CRUNCH USER'S GUIDE

by

Marshall L. Buhl, Jr.
National Wind Technology Center
National Renewable Energy Laboratory
Golden, Colorado

October 2, 2001

INTRODUCTION

Crunch is a software utility program that performs many types of analyses for one or more files. Much of the code for Crunch came from the author's GPP and GenStats programs. Crunch performs many of the same analyses as GPP; however, it is more of a batch program, whereas GPP is an interactive program. If you are performing repetitive tasks, Crunch is the better choice.

Although Crunch was written to process wind-turbine test or simulation data, it may also be useful for most tabular data.

RETRIEVING FILES FROM THE ARCHIVE

You can download the Crunch archive from our web server page http://wind2.nrel.gov/designcodes/. The file should be named something like "Crunch_v236.exe." Create a Crunch folder somewhere on your file system and put this file there. You can double click on it from Windows Explorer or by entering "Crunch_v236" at a command prompt with the Crunch folder as the current folder. This will create some files and folders. Please see *Installing NWTC Design Codes on PCs Running Windows NT®* for details on how to set up your system to run Crunch.

DISTRIBUTED FILES

The files included in the archive of Crunch are as follows:

ArcFiles.txt	The list of files that are written to the archive.
Archive.bat	The batch file that creates the archive.
Change.log	The list of changes to Crunch.
Crunch.doc	This user's guide in Word format.
Crunch.pdf	This user's guide in PDF format.
Source*.f90	Crunch source files.
Test*.tim	Time-series data used for the automated tests.
Test\Bell.txt	A file that includes the bell character to ring the bell.
Test\CertTest.bat	The batch file used to certify the installation.
Test\DateTime.exe	A program used to generate the date and time.
Test\NewLine.txt	A file with only a new-line character in it.
Test\Test??.cru	The Crunch input files for the certification procedure.
Test\Validate.dat	A data file used in the certification.
Test\tstfiles*.*	The results from the certification test.

CERTIFICATION TEST

Before using Crunch, you should run the certification testing program. It is a DOS batch file called "CertTest.bat" and can be found in the "Test" folder. To test the installation, edit "CertTest.bat," and set the environment variables found near the top of the file to settings compatible with your system. You will probably have to change only the "Editor" variable. Then open up a command window, go to the Test folder, and enter "CertTest".

Crunch will run many times. The test procedure will compare the results to those stored in the "Test\TstFiles" folder. The procedure will write the differences between the output files to a file called "CertTest.out." The test procedure will automatically open this file with the editor you specified with the "Editor" variable. Scan through the file; the only differences should be the date and time stamps in the header of the file. If you recompiled Crunch with another compiler, there may be some slight differences in the last digit of many of the numbers.

PROCESSING INPUT

Specifying the input-parameter file

To determine what functions to perform, Crunch reads an input-parameter file. By default, it reads a file called "param.cru." You can override the default name by specifying it on the command line. Enter "Crunch /h" (PC) or "Crunch -h" (UNIX) for command-line help.

Input-parameter file format

Use one of the sample .cru files found in the test folder as a template. Except for calculated-channel, moving-average, crosstalk, load-rose, and extreme-event information and the list of data files, no lines should be added or removed from the sample input file. For calculated-channel, moving-average, crosstalk, and extreme-event information, there should be one line for each channel. For load roses, there should be one line for each rose. For data files, list the file names one per line. Exactly one line separates each section of the parameter-input file from other sections.

Although my sample input files are written using a sentence-like structure, you do not need to do it that way. For example, one of the first lines in the my sample input files says:

```
1 is the row with the channel titles on it.
```

If you prefer, you can rewrite that line as:

```
1     The row with the channel titles on it.
```

As long as the parameter(s) being read is the first "word" on the line and it is separated from the comment portion of the line with any sort of white space, Crunch will not mind. The amount of spacing is not important—use whatever looks good to you.

Specifying layout of the data files

You can let Crunch figure out how many columns and/or rows your data files have, or you can specify these things explicitly. If you want Crunch to figure out how many columns are in the data file and to use all of them, set the number of input channels to zero and don't give it a list of channels.

If you want Crunch to parse titles, and even units, specify the line(s) that contains such information. If you specify a zero for either, Crunch assumes there is no such line. If you tell Crunch which line contains titles, it will use that line to determine the number of channels. It assumes that channel titles are a contiguous group of letters and/or symbols that do not contain a space, tab, comma, apostrophe, or double quote. For instance, if the title line was the following:

```
Chan1  "Chan2""'Chan_3, Chan-4  " Chan 5"
```

Crunch would find six channels whose titles would be "Chan1", "Chan2", "Chan_3", "Chan-4", "Chan", and "5" (without the quotes). If your input file has a title line like this:

```
"Chan 1" "Chan 2" "Chan 3"
```

and it has three columns of data, you cannot use the auto-detection feature for columns because Crunch will think it has six channels (three of them named "Chan").

If you choose to auto-detect channels and give a zero for the line containing the titles, Crunch will use the first data line to figure how many channels there are. It will assign the channels the names "Chan1", "Chan2", etc. if there are fewer than 10 channels. If there are between 10 and 99 channels, the names will be "Chan01", etc. I'll leave it to the reader to figure out the names if there are more than 99 channels.

If you choose to auto-detect channels and give a zero for the line containing the units, Crunch will not use any unit string for the run. If that is the case, you must not specify the units for calculated channels.

If you want to tell Crunch the channel layout, do so by setting the number of input channels to a number greater than zero. Crunch can reorder and rename channels from the original data file with this feature. It will process only those channels you choose, and apply scales and offsets as it reads the data. If you let Crunch auto-detect channels, it cannot apply scales and offsets.

Crunch can also create new channels using typical Fortran expressions with references to other channels. See the **Calculated Channels** section below.

If you want Crunch to auto-detect the number of rows, set the number of data records to zero. You can specify start and end times if time is available in the data file. If you do, Crunch will use only those records that fall within those limits. If you set the start and end times both to zero, then Crunch will read and use all data records in the file.

You can skip as much of the beginning and/or end of the file as you wish by specifying the line containing the first data record and the total number of records to store in the data array. Crunch assumes that all files have the same number of rows and columns. If Crunch has a read error or runs out of data on files other than the first, it will skip those files. Crunch will allocate the storage array at run time, so you are limited only by the available virtual memory in your computer.

FILE TYPES

Crunch can generate many different types of analyses and can generate almost any combination of them during a single run. The analyses can be performed for individual files or an aggregate of all the files. The aggregate is essentially a concatenation, that is, a series or chain, of all the files into a single data set. For most of the analyses, it is perfectly fine to do so. For rainflow cycle counting, a usually negligible error occurs because of the discontinuities.

Output files generated for individual files use the original data file's root name and append an appropriate extension. Output files generated for aggregate analyses use a user-specified root name instead. The extensions are:

azi	Azimuth averages
eev	Extreme events
ext	Extrapolated extreme values
mod	Modified data

pmf	Probability mass functions (histograms)
rcc	Rainflow cycle counts
sts	Statistics
sum	Summary statistics

Crunch can output data in fixed columns or delimited by tabs. The former is best for viewing with an editor or for printing. The latter is best for importing into spreadsheets.

You can also write out the modified data. The output will contain only the data used for the analyses, and include the effects of scales and offsets, crosstalk corrections, and peak finding.

FILE HEADERS

All output files have similar headers. The headers for files generated for individual analyses contain the following information:

- Program name, version, and compile date
- Original data file name
- Number of records used in the analysis
- Indication of whether the peak-finding algorithm was used
- Mean wind speed and turbulence intensity (if wind-speed channel is specified)

The headers for files generated for aggregate analyses contain the following information:

- Program name, version, and compile date
- Number of records and files used in the analysis
- Indication of whether the peak-finding algorithm was used
- Mean wind speed and turbulence intensity of the aggregate (if wind-speed channel is specified)

CRUNCH FEATURES

General Parameters

The second block of input in the parameter file sets up some general parameters. The first line in this block tells Crunch whether or not to generate one or more statistics files.

Use the second line to tell Crunch if it should output the data in its working array. This data includes the channels used from the input file. If scales and offsets were applied to the input data, if the peak finder was used, or if Crunch filtered any of these channels, these modifications will be reflected in the mod file. The mod file will also include calculated channels, moving averages, roses, and channels with the azimuth averages removed.

Next in the parameter input file is a flag that tells Crunch if it should delimit the columns in the various output files with tabs instead of using fixed column spacing. If you want to view the output with an editor or print it, set it to “False”. If you want to import it into a spreadsheet, set it to “True”.

The fourth parameter in this block is the Fortran format specifier for output of floating-point numbers. If you are not generating tab-delimited output, the full width of the field must be at least 11 characters. This is to accommodate the ten-character channel names, plus a space for separating the columns. If you want more space between columns in your fixed-format files, make the field width wider than 11. Here are some example specifiers and what they will make numbers look like in the output files:

For values:	**65.4321**	**-987654.321**	**0.012345678**
F11.4	65.4321	**********	0.0123
E10.3	0.654E+02	-0.988E+06	0.123E-01
1PE10.3	6.543E+01	-9.877E+05	1.235E-02
ES10.3	6.543E+01	-9.877E+05	1.235E-02
ES10.3E1	6.543E+1	-9.877E+5	1.235E-2
EN10.3E1	65.432E+0	**********	12.346E-3
EN11.3E1	65.432E+0	-987.654E+3	12.346E-3
G11.4	65.43	-0.9877E+06	0.1235E-01

Please note that Crunch will not add space between columns of fixed-width output files. Instead, you must ensure that at least one space will precede the numbers with your format specifier. The only specifier listed above that meets this requirement is “ES10.3E1”, but “ES11.3” and “E11.3” will work.

The next parameter in this block tells Crunch whether or not to treat multiple input files as one aggregate file. If this flag is set ‘True”, the output file will no longer use the root names of the input files in the names of the output files. Instead, it will use the quoted string from the next line of the parameter file as the root name of the output files for the aggregate analyses.

Filtering

Crunch can filter your data before it processes or uses it in any other way. The formulation follows the bilinear infinite-impulse recursive filter described in Numerical Recipes, 2nd edition, on pages 553–555.

You can filter your data with a low-pass, high-pass, or band-pass filter. You will need to specify a high cut-off frequency for the low-pass filter. You will need to specify a low cutoff frequency for the high-pass filter. You will need to specify both low and high cutoff frequencies for the band-pass filter.

Here is an example of the section of the parameter-input file for low-pass filtering:

```
6 of the output columns are to be …
4 5 6 7 8 9
1 is the type of filter (1-LowPass,…
0.0 is the low cutoff frequency (ignored…
10.0 is the high cutoff frequency …
```

Here is an example for band-pass filtering using a different input style:

```
 3    Number of filtered channels
 4, 8    9
 3    Type of filter (1-LP, 2-HP, 3-BP)
 5.0  Low cutoff frequency (Hz)
10.0  High cutoff frequency (Hz)
```

This recursive filter is stable with a somewhat rounded and asymmetric filter response below and above the user-selected cutoff frequencies. The data are filtered twice (forward and backward) to remove phase shift. The filter formulation assumes the data are evenly spaced.

It appears that the ends of the times series may have divergent values due to the filtering process. If you have highly variable data that either begins or ends with data points that are away from the trend line, the filter will force itself to follow these points. It may be appropriate to eliminate both ends of the filtered data from your analyses.

Calculated Channels

You can create new channels of data through the calculated-channels feature, which allows you to specify a single algebraic expression for each new channel. Expressions typically follow the format of standard Fortran expressions. Parentheses allow nesting of expressions. Many intrinsic Fortran functions are available and some others have been added. You can refer to any channel you read in or even previously created calculated channels, thus allowing you to construct more complex expressions by creating intermediate channels. Please refer to Appendix A for more details on the allowed expressions.

Calculated channels are numbered in the order created with the first number being:

<CC_1> = <total # of input channels> + 1

The first line in the calculated-channels section of the parameter input file specifies the number of new channels being created. The second line specifies a seed for the random-number generator. Even if you don't use the random-number generator, you must include this seed in the input file. A comment describing the format for the lines describing the calculated channels is next. After that, enter one line for each calculated channel. These lines contain two or three fields; each is enclosed in quotes (either single or double) and separated by some sort of white space. The first field is for the channel name. The second is for the channel units. Skip this field if units are not being used. The last field is the Fortran-style equation.

Here is an example of the calculated-channels section of the parameter-input file when units are included:

```
3 new calculated channels will be generated.
1234567890 is the integer seed for the random
Format for column info is: Col_Title(10 char
"B1_Mmag"    "(kN m)"      "SQRT(C4^2+C5^2)"
"B1_Mphase"  "(deg)"       "ATAN2D(C5,C4)"
"Random"     "(-)"         "RAND+1"
```

Moving Averages

You can create new channels of data that are moving averages of other channels (those that are input and calculated channels). In the input file, you tell Crunch the name of the new channel(s), the number of the channel being averaged, and an averaging period. Crunch assumes that the time step is constant for all files and records. It will compute the time step from the difference between the first two data records of the first data file. Crunch uses the time step to compute the number of records in the averaging period. For instance, if your data file's time step is 0.05 seconds and you specify an averaging period of 3 seconds (Crunch assumes the units of the period and the time column are the same), Crunch will use 60 (3.0/0.5) records for the running average. You cannot use this feature without a time channel.

<MA_1> = <total # of input channels>
+ <total # of calculated channels> + 1

Here is an example of the moving-average section of the parameter-input file:

```
1 channels will have moving averages…
Format for moving-average info is: "Title"…
"WS_ma", 2, 3
```

I have found two uses for this feature. The first was to compute the three-second average wind speed of turbulence files to see if 50 m/s turbulent winds produced a gust that averages over 70 m/s for three seconds. The other use is to filter data.

Time and Wind-Speed Columns

You can tell Crunch which of your input columns contain the time and wind speed. The time is needed for crossing frequency, extreme-value extrapolation, binned rainflow cycle counting, and filtering. The wind speed, if the column is non-zero, will be put in the headers of files along with the turbulence intensity. Either of these channels may be calculated channels.

Here is an example of this section of the input file:

```
1     Time column.
2     Primary wind-speed column.
```

Load Roses

Crunch can generate load roses. You can have multiple roses in a single Crunch job. The user enters one line of input for each rose. The lines contain a quoted root name, a 0º load column, a 90º load column, and the number of sectors. One new column will be added to the data arrays for each sector. The names of the new columns will be the root name with a 2-digit sequential sector number appended. Because of the two-digit field, you cannot specify more than 99 sectors. The units for the two load columns must be the same. The two load columns can be any of the input channels or calculated channels.

The calculation of the new channels uses the following equations:

$$Angle = (Sector - 0.5)*180/NumSectors$$

$$Load(Sector) = Load0*COS(Angle) + Load90*SIN(Angle)$$

If the number of sectors is three, the three new columns will be for angles 30, 90, and 150º. You will never get 0 or 180º points. There is no need to go beyond 180º, because those loads will be the negatives of the load on the opposite side (the sector that is 180º less). Crunch generates loads only for angles between 0 and 180º.

It may be advisable to use an even number of sectors so that you do not waste space by reproducing the 90º-load channel. I believe the IEC standard calls for 15º resolution for load roses. That requires 12 sectors at the following angles: 7.5, 22.5, 37.5, …, 172.5º.

Here is an example of the load-rose section of the parameter file:

```
2 pair(s) of channels will have load roses…
Format for column info is: "Rose_Title"…
"Blade", 4, 5, 12
"Tower", 6, 7, 12
```

Azimuth Averages

Crunch can generate azimuth averages (AA) of selected channels. It creates two new columns for later analysis for each azimuth-averaged channel. One is a pseudo time series in which the value is the average signal value for the azimuth at each time step. Crunch generates the second detrended channel by subtracting the pseudo AA channel from the original signal.

The AA channels have the same channel name as their original channels, but have an "aa" added to the AA column in the analyses tables. The detrended channels will have "-aa" in the AA column of that table. For the summary tables, the file names will be "<ChannelName>_aa.sum" and "<ChannelName>-aa.sum."

To use azimuth averaging, specify:

- Which channels to average
- How many azimuth bins to use
- Which column contains the azimuth signal
- Whether or not to output the azimuth averages.

Here is sample input for azimuth averages:

```
0     Number of AA columns.
0
0     Number of azimuth bins.
0     The azimuth column.
True  Output azimuth averages to a file?
```

Azimuth-averaged channels are numbered such that the "aa" channels are listed immediately following the last non-AA (original input, calculated, or load-rose) channel. The detrended channels ("-aa") then follow in the same order. Thus, for N non-AA channels and M AA channels, the total list of channels would be:

Column #	Channel
1	<non-AA_1>
.	
.	
N	<non-AA_N>
N+1	<AA_1>"_aa"
.	
.	
N+M	<AA_M>"_aa"
N+M+1	<AA_1>"-aa"
.	
.	
N+M+M	<AA_M>"-aa"

Notice: I spent a lot of time one day trying to figure out why an AA of the azimuth position didn't always produce a perfectly straight line. Because Crunch forces the azimuth value to be a number >= 0 and < 360, you can get strange averages if the original azimuth signal is outside that range. This will most likely manifest itself if the original azimuth signals are > 0 and <= 360. In that case, all the 360 values will be accumulated into the first bin instead of the last bin, thus distorting the results.

This is not an issue unless you choose to azimuth average the azimuth column, which is not a very meaningful thing to do. Do not worry about what the azimuth range is for your data files—Crunch will map them to the 0-360 range.

Crosstalk Removal

Crunch can remove the crosstalk from a pair of signals, such as the strain gages on the blades, shaft, and tower. If the gages are not perfectly aligned, they will not produce pure signals. By applying a weight to the signals, you can remove this crosstalk. Once you've determined the proper weights, Crunch can apply them to the data as they are read in. It replaces the original signals stored in Crunch's memory with ones that have had the crosstalk removed. The matrix operation Crunch uses is:

$$Chan_{1,new} = XT_{11}*Chan_{1,orig} + XT_{12}*Chan_{2,orig}$$

$$Chan_{2,new} = XT_{21}*Chan_{1,orig} + XT_{11}*Chan_{2,orig}$$

The first line of the crosstalk section specifies the number of pairs of channels for removng crosstalk. The second line is a comment describing the format of the following lines. After that, one line for each pair follows. The first two parameters on each line are the channel pairs (they must be orthogonal loads). The third through sixth are

the elements of the crosstalk matrix (XT_{11}, XT_{12}, XT_{21}, XT_{22}). Here is an example:

```
2 pairs of columns will have their crosstalk…
Format for crosstalk info is: Col #1, Col #2, …
4 5   1.0 0.0 0.0 1.0
8 9   0.0 1.0 1.0 0.0
```

Peak Finding

Crunch can find peaks and valleys in the signals and fit them with parabolas. It will then replace the peak and valley values with the maxima and minima of the parabolas. The time values for these peaks are not adjusted. The various analyses will use these new values. The .mod files will include the effects of the peak finder.

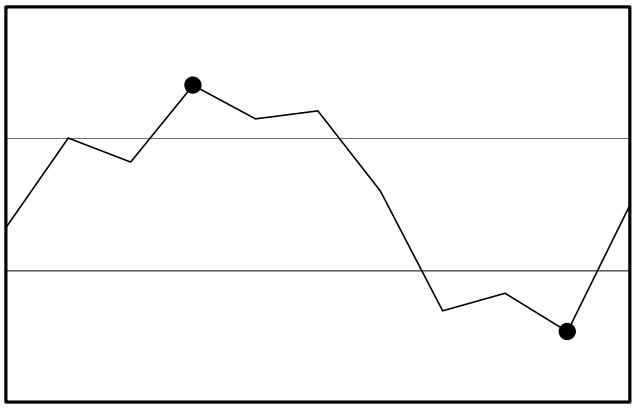
Please note that like all column lists in Crunch, the order of columns do not need to be ascending, as the following example shows:

```
4     Number of columns to use peak finding.
4, 5, 9, 3
```

Peak/Trough Listing

Crunch can generate listings of peaks and/or troughs it finds in the data. It can use either the change-of-slope method or a threshold-crossing method to find them. Crunch creates one file for each channel and input file. If you do an aggregate analysis, Crunch will produce just one aggregate file for each channel. You have the option of including the time the peaks and/or troughs occur in the output files. Please realize that the times will not be monotonically increasing when you do an aggregate analysis. You can also specify whether you want to generate peaks, troughs, or both.

The threshold method divides the data into contiguous blocks that are outside the thresholds. Within each block, it finds the maximum or minimum value and uses it for the peak or trough. The following chart shows an example in which the valid peak and trough are marked with circles:



Please note that the point that just touches the peak threshold is not counted as a peak. In addition, if two data points within a block have the same maximum value, only the first is counted as a peak.

The input section for peak/trough listing consists of a line specifying how many channels to process, one specifying which method to use ("1" for the slope-change method or "2" for the threshold method), and one with a flag that indicates whether or not to include time in the output files. After that is a comment that tells how to enter the channel information. If you set the number of channels to zero, do not include any lines of channel information, but include the comment.

When specifying the channel information, enter one line for each channel that is to be processed. On each line, enter the channel number, a TRUE or FALSE to indicate whether or not to include the troughs in the output, the trough threshold, a TRUE or FALSE to indicate whether or not to include the peaks in the output, and the peak threshold. You cannot specify FALSE for both. If you use the string "MEAN" instead of threshold values, Crunch will use the mean value for the channel and file (or aggregate) as the threshold value for both thresholds. If you are using the slope-change method, the threshold values are ignored, but you must include dummy values. Here is an example:

```
3 channels will have their peaks and/or valleys…
2     Method of identifying peaks (1: slope…
False Include the time in the peak-list…
Format for peak-list info is: Channel,…
3, True, "MEAN", False, MEAN
4, True, -1.0, True, 1.0
5, False, 200.0, True, 300.0
```

In this example, Crunch will use the threshold method. It will not include the time channel in the output. For channel 3, it will find one trough for each block of data that is all below the mean value. For channel 4, it will find both peaks and troughs that are outside the range of –1 to 1. Crunch will output the peaks above 300 for channel 5.

If you analyze files individually, the output files include the root of the input file, the channel name, and the extension "pek." For instance, if you process "input1.dat" and "input2.dat," and list the peaks of "Chan1," Crunch will create two output files called "input1_Chan1.pek" and "input2_Chan1.pek." If you do an aggregate analysis with the aggregate root name of "agg," Crunch will produce only one file and name it "agg_Chan1.pek."

Probability Mass Functions

Crunch can generate probability mass functions (PMFs), which are also sometimes referred to as histograms. A plotted PMF of a signal can give a much better feel for how the data vary than a table of simple statistics.

The input section for PMFs consists of a line specifying how many channels will be processed and one specifying how many bins to use. After that is a comment that tells how to enter the channel information.

When specifying the channel information, enter one line for each channel that is to be processed. On each line, enter the channel number, the minimum, and the maximum. If you set both the minimum and maximum to zero for one or more channels, Crunch will determine them for you. You can mix and match auto-calculation. Here is an example:

```
2 of the output columns will have PMFs…
20 is the number of PMF bins.
Format for column info is: Column #, Minimum…
2, 5.0, 25.0
4, 0.0, 0
```

Rainflow-Cycle Counts

Crunch can generate binned rainflow-cycle counts or raw cycles. You can use these counts or cycles to estimate the fatigue life of a part. Crunch can generate either 1-D (cycle ranges only) or 2-D (cycle ranges and cycle means) tables.

To make it possible to plot the results of one Crunch run with another and have a fair comparison, Crunch can normalize the binned cycle counts by the bin width (1-D) or bin area (2-D). You must multiply the results by the listed normalization constants before using the data to estimate fatigue life. You can also eliminate the need for normalization by specifying the bin sizes explicitly (see below).

To generate rainflow cycles or binned counts, tell Crunch how many channels to process, the rainflow counting period in seconds, whether or not to normalize the results, whether to output spaces or zeros when the bin count is zero and the files are to be tab-delimited, the number of range and mean bins, and the channel information.

If you set the number of range bins to zero, Crunch will output the raw cycles instead of binning them. If you set the number of mean bins to 1, Crunch will output only range data. Otherwise, it outputs both ranges and means. Crunch writes the raw cycles for each channel into separate files with the name "<InFileRootName>_<ChanName>.rcc."

When specifying the channel information, enter one line for each channel that is to be rainflow counted. On each line, enter the channel number, the maximum range, the minimum mean, and the maximum mean. If you set the maximum range to zero, Crunch will auto-calculate it for you. If you set both the minimum mean and maximum mean to zero, Crunch will calculate them, too. You can mix and match auto-calculation. If you are doing only 1-D binning, you do not need to input the minimum mean or the maximum mean. You *can* include them if you like. If you are generating only the raw cycles (unbinned), you do not need to enter the maximum range, minimum mean, or maximum mean. An example of 2D binning follows. Here, channel 4 uses specified bin widths and Crunch automatically calculates bin sizes for channel 5.

```
2     Number of cycle-counted columns.
1     Rainflow counting period (seconds).
True  Normalize rainflow cycle count.
True  For bins with zero counts, output…
20    Number of rainflow range bins
10    Number of rainflow means bins.
      Format for column info is: Column #…
4 1.0e4 –0.5e4 0.5e4
5 0.0 0.0 0.0
```

If you have bazillions of runs and do not want to cycle count them in one huge Crunch job, you can specify the bins widths and then simply add the cycle counts from multiple Crunch jobs. The trick is knowing the maximum range and the limits on the means. You can determine the maximum range you will need by generating aggregate statistics for each batch of files and looking for the largest maximum range from all batches. I would use the most positive maximum and most negative minimum from all batches for the maximum and minimum means.

Extreme-Event Tabulator

You can tabulate extreme events either for individual files or for the aggregate of multiple files. You can create groups of channels to examine additional loads that occur when a member of the group hits an extreme.

Each group has two sets of channels associated with it. The first set is called the extreme channels. These columns of your data will be searched for extremes. The other set is the informational set. The channels in this set will not be searched for extremes, but their values will be added to the tables of extremes.

As an example, you may want to create a blade loads group. In the set of extremes, include the six components of the root forces and moments. Because you may find it useful to know what the time or wind speed are when one of the blade loads hits an extreme, add them to the informational set. This is what the section of the input file might look like:

```
1 groups of parameters will have their extreme…
Format for column info is: Group_Title(100 char…
"Blade Loads", 6, 15,16,17,18,19,20, 2, 1,2
```

You may also be interested in tower loads, but are not necessarily interested in seeing what the tower loads are when the blade loads reach extremes, and vice versa. In this case, create a second group for the tower loads. For the tower loads, you may be interested in knowing the wind direction in addition to the time and wind speed at the times any tower extremes occur. If so, add wind direction to the list of informational channels for the tower group. Each group has its own set of informational channels. If you don't want to have any informational channels in a particular group, set the number of informational channels to zero and

Crunch will ignore any additional numbers on the group line.

For each group, Crunch creates a table of the extreme events. There are twice as many rows in the table as there are extreme channels (minima and maxima). The number of columns in the table is equal to the number of extreme channels plus the number of informational channels. The informational channels are listed last. If you ask for aggregate channels when multiple files are crunched, each row of the table will include the name of the file in which the extreme occurred.

The first line in the extreme-event section of the input file specifies the number of groups. The second is a comment telling you the format of the group lines. After that one line follows for each group. The first field in the group is a quoted string indicating the group name (for example, "Blade Loads"). The string is limited to 100 characters. The second field is the number of channels that will be searched for extreme events.

If you choose aggregate analysis, Crunch will generate one file with the root name specified at the beginning of the input file and with the extension ".eev". If you choose to analyze the files individually, Crunch will generate one extreme-event file for each input file, using the root of the input file name for its root.

Summary Files

Summary files are very useful to summarize the statistics for a number of files. The input for this section is simple. The first line tells Crunch how many parameters to summarize, and the second is a simple list of channels. Here is an example:

```
3      Number of summary files.
4, 3, 5
```

Crunch generates a different output file for each parameter that is summarized. If you specified channel titles or if Crunch found them in the data file(s), the summary files will use the channel titles for the root file name with a .sum extension. If you do not have real titles, the file names will looks like "Chan001.sum."

The first column of the output holds the names of the input data files. If you tell Crunch which channel contains the wind speed, the next two columns of the output will be the mean wind speed and turbulence intensity. Columns for the standard statistics (see the Statistics section below) follow.

Extrapolating Extreme Values

For each channel, Crunch can extrapolate the expected extreme values based on their statistics and the number of hours to extrapolate to. Peter Madsen of Risø (Madsen, et al. 1999 [*editor: I don't know if I'm supposed to use 1998 or 1999*]) developed the method used to calculate these expected values. Output results include expected values for the basis period and for a specified percentile, and expected values for the number of requested hours to extrapolate to based on both "N" and "T" extrapolation. A table of percentiles whose size is determined by the number of input files is also presented along with the data used in the extrapolation process presented in a sorted table.

If azimuth binning is desired, the columns of interest must first be azimuth averaged; then, the "aa" columns must be specified for extrapolation.

This section of the input contains one line telling Crunch how many channels are to be extrapolated. The second is a comment, followed buy one line for each channel (or no lines if none are extrapolated) that contains the channel number, the hours to extrapolate to, and the desired quantile. Here is a sample of the input:

```
1 of the output columns will have their…
Format for statistics info is: Col_#…
4, 1.0, 0.57
```

Statistics

Crunch can generate the following statistics:

- Minimum
- Mean
- Maximum
- Maximum range (Maximum-Minimum)
- Standard Deviation
- Skewness
- Kurtosis
- Mean crossing frequency (how often the signal crosses its mean in a positive-going direction).

Crunch can also create summary files of the statistics for selected channels. For all input data files in a single run, Crunch generates one table of statistics for each requested channel. These tables are written to files called "<ChannelName>.sum." Summary files have two additional columns of statistics—mean wind speed and turbulence intensity.

COMPILING CRUNCH

You should not need to compile Crunch unless you want to make changes to the code. The archive contains code primarily for the Compaq Visual Fortran compiler. All of the compiler-specific code should reside in files called "SysPCD.f90" and "ModPCD.f90." Included in the distribution are untested routines for the Sun f90 compiler. They reside in the files "SysSun.f90" and "SunMod.f90." To compile with either compiler, choose the correct set of compiler-specific files and link them with the other .f90 files. All source code resides in Crunch's "Source" folder.

LIMITATIONS

Crunch has the following limitations:

- All files must have the same number of records.
- All files must have the same channel layout.
- File names must be less than 100 characters long.
- There cannot be more than 998 channels of data.
- Title and units lines must be less than 10,000 characters long if Crunch is automatically determining the file layout (column info).
- Computers must have sufficient virtual memory to contain all the data for a run.
- For aggregate rainflow-cycle counting to work, all files must have the same time step.
- Channel names and units strings are limited to 10 characters each.
- Expressions for calculated channels must be less than 100 characters long.
- For Crunch to work, you must run it on a system that can handle long file names. The distributed Crunch executable file will not work on DOS. It should run fine on 32-bit Windows and UNIX.
- You cannot specify more than 99 sectors for load roses.
- The maximum number of columns in any extreme-event group is 20.

KNOWN BUGS

- None.

POSSIBLE FUTURE ENHANCEMENTS

- Add option to rainflow cycle counting that generates cumulative cycles and outputs the cycle range in the first column.
- For the Extreme-Events feature, add the ability to generate runners up. That is, the second largest loads. Emil Moroz requested this.
- Add binning (e.g., power versus wind speed).
- Add PSDs.
- Calculate equivalent loads and fatigue life.
- Enable Crunch to signify that a value of zero for the time column means that no time column is available. This will mean many features will have to be disabled for that run (crossing frequency, extreme-value extrapolation, binned rainflow cycle counting, and filtering).
- Add Steve Winterstein's extreme-value analysis.
- Consider changing the filtering so that one can specify different filters in a single run. Currently, all filtered channels must use the same filter.
- Modify the calculated-channels tool so that we can specify rows in addition to columns. For example "C1-C1[1]" would subtract the value of column 1 for the first time step from every time step.
- For calculated channels, add integration and differentiation.
- Suggestion from Garrett Bywaters: “For the rainflow counting analysis, it would be really nice if one could specify the # of wind speed bins, and hours in each bin, and run Crunch once, cycle counting and scaling, with say a list of 25 files (5 ten minute files in each of 5 wind speed bins). This would speed up the process, and enable one to readily calculate fatigue under different IEC WTGS wind classes.”
- Add ability to read *BLADED* binary output files.
- Add ability to handle files that are not all the same length. I doubt this will ever happen.

CAVEATS

NREL makes no promises about the usability or accuracy of Crunch, which is essentially a beta code. NREL does not have the resources to provide full support for this program. *You may use Crunch for evaluation purposes only*.

ACKNOWLEDGEMENTS

Marshall Buhl of the National Wind Technology Center (NWTC) wrote most of Crunch. Norm Weaver, of Inter-Weaver Consulting, wrote or designed some algorithms for other programs that Marshall adopted for use in Crunch. Paul Veers of Sandia National Laboratories was the inspiration for the peak-finding algorithm. Larry Schluter of Sandia coded Veers' algorithm in LIFE2 and we started with his code. Crunch uses the rainflow-cycle-counting algorithm taken from “Simple Rainflow Counting Algorithms,” by S.D. Downing and D.F. Socie. Paul Veers originally coded the algorithm and Larry Schluter modified the code to work with the LIFE2 code. The routines used in Crunch are heavily modified versions of those from LIFE2. James Van Buskirk created the parser used for the calculated channels. Dave Laino of Windward Engineering added Peter Hauge Madsen's extreme-value extrapolation algorithms. Kirk Pierce, Garrett Bywaters, Craig Hansen, David Malcolm, and Emil Moroz provided much input, encouragement, and feedback.

Funding for the original Crunch development came from the U.S. Department of Energy (DOE) under contract No. DE-AC36-83CH10093 to the National Renewable Energy Laboratory. The original work was performed under task WE80.4040, which is managed by C.P. “Sandy” Butterfield of the NWTC. Later enhancements were funded by the

DOE under contract No. DE-AC36-98-GO10337 to the NREL. The enhancements were done under many tasks.

FEEDBACK

If you have problems with Crunch, please contact Marshall Buhl. If he has time to respond to your needs, he will do so, but please do not expect an immediate response. Please send your comments or bug reports to:

Marshall L. Buhl, Jr.
NWTC/3811
National Renewable Energy Laboratory
1617 Cole Blvd.
Golden, CO 80401-3393
United States of America

Web: http://wind2.nrel.gov/designcodes/
Email: marshall_buhl@nrel.gov
Voice: (303) 384-6914
Fax: (303) 384-6901

REFERENCES

Madsen, Peter Hauge; Pierce, Kirk; Buhl, Marshall. "Predicting Ultimate Loads for Wind Turbine Design." Presented at the 1999 American Society of Mechanical Engineers Wind Energy Symposium, Reno, Nevada, January 11–14, 1999. NREL/CP-500-25787. Golden, Colorado: National Renewable Energy Laboratory, November 1998.

Appendix A
Crunch Calculated Channels

This is the detailed help for the Calculated-Channels feature of Crunch. Please note that the expressions for calculated channels are not case sensitive.

OPERATORS

"+", "-", "*", and "/" are recognized for algebraic operations. Both "^" and "**" are recognized for exponentiation. Parentheses allow nesting of operations.

The order of operator precedence from highest to lowest is "**" or "^", "*" or "/", and "+" or "-". Expressions within the most deeply nested parenthesis are evaluated first.

All expressions are insensitive to case. Expressions are calculated with 15 digits of precision and can have exponents up to 300.

CONSTANTS

Constants may be entered in the expression as normal decimal numbers or using scientific notation (examples: 0.0123, 1.23E-02 or 1.23D-02). Some common constants are provided for you to make their use easier:

E	2.71828182845904523536
PI	3.14159265358979323846
GAMMA	0.57721566490153286061

CHANNELS

You can refer to existing channels by using a "C" followed by the channel number. For instance, if you want to add channel 3 and channel 6 together to create a new channel, enter the expression "C3+C6".

FUNCTIONS

Many Fortran intrinsic functions are available. We have added some new ones to allow you to specify angles in degrees instead of radians. The following is a list of the functions:

ABS(X)	Returns the absolute value of a real argument.
ACOS(X)	Returns the arc cosine in radians.
ACOSD(X)	Returns the arc cosine in degrees.
ASIN(X)	Returns the arc sine in radians.
ASIND(X)	Returns the arc sine in degrees.
ATAN(X)	Returns the arc tangent in radians.
ATAN2(Y,X)	Returns the arc tangent in radians. Either X or Y must be nonzero.
ATAN2D(Y,X)	Returns the arc tangent in degrees. Either X or Y must be nonzero.
ATAND(X)	Returns the arc tangent in degrees.
CEILING(X)	Returns the smallest integer greater than or equal to the argument. Examples: CEILING(4.8) = 5.0 CEILING(-2.55) = -2.0
COS(X)	Returns the cosine of an angle specified in radians.
COSD(X)	Returns the cosine of an angle specified in degrees.
COSH(X)	Returns the hyperbolic cosine of an angle specified in radians.
DIM(X,Y)	Returns the difference between two numbers if the difference is positive. Otherwise, the result is zero. Examples: DIM(6,2)=4 DIM(-4.0,3.0)=0.0
EXP(X)	Returns the exponential value of a real argument.
FLOOR(X)	Returns the greatest integer less than or equal to the argument. Examples: FLOOR(4.8)=4.0 FLOOR(-5.6)=-6.0
INT(X)	Converts a value to integer type.
LOG(X)	Returns the natural logarithm of a real argument.
LOG10(X)	Returns the common logarithm of a real argument.
MAX(A1,A2,[A3,…])	Returns the maximum value in a list of two or more arguments.
MIN(A1,A2,[A3,…])	Returns the minimum value in a list of two or more arguments.
MOD(A,P)	Returns the remainder when A is divided by P. If P is not zero, Result is A-INT(A/P)*P. If P is zero, the result is undefined. Examples: MOD(7,3)=1 MOD(9,-6)=3 MOD(-9,6)=-3
MODULO(A,P)	Returns the modulo of the arguments. If P is not zero, the result is A-FLOOR(A/P)*P. If P is zero, the result is undefined. Examples: MODULO(7,3)=1 MODULO(9,-6)=-3 MODULO(-9,6)=3
NINT(X)	Returns the nearest integer to a real argument.
RAND	Returns a pseudo-random number with a uniform distribution between 0 and 1. Do not include parentheses or arguments after the function name.
ROOT(X,Y)	Returns X raised to the 1/Y power. If X is negative then Y must be an odd number. Examples: ROOT(8,3)=2 ROOT(-27,3)=-3

SIGN(A,B)	Returns the absolute value of A times the sign of B.
SIN(X)	Returns the sine of an angle in radians.
SIND(X)	Returns the sine of an angle specified in degrees.
SINH(X)	Returns the hyperbolic sine of an angle in radians.
SQRT(X)	Derives the square root of a real argument.
TAN(X)	Returns the tangent of an angle in radians.
TAND(X)	Returns the tangent of an angle specified in degrees.
TANH(X)	Returns the hyperbolic tangent of an angle in radians.