Vehicle N-Post Testing Capabilities (slide 38). GVSC Ground Vehicle Simulation Laboratory. (<http://www.usarmygpsc.com/wp-content/uploads/2019/10/GVSC-LAB-TESTING-CAPABILITIES-BRIEFING-23-JULY19.pdf>) [2020-10-14]. Distribution Statement A. Approved for public release; distribution is unlimited.
- [2] Federal Labs (CCDC GVSC). (<https://federallabs.org/labs/ccdc-ground-vehicle-systems-center>) [2020-10-14]
- [3] US. Army Ground Vehicle Systems Center (<http://www.usarmygpsc.com/>) [2020-10-14]
- [4] Federal Labs (ATC) (<https://federallabs.org/labs/aberdeen-test-center-atc>) [2020-10-14]
- [5] Aberdeen Test Center (<https://www.atec.army.mil/atc/>) [2020-10-14]
- [6] Test Operations Procedure (TOP) 01-1-011B Vehicle Test Facilities at Aberdeen Test Center and Yuma Test Center (12-12-2017). Defense Technical Information Center (DTIC), AD No. AD1043599. (<https://vdl.s.atc.army.mil>) [2020-10-14]. Distribution Statement A. Approved for public release; distribution is unlimited.
- [7] Test Operations Procedure (TOP) 01-1-010A Vehicle Test Course Severity (Surface Roughness) (12-12-2017). Defense Technical Information Center (DTIC), AD No.: AD1043586. (<https://vdl.s.atc.army.mil>) [2020-10-14]. Distribution Statement A. Approved for public release; distribution is unlimited.
- [8] RPC Software (<https://test.mts.com/products/software-monitoring/rpc-software>) [2020-10-14]
- [9] Perl (<https://www.perl.org/>) [2020-10-14]
- [10] Python (<https://www.python.org/>) [2020-10-14]
- [11] ActiveState (<https://www.activestate.com/>) [2020-10-14]
- [12] Powershell (<https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7>) [2020-10-14]
- [13] Bylsma. Spacial and Frequency Domain Calculation of Terrain Roughness Metric Root-Mean-Square (RMS). (TR 26042, November 2014). Defense Technical Information Center (DTIC), AD No. A617210. Distribution Statement A. Approved for public release; distribution is unlimited.
- [14] MATLAB (<https://www.mathworks.com/>) [2020-10-14]
- [15] Hann Function/Raised Cosine Window (https://en.wikipedia.org/wiki/Hann_function) [2020-10-20]
- [16] Composite Formatting (<https://docs.microsoft.com/en-us/dotnet/standard/base-types/composite-formatting>) [2020-10-20]
- [17] Numeric Format (<https://docs.microsoft.com/en-us/dotnet/standard/base-types/standard-numeric-format-strings#DFormatString>) [2020-10-20]
- [18] Nyquist Frequency (https://en.wikipedia.org/wiki/Nyquist_frequency) [2020-10-22]
- [19] .NET Objects List, Dictionary, StreamReader, StreamWriter (<https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=netcore-3.1>, <https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=netcore-3.1>, <https://docs.microsoft.com/en-us/dotnet/api/system.io.streamreader?view=netcore-3.1>, <https://docs.microsoft.com/en-us/dotnet/api/system.io.streamwriter?view=netcore-3.1>)

APPENDIX A.1 – CPC TESTS

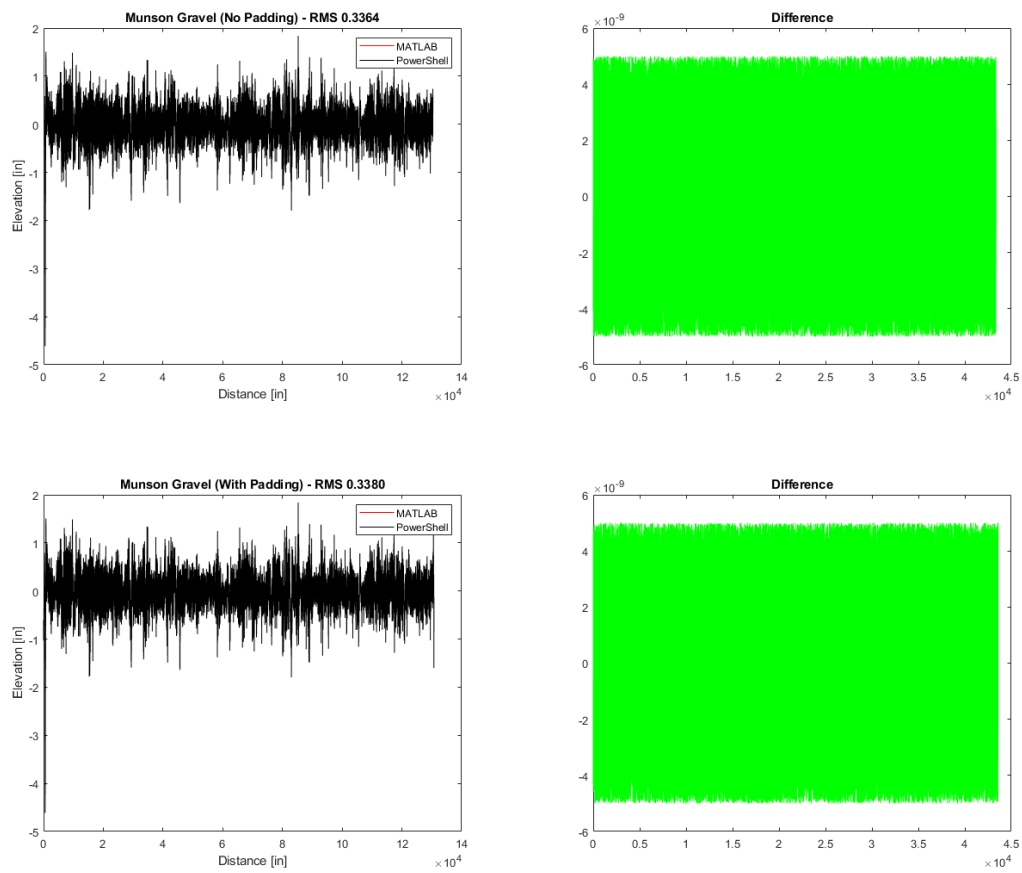


Figure A1.1 – Comparison of Detrending (w/wo Padding) with MATLAB vs CPC

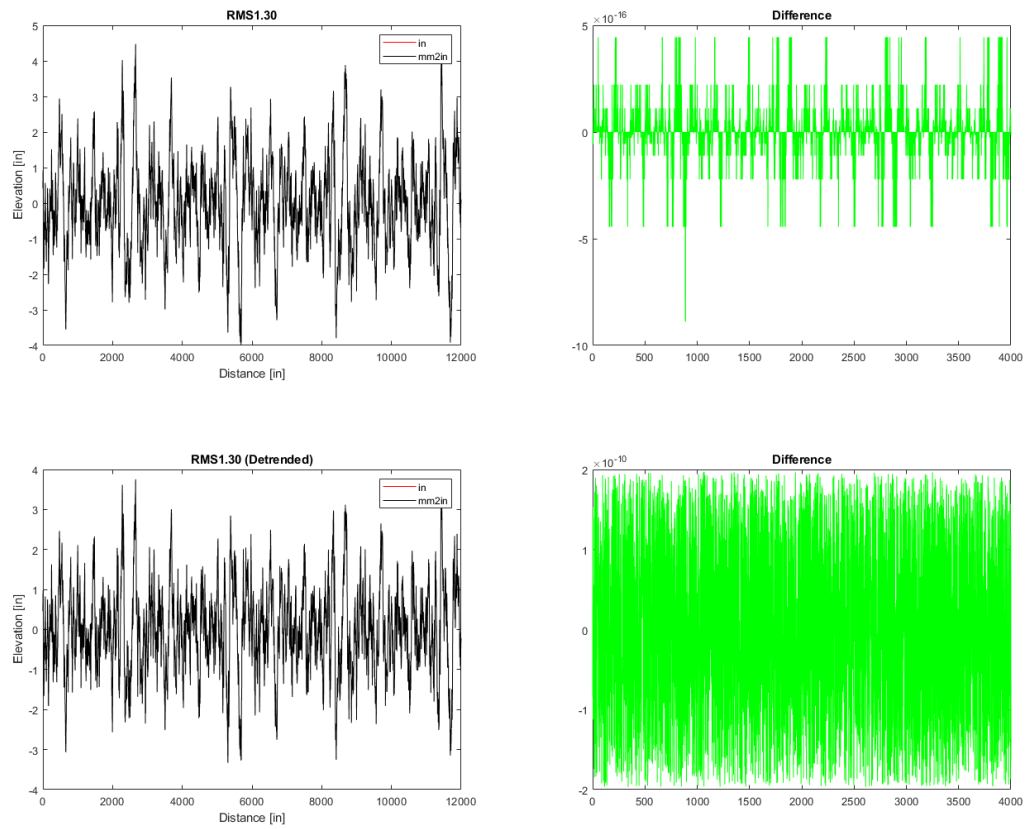


Figure A1.2 – Comparison of “in to m” Units w/wo Detrending

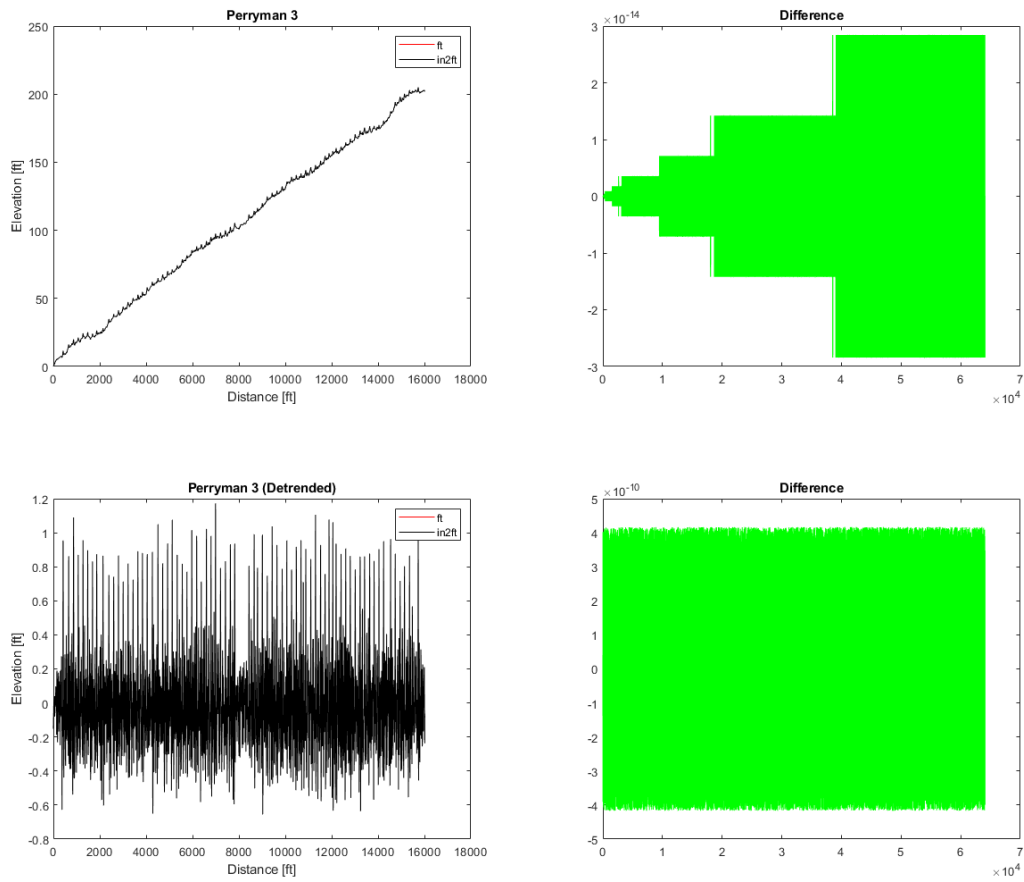


Figure A1.3 – Comparison of “ft to in” Units w/wo Detrending

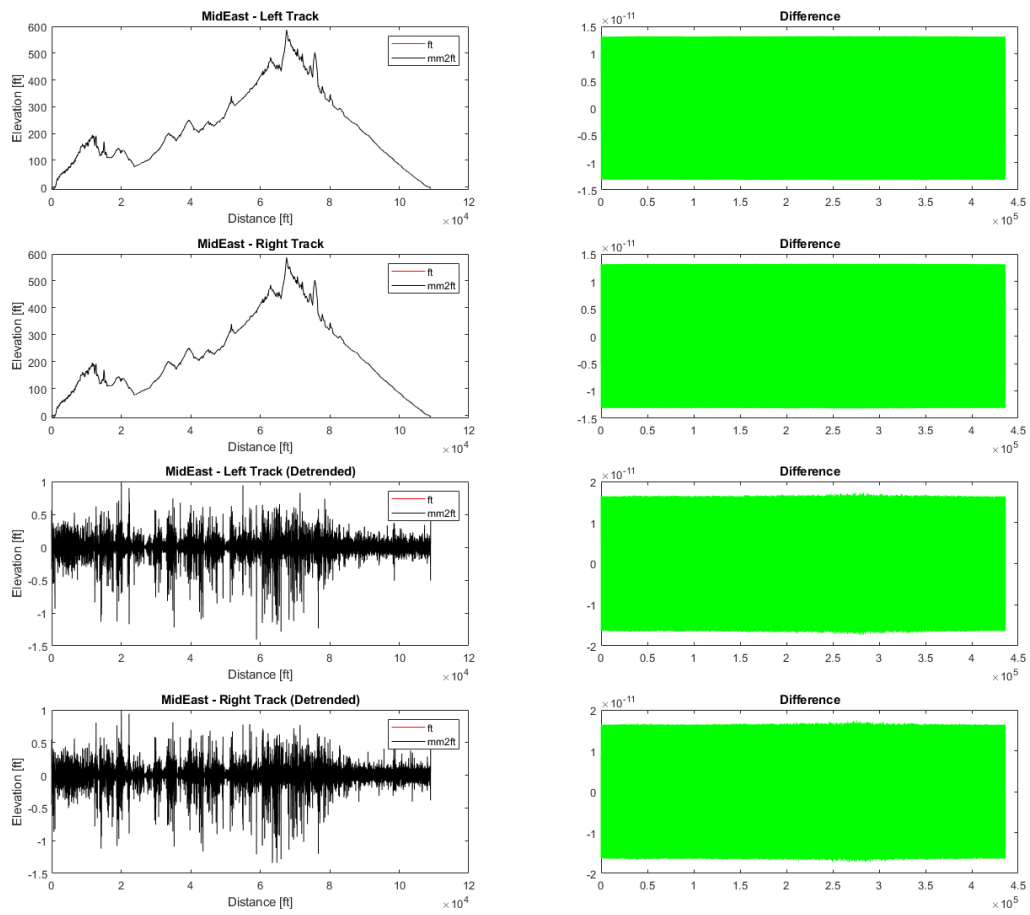


Figure A1.4 – Comparison of Left/Right Track “ft to mm” Units w/o Detrending

APPENDIX A.2 – DFC TESTS

Original 2011 Python script channel time history output files were used for comparisons

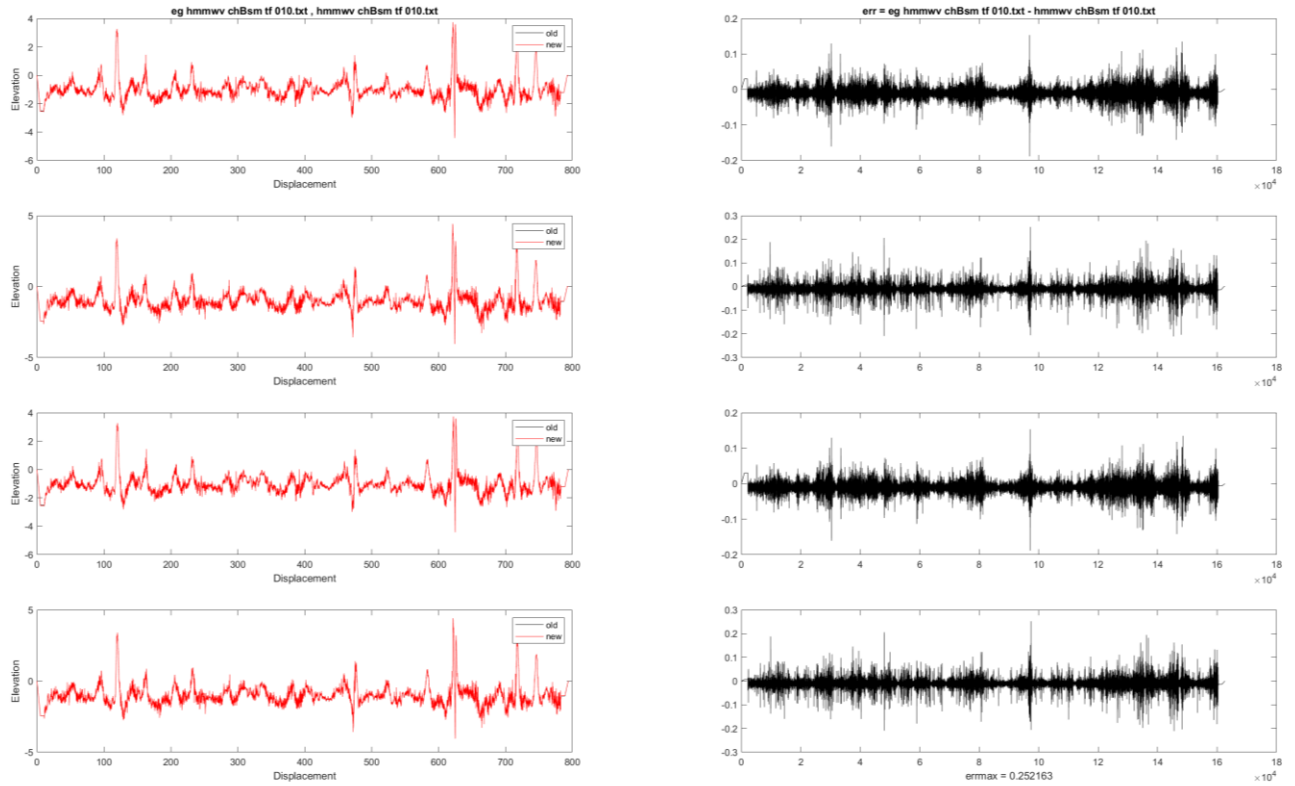


Figure A2.1 – Comparison HMMWV, Churchville B Smooth, 10 MPH

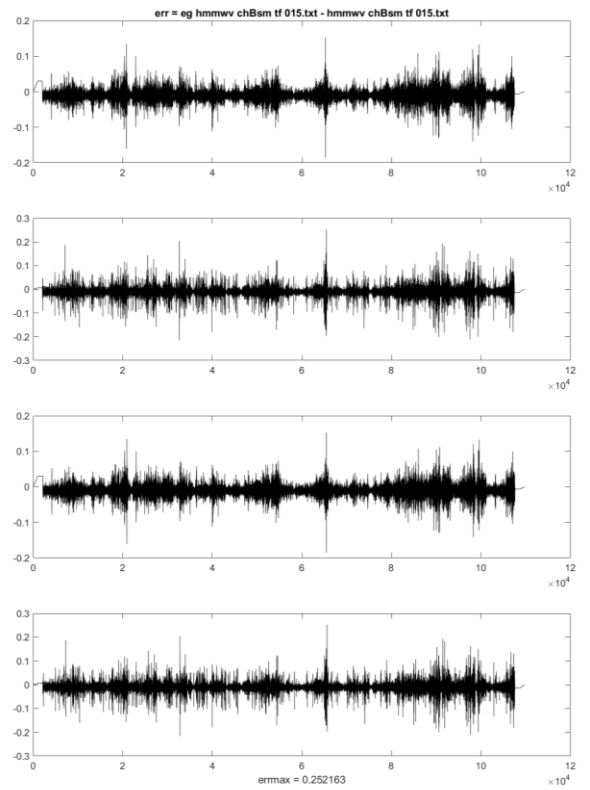
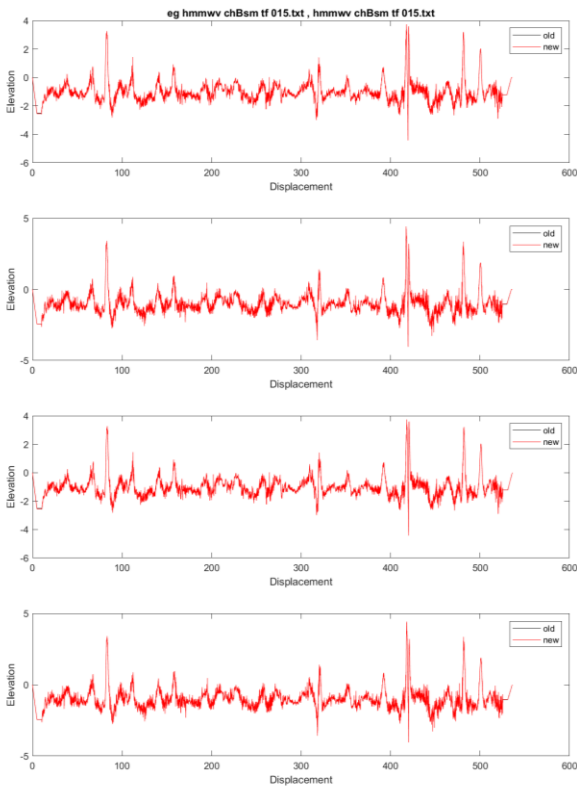


Figure A2.2 – Comparison HMMWV, Churchville B Smooth, 15 MPH

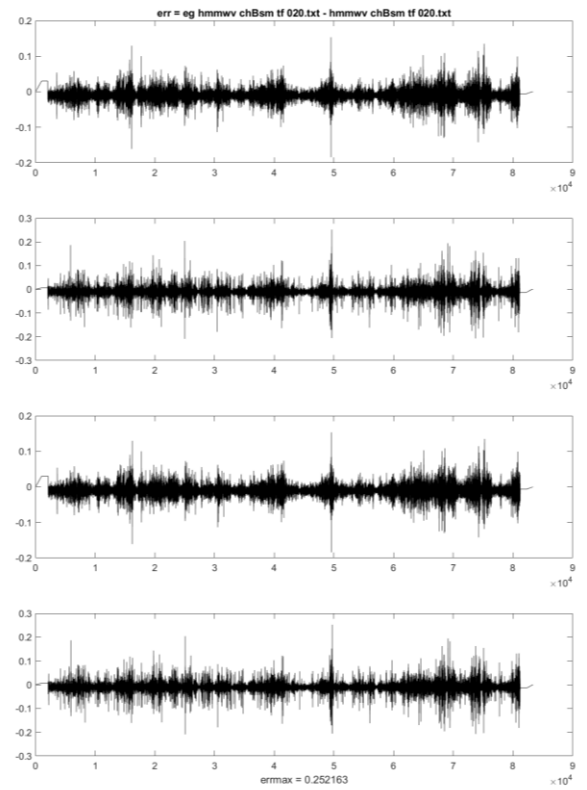
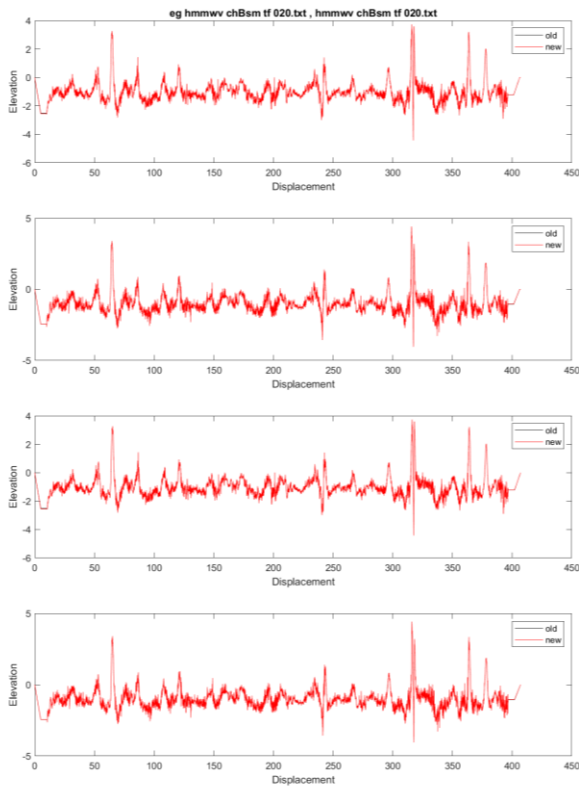


Figure A2.3 – Comparison HMMWV, Churchville B Smooth, 20 MPH

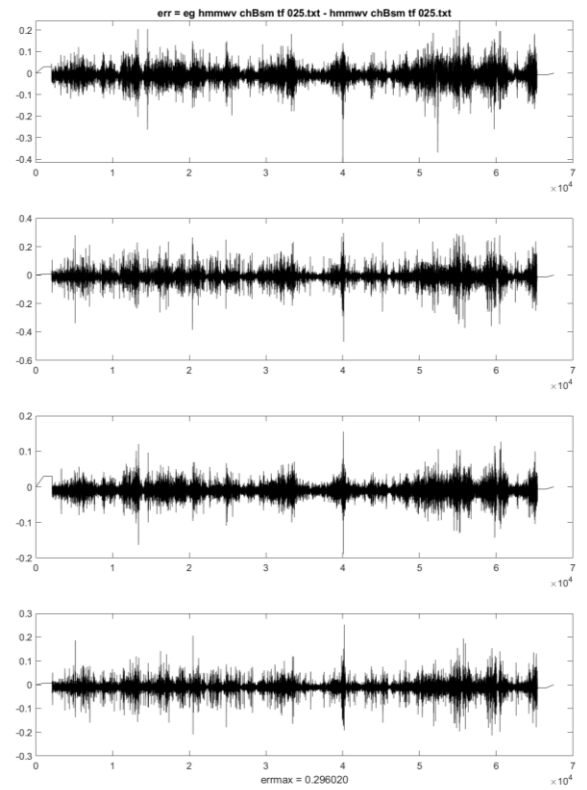
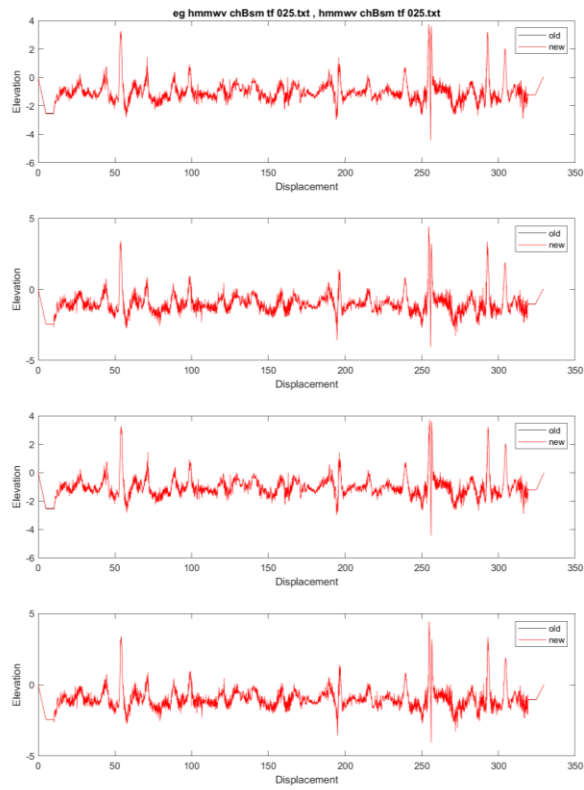


Figure A2.4 – Comparison HMMWV, Churchvile B Smooth, 25 MPH

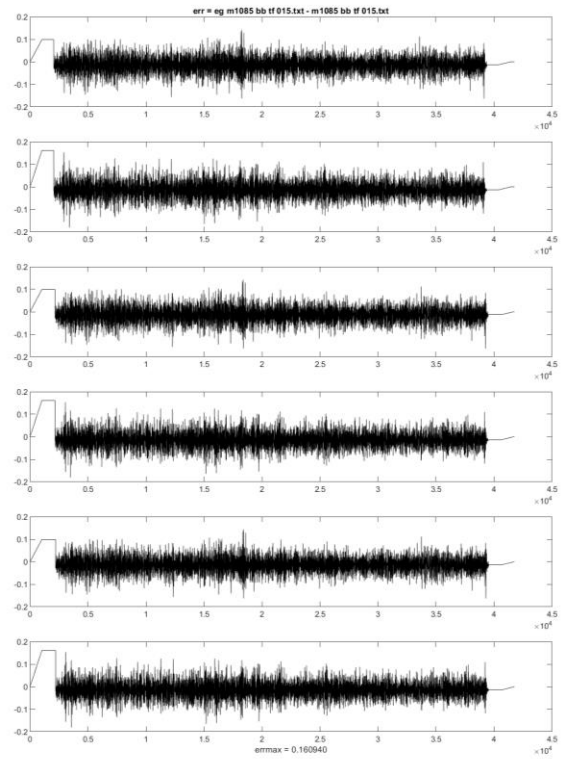
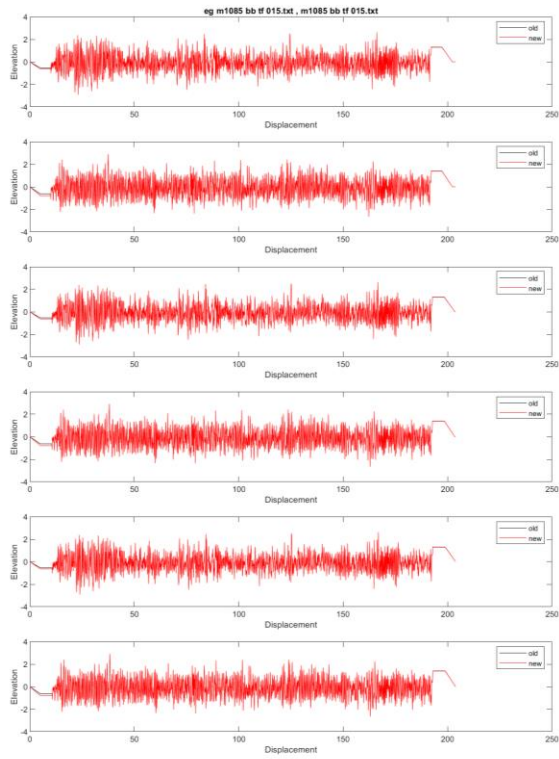


Figure A2.5 – Comparison M1085, Belgian Block, 15 MPH

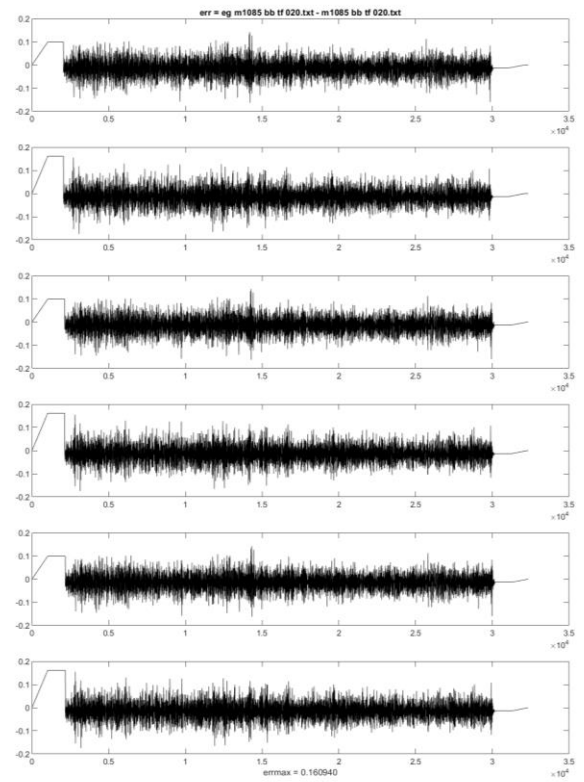
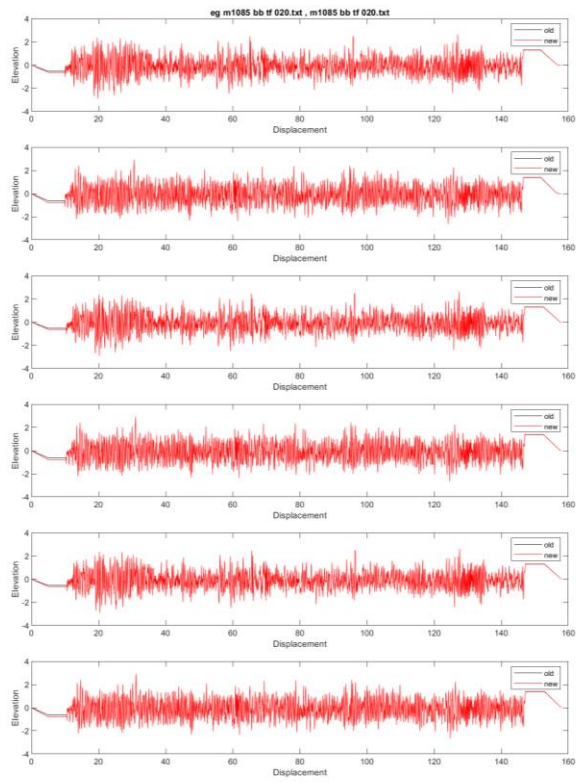


Figure A2.6 – Comparison M1085, Belgian Block, 20 MPH

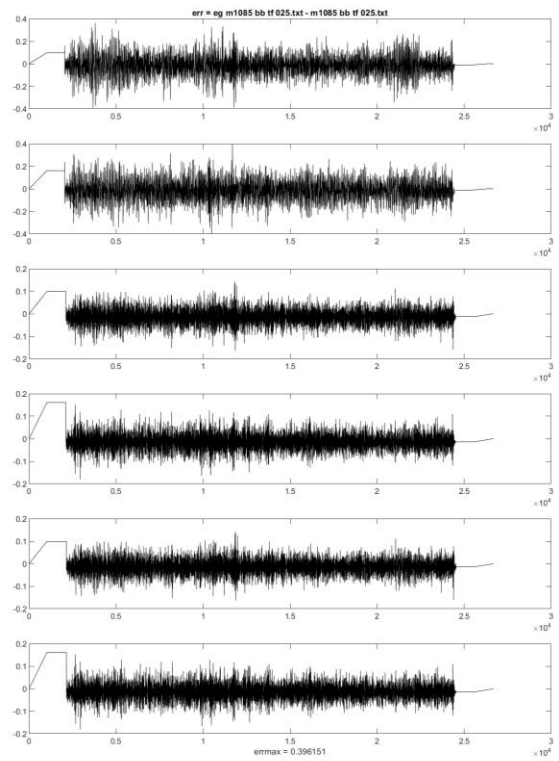
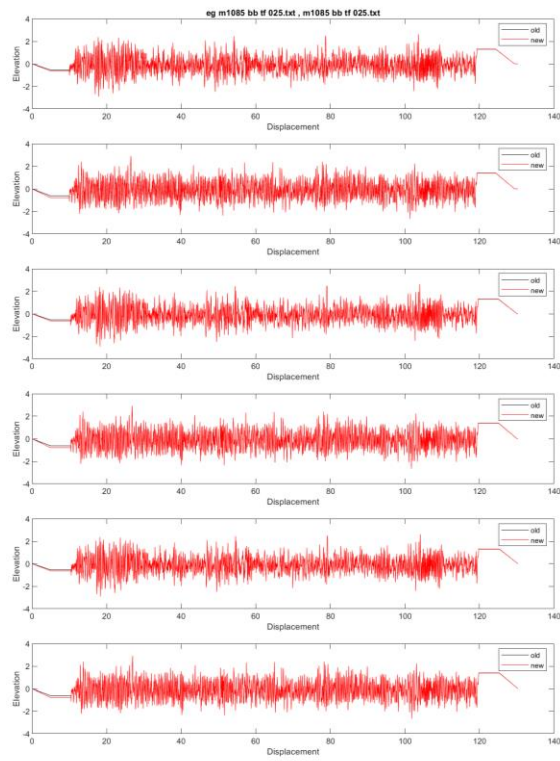


Figure A2.7 – Comparison M1085, Belgian Block, 25 MPH

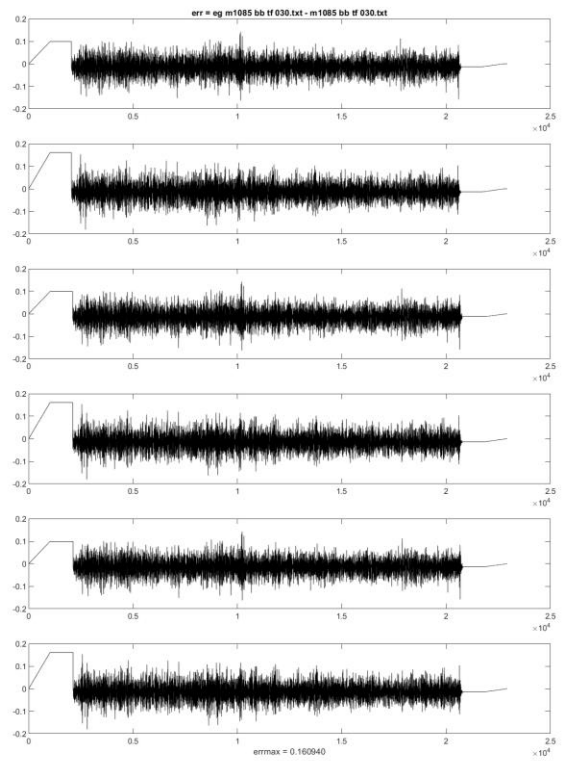
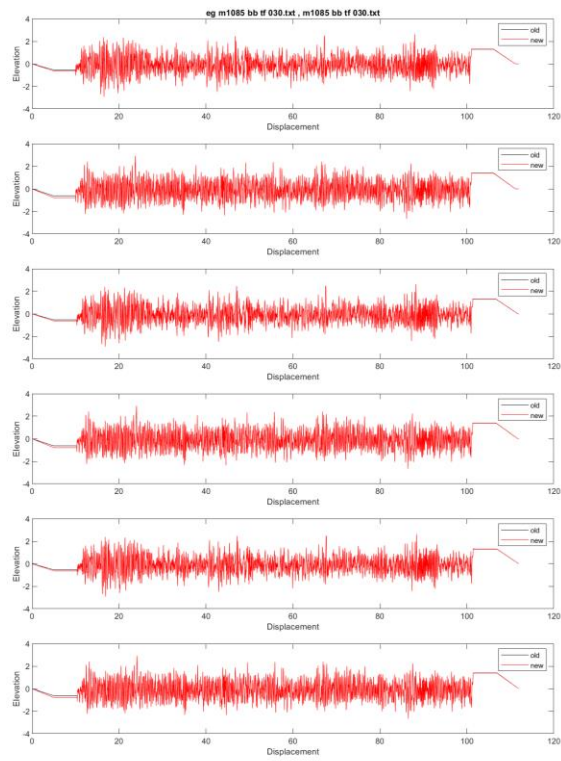


Figure A2.8 – Comparison M1085, Belgian Block, 30 MPH

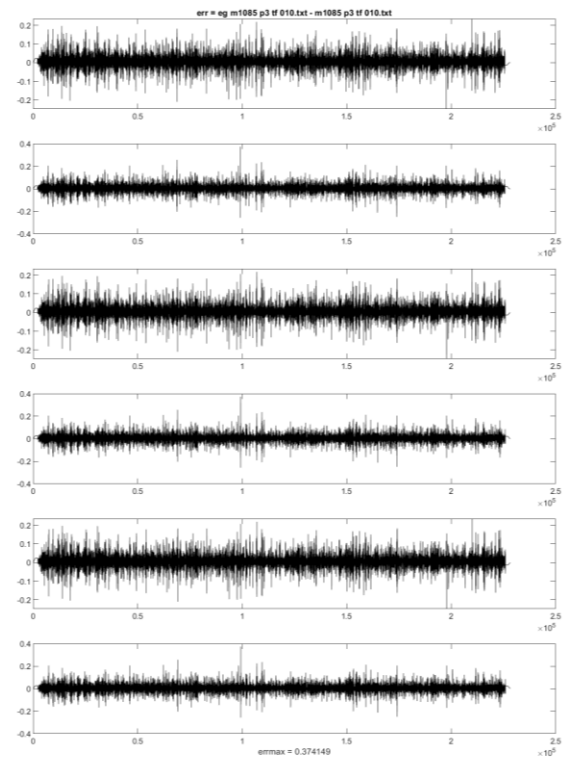
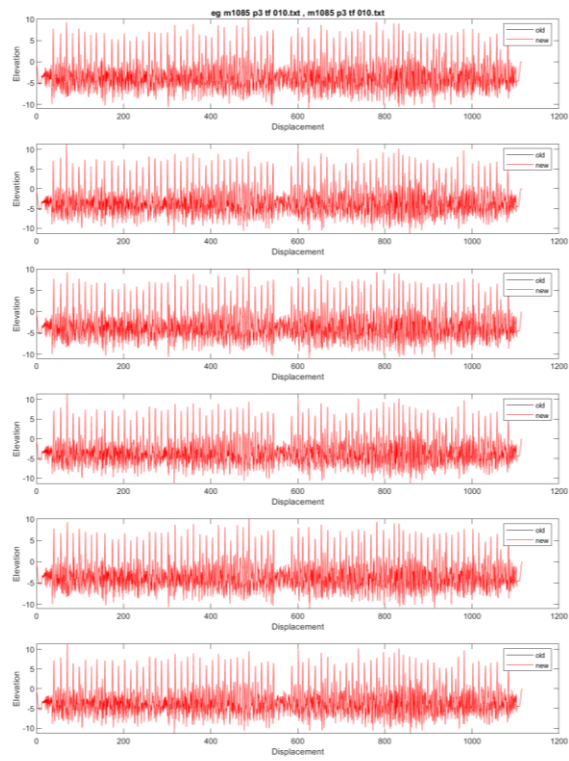


Figure A2.9 – Comparison M1085, Perryman 3, 10 MPH

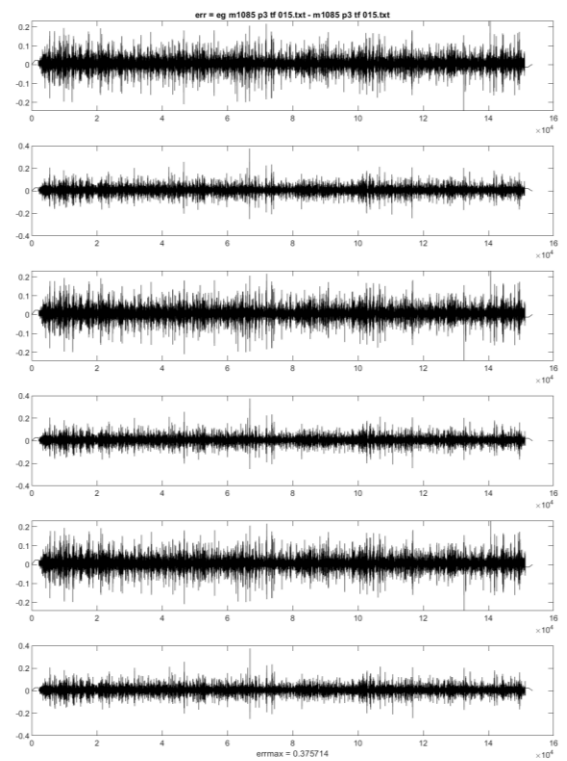
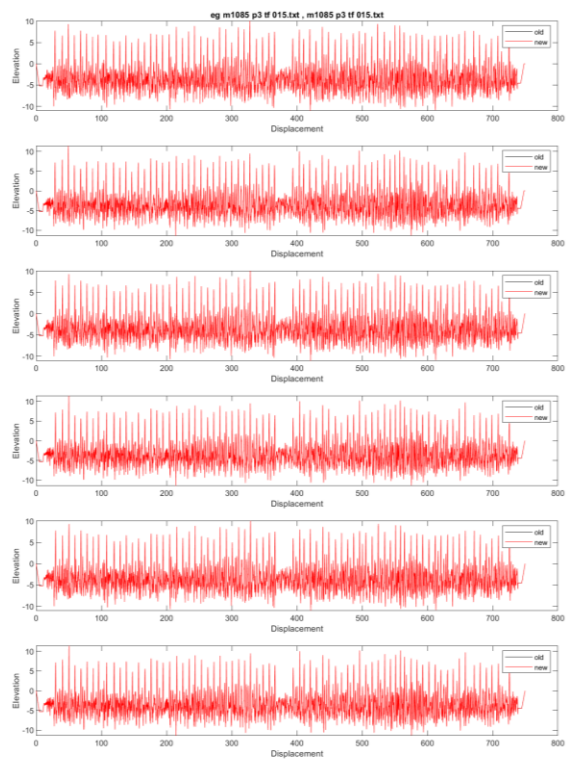


Figure A2.10 – Comparison M1085, Perryman 3, 15 MPH

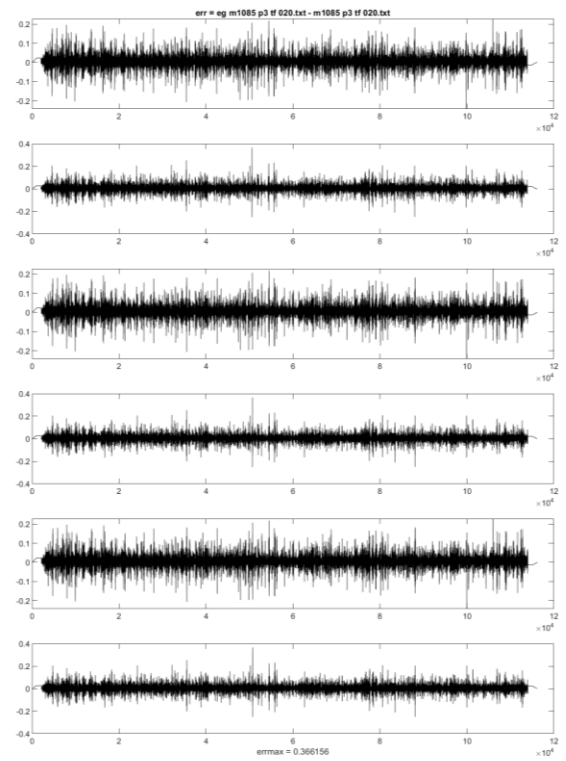
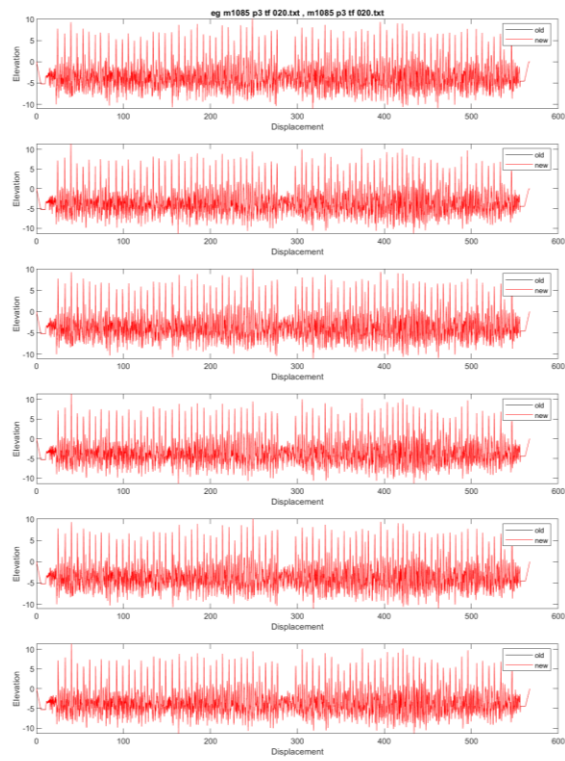


Figure A2.11 – Comparison M1085, Perryman 3, 20 MPH

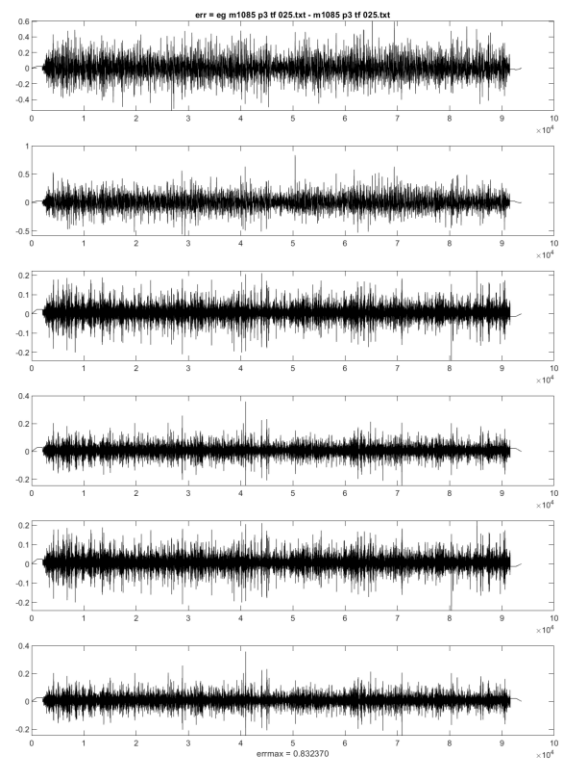
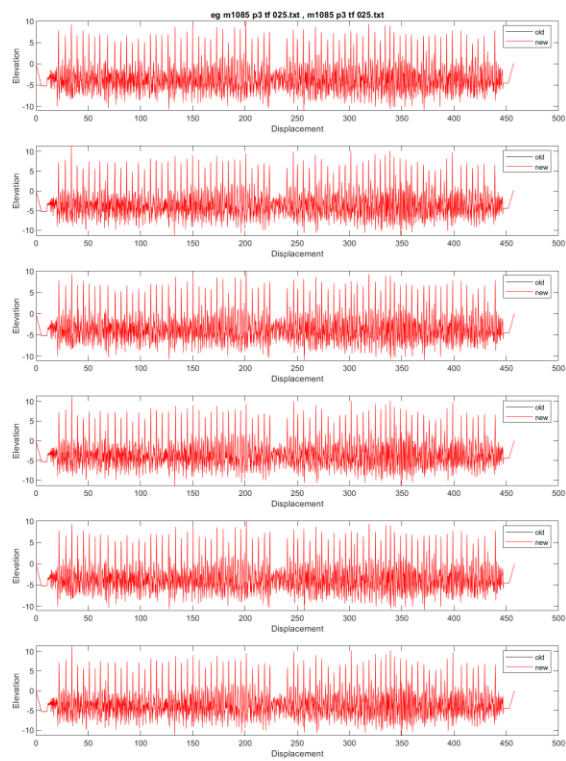


Figure A2.12 – Comparison M1085, Perryman 3, 25 MPH

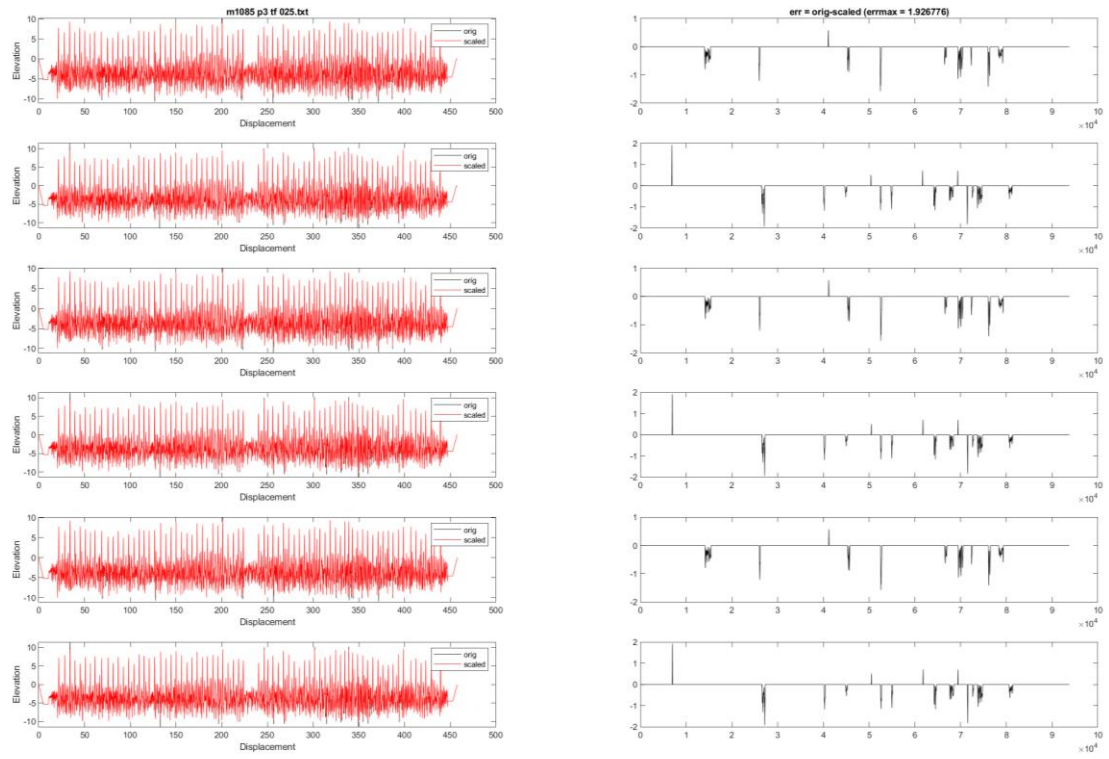


Figure A2.13 – Comparison between Peak Scaling for m1085_p3_tf_025

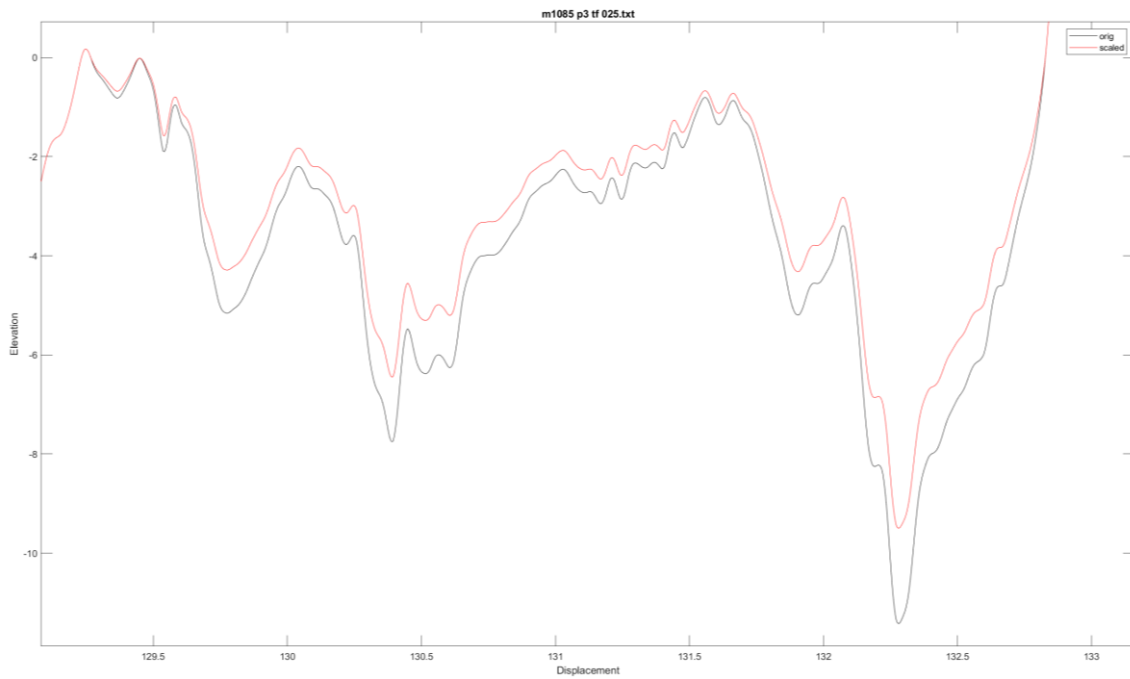


Figure A2.14 – Zoomed Comparison between Peak Scaling for m1085_p3_tf_025

```

v las w1005 p3 tf 005.txt

```

n1005 p3 rf 025.txt

APPENDIX A.3 – CPC SCRIPT

Powershell Environment

```
Name                               Value
----                               -
PSVersion                          5.1.17763.1432
PSEdition                          Desktop
PSCompatibleVersions               {1.0, 2.0, 3.0, 4.0...}
BuildVersion                       10.0.17763.1432
CLRVersion                         4.0.30319.42000
WSManStackVersion                  3.0
PSRemotingProtocolVersion          2.3
SerializationVersion               1.1.0.1

===BOF
<#
.SYNOPSIS
Converts course profile data to a detrended single track data file with unit conversion.

.DESCRIPTION
Course Profile Conversion (CPC) PowerShell script reads course profile data in multiple file
formats and outputs detrended single track data (left and/or right) after unit conversion
using a double-sided exponential filter. The input file name and format type are specified
along with a variable number of parameters depending upon the file format type.

***The supplied unit conversion factors for each type of file format should ensure that the
resulting units for DISPLACEMENT and ELEVATION are the SAME***

***If no detrending filter is specified (detrending is not desired), a warning message is given
as CPC is intended to prepare data to create drive files for VEHICLE SHAKER (N-POST) test
equipment which requires the data be detrended (limited actuator displacement). Not detrending
in CPC will allow an external detrending method to be used.

.INPUTS

===COURSE PROFILE FILE FORMATS

-Type 1 (single track, one column)
[comment lines]
[number of elevation samples, displacement per sample]
[elevation samples]

EXAMPLES
===BOF
apg_IV 2.1942 RMS
APGIV 468 FEET LONG, PROFILE 4 ( 2.194 in. rms ) L=468 ft
    469      12.
    .00      .84    1.68    2.52    3.36    4.08    4.92    5.76    6.60    7.44
    8.28    9.12    9.96    10.80   11.64   11.40   10.32   9.12    8.04    6.84
    5.76    4.56    3.48    2.28    1.20     .00     .00     .00     .00     .00
    .
    .
    .
===EOF

===BOF
RMS1.30-23.txt /magnitude,dx in inches
4000, 3.000000
0.427780
0.779344
0.938570
.
.
.
===EOF

PS> .\cpc <ifn> <ft> (<nocl>, <cf1>, <cf2>, <ofn1>, [fd1s])
PS> .\cpc <input_file_name(ifn)> <format_type(ft)> (<number_of_comment_lines(nocl)>,
    <conversion_factor1_for_displacement_per_sample(cf1)>,
    <conversion_factor2_for_elevation_column_1(cf2)>, <output_file_name(ofn1)>,
    [filter_distance_one_side(fd1s)])

-Type 2 (single track, two columns)
[comment lines]
[displacement samples, elevation samples]

EXAMPLE
===BOF
    0.0000      0.0834
    0.2500      0.0836
    0.5000      0.0808
    .
    .
    .
===EOF

PS> .\cpc <ifn> <ft> (<nocl>, <cf1>, <cf2>, <ofn1>, [fd1s])
```

```

PS> .\cpc <input_file_name(ifn)> <format_type(ft)> (<number_of_comment_lines(nocl)>,
<conversion_factor1_for_displacement_column_1(cf1)>,
<conversion_factor2_for_elevation_column_2(cf2)>, <output_file_name(ofn1)>,
[filter_distance_one_side(fdls)])

-Type 3 (multiple track, multiple columns)
[comment lines]
[displacement samples, elevation 1 samples, elevation 2 samples, ...]

EXAMPLE
===BOF
Distance (ft),Laser 1 (ft),Laser 2 (ft),Laser 3 (ft),Laser 4 (ft),Laser 5 (ft)
0.0000000,0.03423493,0.10571433,0.08979320,-0.24436195,-0.48479081
0.2500000,0.03943786,0.08687522,0.06247506,-0.20190297,-0.52107400
0.5000000,0.01217997,0.07665324,0.02443129,-0.27996569,-0.55725897
.
.
.
===EOF

PS> .\cpc <ifn> <ft> (<nocl>, <cf1>, <cf2>, <cf3>, <lc1>, <rc1>, <ofn1>, <ofn2>, [fdls])
PS> .\cpc <input_file_name(ifn)> <format_type(ft)> (<number_of_comment_lines(nocl)>,
<conversion_factor1_for_displacement_column_1(cf1)>,
<conversion_factor2_for_elevation_left_track(cf2)>,
<conversion_factor3_for_elevation_right_track(cf3)>, <left_track_column_number(lc1)>,
<right_track_column_number(rc1)>, <left_track_output_file_name(ofn1)>,
<right_track_output_file_name(ofn2)>, [filter_distance_one_side (fdls)])

===DETRENDING FILTER DISTANCE (fdls)
The "filter_distance_one_side (fdls)" parameter defines the one-sided distance of the exponential
filter. This filter is also used before calculating roughness in root-mean-square (RMS). The
standard distance is 30 feet [1] on each side of the sample point, giving a total filter distance
of 60 feet (+/-30 feet). The appropriate value to use for fdls is to convert 30 feet into the
desired units which should be the same units of length after the conversion_factors are applied
(see cf1, cf2, cf 3).

EXAMPLE
UNITS    fdls (30 feet)
ft       30
in       360
m        9.144
mm       9144

***The sample distance is also required for the detrending filter but CPC determines it from the
course profile data provided.

.OUTPUTS
Output is a detrended single track data file whose format is two columns, displacement and
elevation data, respectively. Depending on the file format type, separate left and right
detrended single track data files may be generated (see Type 3).

(repeated from .INPUTS)
***The supplied unit conversion factors for each type of file format should ensure that the
resulting units for DISPLACEMENT and ELEVATION are the SAME***

***If no detrending filter is specified (detrending is not desired), a warning message is given
as CPC is intended to prepare data to create drive files for VEHICLE SHAKER (N-POST) test
equipment which requires the data be detrended (limited actuator displacement). Not detrending
in CPC will allow an external detrending method to be used.

.EXAMPLE
#=====Type 1 (in to mm)

PS> $params = @(1, 25.4, 24.5, "crse1_mm_d.cpc", 9144)
PS> .\cpc .\data\crse1.txt 1 @params

#=====Type 2 (ft to in)

PS> $params = @(0, 12.0, 12.0, "crse2_in_d.cpc", 360)
PS> .\cpc .\data\crse2.txt 2 @params

#=====Type 3 (ft to mm)

PS> $params = @(1, 304.8, 304.8, 304.8, 6, 2, "crse3L_mm_d.cpc", "crse3R_mm_d.cpc")
PS> .\cpc .\data\crse3.txt 3 @params

.NOTES
===COLUMN NUMBERS
Column numbers for course profile data begin with the number one (1) (they are zero-based only
internally to CPC).

===COURSE PROFILE DATA===
Detrending of each single track data file (left or right) is done separately, except for file
format type 3 where both left and right track data is provided.

===HISTORY===
2020-09-24 Detrending added
2020-09-21 Powershell script created

===REFERENCES===
[1] TR26042 "Spectral and Frequency Domain Calculation of Terrain Roughness Metric
Root-Mean-Square (RMS)" (Nov 2014). Distribution A (DTIC AD A617 210).

```

```

[2] TN10-0729-01 "Exponential Course Detrending" (29 Jul 2010) in TR21344 "2010
    Analytics/Dynamics Technical Notes and Activities Report (Dec 2010). Distribution B
    (DTIC AD B366 068).
[3] .NET Objects List, Dictionary, StreamReader, StreamWriter (https://docs.microsoft.com/
    en-us/dotnet/api/system.collections.generic.list-1?view=netcore-3.1, https://docs.
    microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view
    =netcore-3.1, https://docs.microsoft.com/en-us/dotnet/api/system.io.streamreader?view=
    netcore-3.1, https://docs.microsoft.com/en-us/dotnet/api/system.io.streamwriter?view=
    netcore-3.1
[4] PowerShell (https://docs.microsoft.com/en-us/powershell/scripting/overview?view=
    powershell-7)

===PowerShell/.NET Notes===
[1] For file positioning with StreamReader:
    $r_file = New-Object IO.StreamReader( <path> )
    $r_file.DiscardBufferedData()
    write-host $r_file.BaseStream.CanSeek
    write-host $r_file.BaseStream.Position
    $r_file.BaseStream.Seek(0, [System.IO.SeekOrigin]::Begin) # rest to beginning
[2] For number format settings:
    [System.Globalization.NumberFormatInfo]::CurrentInfo.NumberDecimalDigits
    write-host ( "{0:N}" -f 4.23 )
[3] Generic.List .ForEach delegate function:
    $data = New-Object System.Collections.Generic.List[double[]] # items are double[]
    $data.foreach(
        { param( $p )
          ( "{0,-15:F8}" -f $p[0] ) + " " + ( "{0,-15:F8}" -f $p[1] )
        }
    )
[4] Generic.List .Sort delegate function
    # Default sorting, ascending
    # $a < $b -1
    # $a = $b 0
    # $a > $b 1
    $data = New-Object System.Collections.Generic.List[double[]] # items are double[]
    $data.Sort(
        { param($a, $b)
          if ($a[0] -lt $b[0]) { return 1 } # descending
          if ($a[0] -eq $b[0]) { return 0 }
          else { return -1 }
        }
    )
#>

param ($ifn, $ft)

#=====FUNCTIONS=====
function rmsexp( $x, $N ) {
<#
    Double-sided exponential weighted filter (standard is NA=30feet, NA/lambda=3)

INPUTS:
$x      track data to filter
$N      filter distance (one side), passed into rmsexp as $fdls
        The standard distance is 30 feet[1] on each side of the sample point, giving a total
        filter distance of 60 feet (+/-30 feet). The appropriate value to use for $N
        (see $fdls) is to convert 30 feet into the desired units which should be the same
        units of length after the conversion_factors are applied (see cf1, cf2, cf3).

EXAMPLE
UNITS   N (30 feet, see fdls)
ft      30
in      360
m       9.144
mm      9144

        The sample distance is also required for the detrending filter but CPC determines it
        from the course profile data provided.
        ([double] $A = $x[1][0] - $x[0][0] # sample distance from input data)

RETURNS:
$fdet   detrended terrain
$rms     rms value
$fmean   trend removed
$xmean   mean of fdet

VARIABLES:
$A       sample distance
$L       end sample number
$lambda  exponential factor (-LA/lambda=3)
$det1    exponential filter coefficients
$aa      initialization array
$dsum     sum of coefficients for normalization
$fdet     detrended data
$fmean    mean of track data
$ie       end index
$indx     offset into filter coefficients
$jb       beginning index
$je       end index
$xmean    mean of detrended data
$rms      root-mean-square value
$fdetnp   detrended data, no padding
$fmeannp  mean of track data, no padding
$pad      pad data before filtering, then length same as original
#>

```

```

$pad = $true    # internal flag to turn on/off input data padding

[double] $A = $x[1][0] - $x[0][0]    # sample distance from input data
[int] $L = [Math]::Truncate( $N / $A )    # end sample number
[double] $lambda = ($L * $A) / 3    # exponential factor

write-host "    Detrending Filter Parameters:"
# output parameters
( ("A", $A),
  ("L", $L),
  ("lambda", $lambda) ) | foreach {

    write-host ("    {0,-6}:" -f ${$_[0]}), ${ $_[1].ToString()}

}

$det1 = New-Object System.Collections.Generic.List[double]    # filter data
[ double[] ] $aa = @( 0 ) * ($L+1)
$det1.AddRange( $aa )    # initialize

# define filter coefficients
$dsum = 0.0

foreach ( $i in (0..$L) ) {

    $det1[$i] = [Math]::exp( -($i * $A) / $lambda )    # exponential weights
    $dsum = $dsum + $det1[$i] * 2    # *2 = double sided when applied

}

$sum = 0

# normalize coefficients
foreach ( $i in (0..$L) ) {

    $det1[$i] = $det1[$i] / $dsum
    $sum = $sum + $det1[$i]

}

# define data for filtering (pad with filter distance (L=1 side) to either end if $pad=$true)
$fdet = New-Object System.Collections.Generic.List[double]    # detrended track data

foreach ( $i in (1..$L) ) {

    if ($pad) {

        $fdet.Add( $x[0][1] )    # initial value, padding

    }

}

$ie = $x.count - 1    # original input data length

foreach ( $i in (0..$ie) ) {

    $fdet.Add( $x[$i][1] )    # track data

}

foreach ( $i in (1..$L) ) {

    if ($pad) {

        $fdet.Add( $x[$ie][1] )    # end value, padding

    }

}

# find average over filter
$fmean = New-Object System.Collections.Generic.List[double]    # mean data
[ double[] ] $aa = @( 0 ) * ($fdet.count)
$fmean.AddRange( $aa )    # initialize

$xmean = 0.0
$ie = $fmean.count - 1 - $L    # start/end sample point at input data

foreach ( $i in ($L..$ie) ) {

    $indx = $i - $L
    $jb = $i
    $je = $i + $L

    foreach ( $j in ($jb..$je) ) {

        # $j gives $i to $i + $L
        # $j - $i gives 0 to $L
        # $i - ( $j - $i ) gives $i to $i - $L
        $fmean[$i] = $fmean[$i] + ( $fdet[$j] * $det1[$j - $i] ) + ( $fdet[$i - ( $j - $i )] *
            $det1[$j - $i] )

    }

}

# remove trend and find total mean (over all data)

```

```

foreach ( $i in ($L..$ie) ) {

    $fdet[$i] = $fdet[$i] - $fmean[$i]
    $xmean = $xmean + $fdet[$i]

}

$xmean = $xmean / ( $ie - $L + 1 )

# remove mean/calc rms
$rms = 0.0

foreach ( $i in ($L..$ie) ) {

    $fdet[$i] = $fdet[$i] - $xmean
    $rms = $rms + ( $fdet[$i] * $fdet[$i] ) # xmean = 0, already removed

}

# divide by count, best unbiased estimator by (count - 1)
$rms = [Math]::sqrt( $rms / ( $ie - $L + 1 ) ) # equivalent to /( $fdet.count - 2*$L )

@"(RMS", $rms) | foreach {

    write-host ( "    {0,-6}:" -f ${$_[0]}), ${$_[1].ToString()}

}

$fdetnp = New-Object System.Collections.Generic.List[double[]] # track data, no padding
$fmeannp = New-Object System.Collections.Generic.List[double[]] # track mean data, no padding

foreach ( $i in ($L..$ie) ) {

    if ($pad) {

        # pad
        $fdetnp.Add( @( $x[$i-$L][0], $fdet[$i] ) )
        $fmeannp.Add( @( $x[$i-$L][0], $fmean[$i] ) )

    } else {

        # no pad
        $fdetnp.Add( @( $x[$i][0], $fdet[$i] ) )
        $fmeannp.Add( @( $x[$i][0], $fmean[$i] ) )

    }

}

return ($fdetnp, $rms, $fmeannp, $xmean)

}

#####

#=====MAIN=====
<#
CPC Entry Point

INPUTS:
$ifn    file name
$ft     file type (1, 2, or 3)
        1=<ifn> <ft> (<nocl>, <cf1>, <cf2>, <ofn1>, [fdls]) (single track, one column)
        2=<ifn> <ft> (<nocl>, <cf1>, <cf2>, <ofn1>, [fdls]) (single track, two columns)
        3=<ifn> <ft> (<nocl>, <cf1>, <cf2>, <cf3>, <lcn>, <rcn>, <ofn1>, <ofn2>, [fdls])
          (multiple track, multiple columns)

$args
$nocl   number of comment lines
$cf1    conversion_factor 1 (elevation/displacement)
$cf2    conversion_factor 2 (displacement per sample/elevation)
$cf3    conversion_factor 3 (elevation)
$lcn    left column number
$rcn    right column number
$ofn1   output file name/left track output file name
$ofn2   right track output file name

VARIABLES:
$pth    full path name to input file
$r_file read file object
$a      hold line/line split
$ns     number of elevation samples
$sd     sample distance
$cnt    sample counter
$hold   unit converted track data/detrended track data
$elev   elevation sample
$rms    root-mean-square of detrended track data
$fmean  trend of track data
$xmean  mean of detrended track data
$ie     last element index
$pad    hold displacement, elevation sample
$disp   displacement sample
$holdl  unit converted left track data
$holdr  unit converted right track data
$lelev  left elevation sample
$relev  right elevation sample
$rmsl   root-mean-square of left detrended track data
$fmeanl trend of left track data

```

```

$emeanl    mean of left detrended track data
$rmsr      root-mean-square of right detrended track data
$femanr    trend of right track data
$smeanr    mean of right detrended track data
#>

write-host "`nCPC v8.0 2020`n"

write-host "`n***Units for DISPLACEMENT and ELEVATION should be the SAME!!!`n"
write-host "Reading $ifn..."

# test if input file exists
if ( Test-Path $ifn ) {

    $pth = ( get-childitem $ifn ).FullName
    write-host "pth:$pth"
    $r_file = New-Object IO.StreamReader( $pth )

} else {

    write-host "Cannot find file [$ifn]"
    exit

}

$ft = [int]::Parse( $ft ) # file format type

if ( ($ft -ne 1) -and ($ft -ne 2) -and ($ft -ne 3) ) {

    write-host "File Format Type not defined"
    exit

}

write-host "    type: $ft`n"

# determine file type
switch ( $ft ) {

<#
- Type 1 (single track, one column)
[comment lines]
[number of elevation samples, displacement per sample]
[elevation samples]
PS> .\cpc <ifn> <ft> (<nocl>, <cf1>, <cf2>, <ofn1>, [fdls])
#>
1 {

    if ( $args.length -lt 4 ) {

        write-host "Format Type 1 requires 4 or 5 parameters (nocl, cf1, cf2, ofn1, [fdls])"
        exit

    } else {

        # get parameters
        $nocl = [int]::Parse( $args[0] ) # number of comment lines
        $cf1 = [double]::Parse( $args[1] ) # conversion factor 1 (displacement per sample)
        $cf2 = [double]::Parse( $args[2] ) # conversion factor 1 (elevation)
        $ofn1 = $args[3] # output file name

        if ( $args.length -eq 5 ) {

            $fdls = [double]::Parse( $args[4] ) # filter distance one side for detrending

        } else {

            $fdls = 0.0 # no detrending
            write-host "`n***NO DETRENDING FILTER SPECIFIED, Press Ctrl-C to Cancel`n"
            Start-Sleep -s 10

        }

        # output parameters
        ( ("nocl", $nocl),
          ("cf1", $cf1),
          ("cf2", $cf2),
          ("ofn1", $ofn1),
          ("fdls", $fdls) ) | foreach {

            write-host ( "    {0,-4}:" -f $_[0]), $_[1].ToString()

        }

        #===read file loop:beg

        # read comment lines
        while ( $nocl -gt 0 ) {

            $a = $r_file.ReadLine()
            write-host "{$a}" # output comment line
            $nocl = $nocl - 1

        }

        # file parameters
        $a = ($r_file.ReadLine()).trim() -split "\s*[(),\s]" # split on white-space or comma
        $ns = [int]::Parse($a[0]) # number of elevation samples
        $sd = [double]::Parse($a[1]) # sample distance

```

```

write-host "    ns: $ns, sd: $sd"
$sd = $sd * $cf1 # convert to proper units
$cnt = 0 # if this was not 0, then *$cf1
write-host "Converting..."

$hold = New-Object System.Collections.Generic.List[double[]] # track data

# file data
while ( $r_file.Peek() -ge 0 ) {

    $a = -split $r_file.ReadLine()

    if ( $a[0].substring(0,1) -eq "#" ) {

        # do nothing, a comment line

    } else {

        $a | foreach {
            $elev = [double]::Parse( $_ ) * $cf2
            $hold.Add( @( [double]::Parse( $cnt ), $elev ) )
            $cnt = $cnt + $sd # $sd converted above
            $ns = $ns - 1

            if ( $ns -le 0 ) { break } # number of samples reached

        }
    }
}

$r_file.Close()
$r_file.Dispose()

# detrend
if ( $fdls -ne 0.0 ) {

    write-host "Detrending..."
    ($hold, $rms, $fmean, $xmean) = rmsexp $hold $fdls
    write-host "    RMS=$rms"

}

$ie = $hold.count - 1

@( # put in array for piping to Out-File
foreach ( $i in (0..$ie) ) {

    $p = $hold[$i]
    ("{0,-15:F8}" -f $p[0]) + "    " + ("{0,-15:F8}" -f $p[1])

}
) | Out-File -FilePath $ofn1 -encoding ASCII # output converted data to single track file

===read file loop:end

}

break

}

<#
-Type 2 (single track, two columns)
[comment lines]
[displacement samples, elevation samples]
PS> .\cpc <ifn> <ft> <nocl>, <cf1>, <cf2>, <ofn1>, [fdls]
#>
2 {

    if ( $args.length -lt 4 ) {

        write-host "Format Type 2 requires 4 parameters (nocl, cf1, cf2, ofn1, [fdls])"
        exit

    } else {

        # get parameters
        $nocl = [int]::Parse( $args[0] ) # number of comment lines
        $cf1 = [double]::Parse( $args[1] ) # conversion factor 1 (displacement)
        $cf2 = [double]::Parse( $args[2] ) # conversion factor 2 (elevation)
        $ofn1 = $args[3] # output file name

        if ( $args.length -eq 5 ) {

            $fdls = [double]::Parse( $args[4] ) # filter distance one side for detrending

        } else {

            $fdls = 0.0 # no detrending
            write-host "`n***NO DETRENDING FILTER SPECIFIED, Press Ctrl-C to Cancel`n"
            Start-Sleep -s 10

        }

        # output parameters
        ( "nocl", $nocl),
        ( "cf1", $cf1),

```

```

        ("cf2", $cf2),
        ("ofn1", $ofn1),
        ("fdls", $fdls) ) | foreach {

            write-host ("    {0,-4}:" -f ${_${0}}), ${ ${_${1}}.ToString()}

        }

    }

    ===read file loop:begin

    # read comment lines
    while ( $nocl -gt 0 ) {

        $a = $r_file.ReadLine()
        write-host "{$a}" # output comment line
        $nocl = $nocl - 1

    }

    $hold = New-Object System.Collections.Generic.List[double[]] # track data
    write-host "Converting..."

    # file data
    while ( $r_file.Peek() -ge 0 ) {

        $a = ($r_file.ReadLine()).trim() -split "\s*[,](\s)*" # split on white-space/comma
        $disp = [double]::Parse($a[0]) * $cf1 # displacement
        $elev = [double]::Parse($a[1]) * $cf2 # elevation
        $hold.Add( @( $disp, $elev ) )

    }

    $r_file.Close()
    $r_file.Dispose()

    # detrend
    if ( $fdls -ne 0.0 ) {

        write-host "Detrending..."
        ($hold, $rms, $fmean, $xmean) = rmsemp $hold $fdls
        write-host "    RMS=$rms"

    }

    $ie = $hold.Count - 1

    @( # put in array for piping to Out-File
    foreach ( $i in (0..$ie) ) {

        $p = $hold[$i]
        ("{0,-15:F8}" -f $p[0]) + "    " + ("{0,-15:F8}" -f $p[1])

    }
    ) | Out-File -FilePath $ofn1 -encoding ASCII # output converted data to single track file

    ===read file loop:end

}

break

}

<#
-Type 3 (multiple track, multiple columns)
[comment lines]
[displacement samples, elevation 1 samples, elevation 2 samples, ...]
PS> .\cpc <in> <ft> <nocl>, <cf1>, <cf2>, <cf3>, <lc>, <rc>, <ofn1>, <ofn2>, [fdls])
#>
3 {

    if ( $args.Length -lt 8 ) {

        write-host ("Format Type 3 requires 8 parameters " +
            "(nocl, cf1, cf2, cf3, lc, rc, ofn1, ofn2, [fdls])")
        exit

    } else {

        $nocl = [int]::Parse( $args[0] ) # number of comment lines
        $cf1 = [double]::Parse( $args[1] ) # conversion factor 1 (displacement)
        $cf2 = [double]::Parse( $args[2] ) # conversion factor 2 (elevation, left track)
        $cf3 = [double]::Parse( $args[3] ) # conversion factor 3 (elevation, right track)
        $lc = [int]::Parse( $args[4] ) # left column number
        $rc = [int]::Parse( $args[5] ) # right column number
        $ofn1 = $args[6] # output file name, left track
        $ofn2 = $args[7] # output file name, right track

        if ( $args.Length -eq 9 ) {

            $fdls = [double]::Parse( $args[8] ) # filter distance one side for detrending

        } else {

            $fdls = 0.0 # no detrending
            write-host "`n***NO DETRENDING FILTER SPECIFIED, Press Ctrl-C to Cancel`n"
            Start-Sleep -s 10

        }

    }

}

```

```

}

# output parameters
( "nocl", $nocl,
  "cf1", $cf1,
  "cf2", $cf2,
  "cf3", $cf3,
  "lcn", $lcn,
  "rcn", $rcn,
  "ofn1", $ofn1,
  "ofn2", $ofn2,
  "fdls", $fdls ) | foreach {

    write-host ( "    {0,-4}:" -f ${_${0}} ), ${ ${_1}.ToString() }

}

#===read file loop:begin

# read comment lines
while ( $nocl -gt 0 ) {

    $a = $r_file.ReadLine()
    write-host "{$a}" # output comment line
    $nocl = $nocl - 1

}

$holdl = New-Object System.Collections.Generic.List[double[]] # track data
$holdr = New-Object System.Collections.Generic.List[double[]] # track data
write-host "Converting..."

# file data
while ( $r_file.Peek() -ge 0 ) {

    $a = ($r_file.ReadLine()).trim() -split "\s*[()|\\s)]\s*" # split on white-space/comma
    $disp = [double]::Parse($a[0]) * $cf1 # displacement
    $lelev = [double]::Parse($a[$lcn-1]) * $cf2 # elevation, left track
    $relev = [double]::Parse($a[$rcn-1]) * $cf3 # elevation, right track
    $holdl.Add( @( $disp, $lelev ) )
    $holdr.Add( @( $disp, $relev ) )

}

$r_file.Close()
$r_file.Dispose()

# detrend
if ( $fdls -ne 0.0 ) {

    write-host "Detrending Left Track..."
    ($holdl, $rmsl, $fmeanl, $xmeanl) = rmsexp $holdl $fdls
    write-host "    RMS=$rmsl"
    write-host "Detrending Right Track..."
    ($holdr, $rmsr, $fmeanr, $xmeanr) = rmsexp $holdr $fdls
    write-host "    RMS=$rmsr"

}

$ie = $holdl.count - 1

# left track
@( # put in array for piping to Out-File
foreach ( $i in (0..$ie) ) {

    $p = $holdl[$i]
    ("{0,-15:F8}" -f $p[0]) + "    " + ("{0,-15:F8}" -f $p[1]) # left track

}
) | Out-File -FilePath $ofn1 -encoding ASCII # output converted data to single track file

$ie = $holdr.count - 1

# right track
@( # put in array for piping to Out-File
foreach ( $i in (0..$ie) ) {

    $p = $holdr[$i]
    ("{0,-15:F8}" -f $p[0]) + "    " + ("{0,-15:F8}" -f $p[1]) # right track

}
) | Out-File -FilePath $ofn2 -encoding ASCII # output converted data to single track file

#===read file loop:end

}

break

}

}

write-host "`nDone.`n"

===EOF

```


APPENDIX A.4 – DFC SCRIPT

Powershell Environment

```
Name                               Value
----                               -
PSVersion                         5.1.17763.1432
PSEdition                         Desktop
PSCompatibleVersions              {1.0, 2.0, 3.0, 4.0...}
BuildVersion                      10.0.17763.1432
CLRVersion                        4.0.30319.42000
WSManStackVersion                 3.0
PSRemotingProtocolVersion         2.3
SerializationVersion              1.1.0.1

===BOF
<#
.SYNOPSIS
Creates laboratory simulator drive files.

.DESCRIPTION
Drive File Creation (DFC) PowerShell script creates time history channel data at specified speeds
that define simulator actuator positions (elevation vs. time) for laboratory testing. The time
histories are based on detrended left and right course profiles (elevation vs. displacement). A
definition file (.def) is required that defines the necessary parameter values to create the
drive files. The terms speed and velocity are used interchangeably. Parameters are sometimes
referred to as commands or functions if their values represent some action to be taken.

.INPUTS
Definition File (.def) name that defines the following parameters.

# COMMENT
A comment line (# at beginning of line)

#TRACKS <l_trk_name> <r_trk_name>
Separate file names for the left and right tracks to be used. The format required for each
track file is two columns, with column 1 holding the displacement samples (equal spaced) and
column 2 holding the associated elevation samples. ***BOTH must be in the SAME units of
length.
(e.g. "#TRACKS ./ter/3courL_in_d.cpc ./ter/3courR_in_d.cpc")

#OUTPUTCH <ch1_trk_idx> <ch2_trk_idx> [<ch3_trk_idx> <ch4_trk_idx> ...]
Index of track file index value for each output channel (1=left, 2=right). The number of
channels should match the number of vehicle wheels/spindles being actuated (see #TRACKS and
#DELAY). The ordering of channels is left to the user, but the ordering scheme must be
followed consistently for the other parameters (see #DELAY).
(e.g. "#OUTPUTCH 1 2 1 2" for a 2-axle vehicle where the 2nd and 4th channels are for
actuating the front RIGHT and rear RIGHT wheels/spindles and the 1st and 3rd channels are for
actuating the front LEFT and rear LEFT wheels/spindles)

#DELAY <ch1_delay_dis> <ch2_delay_dis> [<ch3_delay_dis> <ch4_delay_dis> ...]
Delay in displacement (units of length) for each channel. The number of delays should match
the number of output channels and order specified in #OUTPUTCH and represents the distance
behind the front (first) axle.
(e.g. "#DELAY 0 0 130.0 130.0" for a 2-axle vehicle, zero distance for the front left/right
channels and 130.0 wheelbase distance for the rear left/right channels in the units of length
in the track data files. Each delay corresponds to the selected track file (by index number)
in #OUTPUTCH)

#SPEED <spd1>, <spd2> [, <spd_range>]
Single or range of speeds with increments, in units of speed, which the time history output is
to represent. Only INTEGER speeds allowed. The units of speed used here when multiplied by
#VEL2LEN and unit time [sec] should match the units of length in the track data files for
displacement and elevation. (e.g. "#VEL2LEN 17.6" would convert velocity in units of [MPH]
times 1 [sec] to length in units of [in] or "#VEL2LEN 0.277" would convert velocity in units
of [kph] times 1 [sec] to length in units of [m]). Single speeds and ranges are separated by
commas ",", ". A speed range is specified with a beginning and ending speed separated by colons
":" with an increment value in between (e.g. 1:10:100 - a range from 1 to 100 in increments of
10).
(e.g. "#SPEED 3,1:10:100,55" to specify two single speeds and a range from 1 to 100 in
increments of 10)

#SAMPLEFREQ <sampling_freq>
Channel output sampling frequency in units of [Hz]. (The value of 204.8 Hz is required for
specific equipment in the GVSC laboratory.)
(e.g. "#SAMPLEFREQ 204.8")

#RAMPTIME <ramp_time>
Time in units of [sec] to ramp "up" the time history signal from zero to its beginning
elevation value and ramp "down" its ending elevation value to zero.
(e.g. "#RAMPTIME 5" for a ramp time of 5 seconds)

#WAITTIME <wait_time>
Time in units of [sec] before the channel data should start after ramp time.
(e.g. "#WAITTIME 5" for a wait time of 5 seconds)

#LIMITTRAV <travel_limit>
Absolute value of the travel limit from zero, which should match the units of length in the
track data files for displacement and elevation. This value is used only to indicate if the
```

track file data requires the actuator to exceed this amount and in #SCALEPK to resolve this issue (if peak scaling is requested).
(e.g. "#LIMITTRAV 10" for an absolute travel limit of 10 units of length)

#LIMITVEL <velocity_limit>
Absolute value of the velocity limit, which should match the units of length in the track data files for displacement and elevation divided by time (1 [sec]). (**This is an exception to the velocity units defined elsewhere because of the way it is calculated---a difference in the elevation divided by sample time (reciprocal of the sampling frequency) and is generally expressed in smaller units than vehicle speed**) This is used only to indicate if the track file data requires the actuator to exceed this amount.
(e.g. "#LIMITVEL 75" for an absolute velocity limit of 75 [UNITLEN/sec])

#PATCH <patch_filter_len>
Tire patch filter length in units of length (0=no filtering). The length units should match those in the track data files for displacement and elevation. This filter uses a Hann Function or Raised Cosine Window to perform low-pass filtering to simulate the effects tires would have on softening the impact of the terrain into the vehicle. The filter length is usually set to 1.5 times (1.5x) the tire radius (there is no theoretical basis for using this value, only observation from limited test data for a specific vehicle).
(e.g. "#PATCH 25.5" for a filter length with a tire with radius of 17 units of length (17 x 1.5 = 25.5))

#OUTPUT <ch_output_name>
Output file name for the channel time history data based on the parameters specified. Because the time histories are speed dependent, the following composite formatting syntax
"[SPEED={index[,alignment][:formatString]]]" can be included as part of the output name to have the speed value substituted into it.
(e.g. with "#SPEED 3,1:10:100,55" then "#OUTPUT veh1_crsl1_[SPEED={0,-3:D3}].dfc" would output channel time history data for each speed specified in #SPEED as a zero padded 3 digit integer, "veh1_crsl1_003.dfc", "veh1_crsl1_001.dfc", "veh1_crsl1_011.dfc", ..., "veh1_crsl1_100.dfc", "veh1_crsl1_055.dfc")

This is a special parameter. As soon as it is encountered the drive file creation process begins after checking to ensure all required parameter values have been defined. Only parameter values that change need to be updated before using this parameter again. With this feature, multiple course profile, vehicle, and speed dependent drive files can be created from one ".def" file

#UNITLEN <len_symbol>
Symbol that represents the units of length used. This is used only to document the required length units that are to be used. All units of length are assumed to be the same and of this type. The length units should match those in the track data files for displacement and elevation.
(e.g. "#UNITLEN in" for units of length in inches, "mm" for units of length in millimeters, etc.)

#UNITVEL <vel_symbol>
Symbol that represents the units of velocity used. This is used only to document the required velocity units that are to be used. All units of velocity are assumed to be the same and of this type ***with the exception of the units in #LIMITVEL***. The velocity units should match those in #SPEED.
(e.g. "#UNITVEL MPH" for units of velocity in miles per hour, "kph" for units of velocity in kilometers per hour, etc.)

#VEL2LEN <vel_to_len>
Conversion factor between velocity and length. This value should correspond to converting the units in #UNITVEL times unit time (1 [sec]) to the units in #UNITLEN ***with the exception of the units #LIMITVEL***. The basis for this is the relation $D=VT$ where D [length] = $(V[\text{length}/\text{time}] \times \text{vel_to_len}) \times T[\text{sec}]$.
(e.g. "#VEL2LEN 17.6" would convert velocity in units of [MPH] times unit time to length in units of [in], "#VEL2LEN 0.277" would convert velocity in units of [kph] times unit time to length in units of [m], "#VEL2LEN 277.78" would convert velocity in units of [kph] times unit time to length in units of [mm])

#SCALEPK <percent_pk_scale>
Percentage of #LIMITTRAV to scale peak values greater than #LIMITTRAV to (0=no scaling).
(e.g. "#SCALEPK 0.95" would scale peaks greater than #LIMITTRAV to 95% of #LIMITTRAV)

#INSERT <ncol> <ncol> [<ncol> <ncol> ...]
Number of zero value columns to insert before, between, and after defined channels (0=no insertion, max=10). The number of values should be one more than the length of #OUTPUTCH.
(e.g. #OUTPUTCH 1 2 1 2
#INSERT 3 1 1 0 2
insert 3 before ch1, 1 between ch1/ch2, 1 between ch2/ch3, none between ch3/ch4, and 2 after ch4)

.OUTPUTS
A multiple column actuator position time history data file for each set of parameters specified before each #OUTPUTCH command (e.g. vehicle parameters, speeds, course profile, processing options) whose columns represent channels actuator position time history data. The output is usually designated as a ".dfc" file (with the name including the units of length). See the #OUTPUT parameter.

.EXAMPLE
=====Drive File Creation

PS> .\dfc hmwv.def

.EXAMPLE
=====".def" Example

PS> cat hmwv.def

HMMWV, 2 axles, wheel base 130 inches

```

#OUTPUTCH 1 2 1 2
#DELAY 0 0 130.0 130.0
#SPEED 45
#SAMPLEFREQ 204.8
#RAMPTIME 5
#WAITTIME 5
#LIMITTRAV 10
# Units of UNITLEN/sec
#LIMITVEL 75
# tire radius = 17, 1.5x = 25.5
#PATCH 25.5
#UNITLEN in
#UNITVEL MPH
#VELZLEN 17.6
#INSERT 0

#-----COURSES (SET 1)
#SPEED 10:5:25
# scale just for P3
#SCALEPK 0.95
#TRACKS ./ter/3courL_in_d.cpc ./ter/3courR_in_d.cpc
#OUTPUT ./veh/hmmwv/hmmwv_p3_[SPEED={0,-3:D3}].dfc
#SCALEPK 0

#TRACKS ./ter/BBL_in_d.cpc ./ter/BBR_in_d.cpc
#OUTPUT ./veh/hmmwv/hmmwv_bb_[SPEED={0,-3:D3}].dfc

#TRACKS ./ter/CHBSML_in_d.cpc ./ter/CHBSMR_in_d.cpc
#OUTPUT ./veh/hmmwv/hmmwv_chBsm_[SPEED={0,-3:D3}].dfc

#-----COURSES (SET 2)
#SPEED 20:5:40
#TRACKS ./ter/MUNGL_in_d.cpc ./ter/MUNGR_in_d.cpc
#OUTPUT ./veh/hmmwv/hmmwv_mg_[SPEED={0,-3:D3}].dfc

#TRACKS ./ter/PRMAL_in_d.cpc ./ter/PRMAR_in_d.cpc
#OUTPUT ./veh/hmmwv/hmmwv_pa_[SPEED={0,-3:D3}].dfc
#-----END

.NOTES

===UNITS===
Units of time are always in seconds.

===COURSE DATA===
It is assumed that the left and right tracks are available and detrended.

===HISTORY===
2020-09-15 complete insert column, redo lindx after insert
2020-09-14 v5, update handling units and scalepk, insert column.
2020-09-02 Update from Hashtable (@{})/ArrayList to System.Collections.Generic.Dictionary/List
2020-09-01 Updated Shift mean to show after removing it from track data, loop counter in
AveragePatch, speed values only integer
2020-08-31 Corrected Average Patch Filter
2020-08-20 Convert to Powershell. (in .py range(0..end-1) does not include end index. In
Powershell use List.Count)
2011-11-02 Convert to Python.
2007-09-05 Filter spacial data with tire patch (average over) to account for tire filtering.
2007-07-06 Original Perl script created.

===REFERENCES===
[1] Vehicle N-Post Testing Capabilities (slide 38). GVSC Ground Vehicle Simulation Laboratory.
(http://www.usarmygvsc.com/wp-content/uploads/2019/10/GVSC-LAB-TESTING-CAPABILITIES-BRIEFING-23-JULY19.pdf) [2020-10-14]. Distribution Statement A. Approved for public
release; distribution is unlimited.
[2] Federal Labs (CCDC GVSC). (https://federallabs.org/labs/ccdc-ground-vehicle-systems-center)
[2020-10-14]
[3] US. Army Ground Vehicle Systems Center (http://www.usarmygvsc.com/) [2020-10-14]
[4] Federal Labs (ATC) (https://federallabs.org/labs/aberddeen-test-center-atc) [2020-10-14]
[5] Aberdeen Test Center (https://www.atc.army.mil/atc/) [2020-10-14]
[6] Test Operations Procedure (TOP) 01-1-011B Vehicle Test Facilities at Aberdeen Test Center
and Yuma Test Center (12-12-2017). Defense Technical Information Center (DTIC), AD No.
AD1043599. (https://vdl.s.atc.army.mil) [2020-10-14]. Distribution Statement A. Approved
for public release; distribution is unlimited.
[7] Test Operations Procedure (TOP) 01-1-010A Vehicle Test Course Severity (Surface Roughness)
(12-12-2017). Defense Technical Information Center (DTIC), AD No.: AD1043586.
(https://vdl.s.atc.army.mil) [2020-10-14]. Distribution Statement A. Approved for public
release; distribution is unlimited.
[8] RPC Software (https://test.mts.com/products/software-monitoring/rpc-software) [2020-10-14]
[9] Perl (https://www.perl.org/) [2020-10-14]
[10] Python (https://www.python.org/) [2020-10-14]
[11] ActiveState (https://www.activestate.com/) [2020-10-14]
[12] Powershell (https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell
-7) [2020-10-14]
[13] Bylsma. Spacial and Frequency Domain Calculation of Terrain Roughness Metric Root-Mean-
Square (RMS). (TR 26042, November 2014). Defense Technical Information Center (DTIC), AD
No. A617210. Distribution Statement A. Approved for public release; distribution is
unlimited.
[14] MATLAB (https://www.mathworks.com/) [2020-10-14]
[15] Hann Function/Raised Cosine Window (https://en.wikipedia.org/wiki/Hann\_function)
[2020-10-20]
[16] Composite Formatting (https://docs.microsoft.com/en-us/dotnet/standard/base-types/composite
-formatting) [2020-10-20]
[17] Numeric Format (https://docs.microsoft.com/en-us/dotnet/standard/base-types/standard-
numeric-format-strings#DFormatString) [2020-10-20]
[18] Nyquist Frequency (https://en.wikipedia.org/wiki/Nyquist\_frequency) [2020-10-22]
[19] .NET Objects List, Dictionary, StreamReader, StreamWriter (https://docs.microsoft.com/en-us

```

```

    /dotnet/api/system.collections.generic.list-1?view=netcore-3.1, https://docs.microsoft.com/
    en-us/dotnet/api/system.collections.generic.dictionary-2?view=netcore-3.1, https://docs.
    microsoft.com/en-us/dotnet/api/system.io.streamreader?view=netcore-3.1, https://docs.
    microsoft.com/en-us/dotnet/api/system.io.streamwriter?view=netcore-3.1

===PowerShell/.NET Notes===
[1] The type [array] in Powershell is equivalent to:
    # [array]
    $a = [System.Array]::CreateInstance([Object],0)
    # [double[]]
    [double[]] $b = @(1.0, 2.0)
    $b = [System.Array]::CreateInstance([double],0)
    $b = @(0) * 10
[2] For error exception handling:
    try {
        ...
    }
    catch [ArgumentOutOfRangeException] {
        write-host "ArgumentOutOfRangeException"
        $_.Exception.Message
        $_.InvocationInfo.Line
    }
    catch [System.SystemException] { write-host "Base Exception" }
    catch { write-host "catch all Exceptions" }
#>

param ( [string]$defnm )

#=====FUNCTIONS=====
function filltbl( $line, $cmd, $tracks ) {
<#
    read parameters and fill $cmd for processing

    INPUTS:
    $line    line from the definition file (.def)
    $cmd     parameters in definition file (.def)
    $tracks  course profile track data

    VARIABLES:
    $a       hold line split
    $alen    number of elements in line
    $alen1   last element index
#>

    # split line to find the parameter values
    $a = -split $line

    $alen = $a.length
    $alen1 = $alen - 1    # last element

    if ( $alen -lt 1 ) { return }

    switch ( $a[0] ) {

        "" { break }
        $cTRACKS { $cmd[$cTRACKS] = $a[1..$alen1]; break }
        $cOUTPUTCH { $cmd[$cOUTPUTCH] = @( $a[1..$alen1] | foreach { [int]::Parse($_) } ); break }
        $cDELAY { $cmd[$cDELAY] = $a[1..$alen1] | foreach { [double]::Parse($_) }; break }
        $cSPEED { $cmd[$cSPEED] = @( $a[1] ); break }
        $cSAMPLEFREQ { $cmd[$cSAMPLEFREQ] = @( [double]::Parse( $a[1] ) ); break }
        $cRAMPTIME { $cmd[$cRAMPTIME] = @( [double]::Parse( $a[1] ) ); break }
        $cWAITTIME { $cmd[$cWAITTIME] = @( [double]::Parse( $a[1] ) ); break }
        $cLIMITTRAV { $cmd[$cLIMITTRAV] = @( [double]::Parse( $a[1] ) ); break }
        $cLIMITVEL { $cmd[$cLIMITVEL] = @( [double]::Parse( $a[1] ) ); break }
        $cPATCH { $cmd[$cPATCH] = @( [double]::Parse( $a[1] ) ); break }
        $cOUTPUT { $cmd[$cOUTPUT] = @( $a[1] ); output $cmd $tracks; break }
        $cUNITLEN { $cmd[$cUNITLEN] = @( $a[1] ); break }
        $cUNITVEL { $cmd[$cUNITVEL] = @( $a[1] ); break }
        $cVEL2LEN { $cmd[$cVEL2LEN] = @( [double]::Parse( $a[1] ) ); break }
        $cSCALEPK { $cmd[$cSCALEPK] = @( [double]::Parse( $a[1] ) ); break }
        $cINSERT { $cmd[$cINSERT] = @( $a[1..$alen1] | foreach { [int]::Parse($_) } ); break }

    }
}

#=====
function output( $cmd, $tracks ) {
<#
    process and output data (see #OUTPUT)

    INPUTS:
    $cmd     parameters in definition file (.def)
    $tracks  course profile track data
#>

    # output parameters
    write-host "`n===PARAMETERS==="

    $cmd.keys | sort | foreach {

        write-host "$_.:", $( ( $cmd[$_] | foreach { $_.ToString() } ) -join ", ")
        # or [string]$_ vs. $_.ToString()

    }

    chxexist $cmd

```

```

write-host "`n"
write-host "****READING TRACKS****"
readtracks $cmd $tracks # read course profile track data

write-host "****SHIFTING****"
# shift mean of all tracks to be about zero and determine vertical travel limits
shiftall $cmd $tracks

write-host "****CREATING CHANNELS****"
# create channel data and determine sampling frequency and vertical velocity
createchannels $cmd $tracks
}

#=====
function readtracks( $cmd, $tracks ) {
<#
    read track data

    INPUTS:
    $cmd      parameters in definition file (.def)
    $tracks   course profile track data

    VARIABLES:
    $ie       last element index
    $hold     hold track data read in
    $pth      full path name to course profile track data file
    $r_file   file read object
    $a        hold line split
    $rds      sampling distance from track data
    $patchnum number of samples to use for filtering ($L ==> filter length 2*$L+1)
#>

    $ie = $cmd[$cTRACKS].length-1

    foreach ( $i in (0..$ie) ) {

        $hold = New-Object System.Collections.Generic.List[double[]] # track data (was [array])

        if ( Test-Path $cmd[$cTRACKS][$i] ) {

            $pth = ( get-childitem $cmd[$cTRACKS][$i] ).FullName
            $r_file = New-Object IO.StreamReader( $pth )

        } else {

            write-host "Cannot find file [$pth]"
            exit

        }

        write-host "    ",$cmd[$cTRACKS][$i]

        while ( $r_file.Peek() -ge 0 ) {

            $a = -split $r_file.ReadLine()
            $hold.Add( @( [double]::Parse( $a[0] ), [double]::Parse( $a[1] ) ) )

        }

        $r_file.Close()
        $r_file.Dispose()

        # average spacial over patch length
        if ( $cmd[$cPATCH][0] -ne 0 ) {

            write-host "    Average Patch"
            $rds = $hold[1][0] - $hold[0][0] # delta spacial
            write-host "    Displacement Delta = $rds"
            [int]$patchnum = $( $cmd[$cPATCH][0] / (2.0*$rds) + 1.0 ) # samples (1.5x tire radius)
            write-host "    Patch Samples (L) = $patchnum"
            averagepatch $hold 0 ($hold.count-1) $patchnum

        }

        $tracks[$i] = $hold # set track to patch averaged data
        write-host

    } # $i
}

#=====
function averagepatch( $avg, $beg, $end, $L ) {
<#
    average patch filter

    INPUTS:
    $avg      track data to filter
    $beg      start index
    $end      end index
    $L        number of samples (1 side)

    VARIABLES:
    $holdh    filter coefficients (Hann window, see References)
    $aa       initialization array
    $alpha    filter parameter

```

```

$N      size of filter (length = N + 1)
$sum    sum of coefficients/filtered values
$ie     last filter element indexa
$hold   data for filtering
$cnt    offset into track data where filtered value should go
$indx   offset for index into filter
$b      begin index of section to filter
$e      end index of section to filter
#>

$holdh = New-Object System.Collections.Generic.List[double] # filter data
[ double[] ]$aa = @( 0 ) * ($L*2+1)
$holdh.AddRange( $aa ) # initialize

$alpha = 0.5 # Hann window/filter parameter
[double]$N = $L * 2 # length = N + 1 therefore N = length-1 = 2*L+1 -1 = 2*L
$sum = 0.0

# define filter coefficients
$ie = $N

foreach ( $i in (0..$ie) ) {

    # Hann Function/Raised Cosine Window (https://en.wikipedia.org/wiki/Hann_function)
    $holdh[$i] = $alpha - $alpha * [Math]::cos( 2.0 * 3.14159 * $i/$N)
    $sum = $sum + $holdh[$i]

}

# normalize coefficients
foreach ( $i in (0..$ie) ) {

    $holdh[$i] = $holdh[$i]/$sum

}

$sum = 0.0

foreach ( $i in (0..$ie) ) {

    $sum = $sum + $holdh[$i]
    write-host "    L = ",$i," : ", $holdh[$i]

}

write-host "    Sum = $sum"

# define data for filtering (add L to either end)
$hold = New-Object System.Collections.Generic.List[double]

foreach ( $i in (1..$L) ) {

    $hold.Add( $avg[$beg][1] )

}

foreach ( $i in ($beg..$end) ) {

    $hold.Add( $avg[$i][1] )

}

foreach ( $i in (1..$L) ) {

    $hold.Add( $avg[$end][1] )

}

# find average over filter
$cnt = 0

$ie = $hold.count - $L - 1

foreach ( $i in ($L..$ie) ) {

    $sum = 0.0
    $indx = $i - $L
    $b = $i - $L
    $e = $i + $L

    foreach ( $j in ($b..$e) ) {

        # holdh index from 0 to 2*L, L=0,2*L are zero, could modify to not include
        $sum = $sum + $hold[$j] * $holdh[ $j - $indx ]

    }

    # $sum = $sum / (($L*2)+1); # not need when normalize coefficients to one already
    # see $holdh above
    $avg[ $beg + $cnt ][1] = $sum
    $cnt = $cnt + 1

}

}

#=====
function shiftall( $cmd, $tracks ) {

```

```

<#
center channel data

INPUTS:
$cmd      parameters in definition file (.def)
$tracks   course profile track data

VARIABLES:
$tlim     abs travel limit
$stlim    string output for travel limit
$min      min elevation
$max      max elevation/abs(min/max)
$mean     mean of min/max
$ie       last track file element index
$je       last track data element index
$tmmin    min elevation after mean subtracted
$tmmax    max elevation after mean subtracted
$smmax    string of $max
#>

$tlim = $cmd[$cLIMITTRAV][0]
$stlim = "{0,5:F2}" -f $tlim

$min = 1.0e15
$max = -1.0e15

$mean = 0.0

# find min/max
$ie = $cmd[$cTRACKS].length - 1

foreach ( $i in (0..$ie) ) {

    $je = $tracks[$i].count - 1

    foreach ( $j in (0..$je) ) {

        if ( $tracks[$i][$j][1] -lt $min ) { $min = $tracks[$i][$j][1] }
        if ( $tracks[$i][$j][1] -gt $max ) { $max = $tracks[$i][$j][1] }

    }

}

$mean = ($max + $min) / 2.0
write-host "    Mean = $mean"

# subtract mean and find abs max travel from zero
$ie = $cmd[$cTRACKS].length-1

foreach ( $i in (0..$ie) ) {

    write-host "`n ",$cmd[$cTRACKS][$i]

    $tmmin = 1.0e15
    $tmmax = -1.0e15

    $je = $tracks[$i].count-1

    foreach ( $j in (0..$je) ) {

        $tracks[$i][$j][1] = $tracks[$i][$j][1] - $mean
        if ( $tracks[$i][$j][1] -lt $tmmin ) { $tmmin = $tracks[$i][$j][1] }
        if ( $tracks[$i][$j][1] -gt $tmmax ) { $tmmax = $tracks[$i][$j][1] }

    }

    # get abs of min/max after mean was subtracted
    if ( [Math]::abs( $tmmin ) -gt [Math]::abs( $tmmax ) ) {

        $max = [Math]::abs( $tmmin )

    } else {

        $max = [Math]::abs( $tmmax )

    }

    # output travel limit
    $smmax = "{0,5:F2}" -f $max

    if ( $max -gt $tlim ) {

        write-host "    *** $smmax > $stlim TRAVEL LIMIT"

    } else {

        write-host "    $smmax > $stlim TRAVEL LIMIT"

    }

    write-host

}

}

#=====

```

```

function speeds( $cmd ) {
<#
determine speed values (integer values only)

INPUTS:
$cmd      parameters in definition file (.def)

VARIABLES:
$speedvals  speed values (MUST BE INTEGER)
$a          hold line split
$je        last element index
$b          hold line split
$k          speed value in a range
#>

$speedvals = New-Object System.Collections.Generic.List[int]
$a = $cmd[$cSPEED][0] -split "," # split at "," for individual values

$je = $a.length-1

foreach ( $j in (0..$je) ) {

    $b = $a[$j] -split ":" # split at ":" for value range

    if ( $b.length -lt 2 ) {

        # just one value
        $speedvals.Add( [int]::Parse( $b[0] ) )

    } else {

        # value range
        $k = [int]::Parse( $b[0] ) # start value

        while ( $k -le [int]::Parse( $b[2] ) ) { # less than/equal end value

            $speedvals.Add( $k ) # add value in range
            $k = $k + [int]::Parse( $b[1] ) # add increment

        }

    }

}

$speedvals.Sort()
write-host "    Speeds = ", ($speedvals -join ", ")
return $speedvals
}

#=====
function createchannels( $cmd, $tracks ) {
<#
create time based channels

INPUTS:
$cmd      parameters in definition file (.def)
$tracks   course profile track data

VARIABLES:
$speedvals  speed values
$ie        last speed index
$je        last channel index/index counter
$ch        channel data (time based)
$del_t     sampling frequency period (time)
$ramp_t    ramp time (zero to beginning elevation)
$wait_t    wait time
$vlm       velocity limit
$svlim     string of $vlm
$ds        track data spacial sampling distance (spacial frequency = 1/ds)
$dsf       speed based sampling distance (spacial frequency = 1/dsf)
$sc1       formatted string for 2*$dsf
$sc2       formatted string for $ds
$yb        beginning track data value (elevation)
$mtime     channel time (considers speed)
$k         event counter (distance or time)
$val2      temporary value for array creation
$run       wait time distance (considers SPEED speed, WAITTIME time, and DELAY distance)
$mark      end channel index before track data
$ch_len    end track data index
$ke        end track data displacement
$h1        track data sample index < $k (integer)
$h2        fraction beyond $h1
$h3        next sample after $k
$hslp      slope of track data (elev/dist = elev/ds)
$h         linear interpolated elevation between track data sample points

# scalepk
$pmindex   min elevation index
$pmindex   max elevation index
$tlm       abs travel limit for peak check
$scalepk   percentage of $tlm to scale peaks
$pb        beginning channel index to check for peaks > $tlm
$pe        ending channel index to check for peaks > $tlm
$start     index of beginning of peak at zero
$end       index of ending of peak at zero
$pidx     index of maximum peak value

```

```

$pmx      maximum peak value
$sf       peak scale factor ($tlim/$pmx * $scalepk)
$pmn      minimum peak value

$minidx   min velocity index
$maxidx   max velocity index
$mvb      beginning index for max velocity
$mve      ending index for max velocity
$diff     difference in channel elevations between samples
$max      maximum channel elevation difference/max velocity
$min      minimum channel elevation difference
$smax     string of $max
$ye       ending track data value (elevation)
$hold     longest channel data index
$c        channel length
$lindx    index of longest channel
#>

[double]$ds = 0.0
[double]$dsf = 0.0
[double]$run = 0.0
[double]$yb = 0.0
[double]$ye = 0.0
[double]$mtime = 0.0
[double]$k = 0.0
[double]$h = 0.0
[double]$del_t = (1 / $cmd[$cSAMPLEFREQ][0])
[double]$ramp_t = $cmd[$cRAMPTIME][0]
[double]$wait_t = $cmd[$cWAITTIME][0]
[double]$vlim = $cmd[$cLIMITVEL][0]
$svlim = "{0,5:F2}" -f $vlim

# get speed values
$speedvals = speeds $cmd

$ie = $speedvals.count - 1
$je = $cmd[$cOUTPUTCH].length - 1

foreach ( $i in (0..$ie) ) { # each speed value

    write-host "`n  Speed", $speedvals[$i]
    # was .List[[array]
    $ch = New-Object "System.Collections.Generic.Dictionary[ [string], `
        [System.Collections.Generic.List[double]] ]"

    foreach ( $j in (0..$je) ) { # each output channel

        # del disp
        $ds = $tracks[ $cmd[$cOUTPUTCH][$j]-1 ][1][0] - $tracks[ $cmd[$cOUTPUTCH][$j]-1 ][0][0]
        $dsf = $speedvals[$i] * $cmd[$cVEL2LEN][0] * $del_t # time sample distance, D=R*T

        # sample check, (1/dsf) > 2*(1/ds) ==> ds > 2*dsf, dsf (speed/sample freq)/ds
        # (course profile sample freq)
        write-host "`n  Channel", ($j+1)
        $sc1 = "{0,5:F2}" -f (2*$dsf)
        $sc2 = "{0,5:F2}" -f $ds
        write-host "    $sc1 > $sc2 Spacial Frequency"

        # ramp beg
        $ch[$j] = New-Object System.Collections.Generic.List[double[]]
        $yb = $tracks[ $cmd[$cOUTPUTCH][$j]-1 ][0][1] # beg elev
        $mtime = 0.0

        $k = 0.0

        while ( $k -lt $ramp_t ) {

            # scale beg elev by ramp time, had [double]::Parse( ... ) for both
            $val2 = ($yb/$ramp_t) * $k
            $ch[$j].Add( @( $mtime, $val2 ) )
            $mtime = $mtime + $del_t
            $k = $k + $del_t # k a time counter here

        }

        # wait beg (speed dependent distance)
        $k = 0.0
        # run distance based on wait time (speed, delay dependent)
        $run = $speedvals[$i] * $cmd[$cVEL2LEN][0] * $wait_t + $cmd[$cDELAY][$j]

        while ( $k -lt $run ) {

            $ch[$j].Add( @( $mtime, $yb ) ) # run distance is beg elev
            $mtime = $mtime + $del_t
            $k = $k + $dsf # k a distance counter here (speed,sample freq dependent)

        }

        $mark = $ch[$j].count - 1 # end of wait value

        # track data
        $ch_len = $tracks[ $cmd[$cOUTPUTCH][$j]-1 ].count-1

        $k = 0.0
        # track end disp
        $ke = $tracks[ $cmd[$cOUTPUTCH][$j]-1 ][$tracks[ $cmd[$cOUTPUTCH][$j]-1 ].count-1][0]

```

```

while ( $k -le $ke ) {

    #==** ASSUMES ds does not change, equal spaced
    [int] $h1 = [Math]::Truncate($k / $ds) # first base track sample point < $k
    $h2 = ($k / $ds) - $h1 # fraction part
    $h3 = $h1 + 1 # next sample after sample < $k

    if ( $h3 -gt $ch_len ) { $h3 = $ch_len }

    $hslp = ( $tracks[ $cmd[$cOUTPUTCH][$j]-1 ][$h3][1] - `
        $tracks[ $cmd[$cOUTPUTCH][$j]-1 ][$h1][1] ) / $ds # slope of track data
    # base + slope * frac * ds (elev at frac of ds)
    $h = $hslp * ($h2 * $ds) + $tracks[ $cmd[$cOUTPUTCH][$j]-1 ][$h1][1]
    $ch[$j].add( @( $mtime, $h) )
    $mtime = $mtime + $del_t
    $k = $k + $dsf # k a distance counter here (speed,sample freq dependent)

}

# channel data wo/ramp, wait
write-host " Course Profile Index ", ($mark + 1), " to ", ($ch[$j].count-1)

# scale peaks (before find max velocity), assumes mean taken out (peaks about zero)
if ( $cmd[$cSCALEPK][0] -ne 0 ) {

    $pminidx = New-Object System.Collections.Generic.List[array]
    $pmaxidx = New-Object System.Collections.Generic.List[array]
    $tlim = $cmd[$cLIMITTRAV][0]
    $scalepk = $cmd[$cSCALEPK][0]

    write-host "`n Scale Peak"

    $pb = $mark + 1 # start after beginning ramp and wait
    $pe = $ch[$j].count-1

    foreach ( $p in ($pb..$pe) ) {
        write-host -nonewline " $p of $pe`r"
        # positive peaks
        if ( $ch[$j][$p][1] -gt $tlim ) { # exceeds travel limit

            $pstart = $p

            while ( $ch[$j][$pstart][1] -ge 0 ) { # find beginning at zero

                $pstart = $pstart - 1

            }

            $pstart = $pstart + 1
            $pend = $p

            while ( $ch[$j][$pend][1] -ge 0 ) { # find ending at zero

                $pend = $pend + 1

            }

            $pend = $pend - 1
            $pidx = 0
            $pmax = -1

            foreach ( $pp in ($pstart..$pend) ) { # find pos peak

                if ( $ch[$j][$pp][1] -gt $pmax ) {

                    $pmax = $ch[$j][$pp][1]
                    $pidx = $pp
                    #write-host (" {0,5:F2} at {1,5:D}" -f $pmax, $pidx)

                }

            }

            $pmaxidx.Add( @( $pidx, $pmax ) )
            $sf = $tlim/$pmax * $scalepk

            foreach ( $pp in ($pstart..$pend) ) { # scale peak

                $ch[$j][$pp][1] = $ch[$j][$pp][1] * $sf;

            }

            $p = $pend

        }

        # negative peaks
        if ( $ch[$j][$p][1] -lt -$tlim ) { # exceeds travel limit

            $pstart = $p

            while ( $ch[$j][$pstart][1] -le 0 ) { # find beginning at zero

                $pstart = $pstart - 1

            }

            $pstart = $pstart + 1

```

```

$pend = $p

while ( $ch[$j][$pend][1] -le 0 ) {

    $pend = $pend + 1

}

$pend = $pend - 1
$pidx = 0
$pmmin = 1

foreach ( $pp in ($pstart..$pend) ) {    # find neg peak

    if ( $ch[$j][$pp][1] -lt $pmmin ) {

        $pmmin = $ch[$j][$pp][1]
        $pidx = $pp
        #write-host ("    {0,5:F2} at {1,5:D}" -f $pmmin, $pidx)

    }

}

$pmminidx.Add( @( $pidx, $pmmin ) )
# abs so -$pmmin * -$signal not make positive
$sf = $tlim/[Math]::abs( $pmmin ) * $scalepk

foreach ( $pp in ($pstart..$pend) ) {    # scale peak

    $ch[$j][$pp][1] = $ch[$j][$pp][1] * $sf;

}

$p = $pend

}

} # $p

write-host "    Peak (Index, Value)"
$pmmaxidx.foreach( {param($p) write-host ("    ({0,-5:D}, {1,-5:F2})" -f $p[0], $p[1])} )
$pmminidx.foreach( {param($p) write-host ("    ({0,-5:D}, {1,-5:F2})" -f $p[0], $p[1])} )
write-host

} # end scale peak

# find max velocity
$minidx = 0
$maxidx = 0
$min = 1e15
$max = -1e15

$mvb = $mark + 2    # start after beginning ramp and wait +1 for difference/slope
$mve = $ch[$j].count-1

foreach ($mv in ($mvb..$mve) ) {

    $diff = $ch[$j][$mv][1] - $ch[$j][$mv-1][1]
    if ( $diff -lt $min ) { $min = $diff; $minidx = $mv }
    if ( $diff -gt $max ) { $max = $diff; $maxidx = $mv }

}

# vel check
if ( [Math]::abs( $min ) -gt [Math]::abs( $max ) ) {

    $max = [Math]::abs( $min )

} else {

    $max = [Math]::abs( $max )

}

$max = $max / $del_t
$smax = "{0,5:F2}" -f $max

if ( $max -gt $vlim ) {

    write-host "    *** $smax > $svlim VELOCITY LIMIT"

} else {

    write-host "    $smax > $svlim VELOCITY LIMIT"

}

# wait end
# end elev
$ye = $tracks[ $cmd[$cOUTPUTCH][$j]-1 ][$tracks[ $cmd[$cOUTPUTCH][$j]-1 ].count-1][1]
$ke = 0.0

while ( $k -lt $run ) {

    $ch[$j].Add( @( $mtime, $ye ) )
    $mtime = $mtime + $del_t
    $k = $k + $dsf    # k a distance counter here (speed,sample freq dependent)

```

```

    }

    # ramp end
    $k = 0.0

    while ( $k -le $ramp_t ) {

        $val2 = (-$ye/$ramp_t) * $k + $ye # scale end elev by ramp time
        $ch[$j].Add( @( $mtime, $val2 ) ) # scale end elev by ramp time
        $mtime = $mtime + $del_t
        $k = $k + $del_t # k a time counter here

    }

} # $j (each output channel)

$hold = New-Object System.Collections.Generic.List[int]

foreach ( $j in (0..$je) ) {

    $hold.Add( $ch[$j].count-1 )

}

# also $hold = $hold | sort -descending
$hold.Sort()
$hold.Reverse()

foreach ( $j in (0..$je) ) {

    $c = $ch[$j].count-1

    if ( $c -eq $hold[0] ) { # find longest index

        $lindx = $j
        break

    }

}

# insert channels (if NOT (len=1 AND val=0))
if ( -not ( ($cmd[$cINSERT].length -eq 1) -and ($cmd[$cINSERT][0] -eq 0) ) ) {

    write-host " Inserting Channels"
    $ch = insertchannels $cmd $ch $lindx # insert channel data

    # update longest channel
    $hold.Clear()

    $je = $ch.count - 1 # use ch now instead of $cmd[$cOUTPUTCH].length - 1

    foreach ( $j in (0..$je) ) {

        $hold.Add( $ch[$j].count-1 )

    }

    $hold.Sort()
    $hold.Reverse()

    foreach ( $j in (0..$je) ) {

        $c = $ch[$j].count-1

        if ( $c -eq $hold[0] ) { # find longest index

            $lindx = $j
            break

        }

    }

}

outputchannels $cmd $ch $speedvals[$i] $lindx

} # $i (each speed value)

}

#####
function chkexist( $cmd ) {
<#
    check for required parameters
#>

($ctracks, $coutputch, $cdelay, $cspeed, $csamplefreq, $cramp_time, $cwaittime, $climittrav, `
$climitvel, $cpatch, $cunitlen, $cunitvel, $cvel2len, $cscalepk, $cinsert).foreach(
{

    if ( -not $cmd.ContainsKey( $_ ) ) {

        write-host "Required Option $_ Not Defined"
        exit

    }

}

```

```

    }

) # end foreach
}

#####
function outputchannels( $cmd, $sch, $spd, $lindx ) {
<#
    output channel data

    INPUTS:
    $cmd      parameters in definition file (.def)
    $sch      channel data (time based)
    $spd      speed values
    $lindx    index of longest channel

    VARIABLES:
    $fout     output file name parsed for substitution
    $fmt      channel output format
    $pth      full path name to channel time history output file
    $w_file   write file object
    $je       longest channel ending index
    $ke       channel index (all channels)
    $ch_len   ending index of current channel
#>

    $fout = subfout $cmd[$cOUTPUT][0] $spd      #+"\n"
    write-host "    Writing", $fout
    $fmt = '{0,-13:F8}'

    if ( Test-Path $fout ) {

        $pth = (get-childitem $fout).FullName
        write-host "    ***File Already Exists, Overwriting...Press CTRL-C to Cancel"
    # pause

    } else {

        new-item -itemtype file -force -path $fout | out-null

        if ( Test-Path $fout ) {

            $pth = (get-childitem $fout).FullName

        } else {

            write-host "Could Not Create File [$fout]"
            exit

        }

    }

    $w_file = New-Object IO.StreamWriter( $pth )

    $je = $sch[$lindx].count-1    # longest channel

    foreach ( $j in (0..$je) ) {

        $w_file.write( ($fmt) -f $sch[$lindx][$j][0] )    # time
        $ke = $sch.count - 1    # use $sch in case of insert, NOT $ke = $cmd[$cOUTPUTCH].length - 1

        foreach ( $k in (0..$ke) ) {

            $ch_len = $sch[$k].count-1

            if ( $j -gt $ch_len ) {

                # if past end, just keep adding end value (padding)
                $w_file.write( ($("    "+$fmt) -f $sch[$k][$ch_len][1]) )

            } else {

                $w_file.write( ($("    "+$fmt) -f $sch[$k][$j][1]) )

            }

        }

        $w_file.writeline()

    }

    $w_file.close()
    $w_file.dispose()

}

#####
function subfout( $fout, $spd ) {
<#
    substate values in output name (e.g. #OUTPUT trk_tf_[SPEED={0,-3:D3}].dfc )

    INPUTS:
    $fout     output file name
    $spd      speed values

```

```

VARIABLES:
$beg      index of "["
$end      index of "]"
$endp1    $end plus 1 (+1)
$endm1    $end minus 1 (-1)
$str      substring of output file name to substitute
$a        hold line split
$sout     formatted speed value to be used as the substitution string
#>

if ( ($fout.IndexOf("[") -ne -1) -and ($fout.IndexOf("]") -ne -1) ) { # use while to repeat

    $beg = $fout.IndexOf("[")
    $begp1 = $beg + 1
    $end = $fout.IndexOf("]")
    $endp1 = $end + 1
    $endm1 = $end - 1
    $str = $fout.Substring( $begp1, $endm1 - $begp1 + 1)
    $a = $str -split "="

    if ( $a.length -lt 2 ) {

        write-host "Error in output substitution string, see #OUTPUT"
        exit

    }

    if ( $a[0] -eq "SPEED" ) {

        try {
            $sout = $a[1] -f $spd
        }
        catch {

            # $error[0].exception.gettype().fullname
            write-host "Invalid Format Specification [$sout]"
            exit

        }

        $fout = $fout.Substring( 0, $beg - 1 + 1) + $sout + $fout.Substring( $endp1, `
            $fout.length - 1 - $endp1 + 1)

    } else { # other substitutions
        # empty
    }

}

return $fout
}

#####
function insertchannels( $cmd, $ch, $lindx ) {
<#
    insert zero channels

INPUTS:
$cmd      parameters in definition file (.def)
$ch        channel data (time based)
$lindx    ending index of longest channel

VARIABLES:
$ie       last insert channels index
$cc       last output channel index
$chtmp    temporary channel data
$z        array to fill for insert
$je       last longest channel index
$oidx     old index from $ch
$nidx     new index in $chtmp
$col      number of zero columns to insert
#>

$ie = $cmd[$cINSERT].length - 1
$cc = $cmd[$cOUTPUTCH].length

if ( $ie -ne $cc ) { # insert cols should be one more than output channels

    write-host "    Invalid Number of Insertion Columns, $($cc+1) Required"

}

$chtmp = New-Object "System.Collections.Generic.Dictionary[ string], `
[System.Collections.Generic.List[double[]]] ]"
[ double[] ] $z = (, @( 0.0, 0.0 ) ) * ($ch[$lindx].count)

$je = $ch[$lindx].count - 1

foreach ( $j in (0..$je) ) {

    $z[$j][0] = $ch[$lindx][$j][0]
    $z[$j][1] = 0.0

}

$oidx = 0

```

```

$nidx = 0

foreach ( $i in (0..$ie) ) {

    $col = [Math]::abs( $cmd[$cINSERT][$i] )

    if ( $col -gt 10 ) {

        write-host "    Max Columns to Insert Exceeded, Limit is 10"

    } else {

        if ($col -ne 0) {

            foreach ( $j in (1..$col) ) {

                write-host "        New $nidx"
                $chtmp[$nidx] = New-Object System.Collections.Generic.List[double[]]
                $chtmp[$nidx].AddRange( $z)
                $nidx = $nidx + 1

            }

        }

        if ($i -eq $ie ) { break }

        write-host "        New $nidx from Old $oidx"
        $chtmp[$nidx] = New-Object System.Collections.Generic.List[double[]]
        $chtmp[$nidx] = $ch[$oidx]
        $nidx = $nidx + 1
        $oidx = $oidx + 1

    }

}

write-host
return $chtmp

}

#=====
#=====MAIN=====
<#
    DFC Entry Point

    INPUTS:
    $defnm      definition file name
#>

if ( $defnm.length -lt 5 ) {

    write-host "Usage: .\dfc [*.def]"

}

write-host "`nDFC v8.0 2020`n"

# define .def parameter names
$cTRACKS = "#TRACKS"
$cOUTPUTCH = "#OUTPUTCH"
$cDELAY = "#DELAY"
$cSPEED = "#SPEED"
$cSAMPLEFREQ = "#SAMPLEFREQ"
$cRAMPTIME = "#RAMPTIME"
$cWAITTIME = "#WAITTIME"
$cOUTPUT = "#OUTPUT"
$cLIMITTRAV = "#LIMITTRAV"
$cLIMITVEL = "#LIMITVEL"
$cPATCH = "#PATCH"
$cSCALEPK = "#SCALEPK"
$cUNITLEN = "#UNITLEN"
$cUNITVEL = "#UNITVEL"
$cVEL2LEN = "#VEL2LEN"
$cINSERT = "#INSERT"

# define data structures
$cmd = New-Object "System.Collections.Generic.Dictionary[ string], [array] )" # parameters
$tracks = New-Object "System.Collections.Generic.Dictionary[ string], `
[System.Collections.Generic.List[array]] )" # course profile data

# read .def file, for each parameter and set its value and/or execute a command (see #OUTPUT)
if ( Test-Path $defnm ) {

    write-host "`nProcessing Definition File [$defnm]`n"

    get-content $defnm | foreach {

        write-host "    [$_]"
        filltbl $_ $cmd $tracks

    }

} else {

    write-host "Could Not Find File [$defnm]"

}

```

```
}  
write-host "`nProcessing [$defnm] Complete.`n"  
===EOF
```

APPENDIX A.5 – POWERSHELL NOTES

===PowerShell/.NET Notes===

[1] The type [array] in Powershell is equivalent to:

```
# [array]
$a = [System.Array]::CreateInstance([Object],0)
# [double[]]
[double[]] $b = @(1.0, 2.0)
$b = [System.Array]::CreateInstance([double],0)
$b = @(0) * 10
```

[2] For error exception handling:

```
try {
    ...
}
catch [ArgumentException] {
    write-host "ArgumentException"
    $_.Exception.Message
    $_.InvocationInfo.Line
}
catch [System.SystemException] { write-host "Base Exception" }
catch { write-host "catch all Exceptions" }
```