Notes for "tab" suite of utilities - Elihu Ihms

Last revision date: 04.25.2011

These Perl scripts are for the modification of tab or other character delimited data files. In particular, they are very useful for modifying long columns of information that must have a systemic or position-dependent change performed upon them. These types of data are commonly encountered when working with spectral (i.e. absorbance or fluorescence intensity) or the readings of a time-dependent experiment.

The suite consists of several perl scripts, all of which should be set executable via *chmod 700* or the like. It would be most useful to either put them in a directory discoverable via your shell's $path. The scripts were designed to be piped together via the Unix pipe command "|", so that a cohesive workflow can be executed at once. For a brief description of any script's usage and options, you may execute it alone with a sole `-h` option (help).

tread

```
tread -col N [-start N -end N -w|-comma|-delimiter "?"] [file]
```

tread extracts a single column of data from a delimited ASCII file, which it then outputs to STDOUT. If no file is specified, will read from STDIN. Delimiters may be specified explicitly through the *-delimiter* option, or through the *-whitespace* or *-comma* delimiter options. If a delimiter is not specified, tread will default to strict tab-delimited behavior. The first column in a file is column 0, and similarly, the first line in a file is line 0. If the *-start* option is not specified, tread will start reading from the first line. Likewise, if the -end option is not specified, tread will read to the end of the file.

Options:

`-col N`	Read column N of `[file]` (required)
`-w`	Break on whitespace
`-comma`	Break on commas
`-delimiter "x"`	Break on delimiter "x"
`-start N`	Start reading from row N of `[file]`
`-end N`	End reading at row N of `[file]`
`[file]`	File to read from (required)

trex

```
trex [delimiters] [-start N -end N -cols a,b,c...] [-title N] [-ext EXT]
```

trex extracts every row in a passed dataset (following identical delimiter rules as tread) to their own files. Delimiter identity is retained in the extraction process. Columns can be specified by the *-start* or *-end* options, or explicitly specified for extraction (and possible reordering) via the *-cols* command. Issuing trex with the option *-cols 3,0,1* would extract only the specified columns and write them in the resulting file as column3 (delimiter) column0 (delimiter) column1 (newline) . By default, the resulting row files are named after their row number in the parent file. The *-title* option, however, enables the user to change the resulting row name to the value in the row specified. Note that rows with identical values in that column will be overwritten. Finally, the *-ext* option allows for the specification of the resulting row file extension.

Options:

`[delimiters]`	See tread for delimiter specification options
`-start N`	Start extracting columns at column `N`, default is zero.
`-end N`	Stop extracting at column `N`, default is all columns.
`-cols a,b,c...`	Only extracts the columns specified in a comma-delimited list, and will write them to the resulting row file in the specified order.
`-title N`	Instead of writing row files named by their row number, use the value in column `N`.
`-ext "extension"`	Use the specified file extension instead of ".tab"

tmath

math -f [function]

tmath systematically applies a mathematical function to every data element in the passed data column. This can be very useful for removing an offset, or scaling data.

Options:

-f [function] N	Mathematical function to apply, where `function` is "`add`", "`subtract`", "`multiply`", "`divide`", "`round`", "`power`", "`ln`", "`log`", or "`random`". For "`ln`" and "`log`", no second argument is required. For "`round`", `N` is the number of digits past the decimal to round to. For "`rand`", `N` is the maximum random value possible.

tmunge

tmunge -f [function] [operator_file]

tmunge applies an arithmetic function to each element in the passed list based upon a value found within the operator file of the same line number. Allowed functions are *add, subtract, multiply,* and *divide*.

Options:

-f [function]	*Mathematical function to apply. `function` can be "`add`", "`subtract`", "`multiply`", "`divide`", "`chisq`", or "`logchisq`". For the latter two functions, the chi-square or log chi-square difference between the passed value and the corresponding operator file value will be calculated.*
[operator_file]	Operator file to apply to the passed list. (Required)

tsmooth

tsmooth [-mean N|-exp N|-sg 5|7|9|11]

Options:

-mean N	A running mean smoothing over *N* points.
-exp N	Exponential smoothing (*N* can range from 0 to 1, with 1 meaning no smoothing and 0 for maximal smoothing).
-sg 5|7|9|11	Savitsky-Golay smoothing, over 5, 7, 9, or 11 points. Internal quadratic values are taken from *http://www.chem.uoa.gr/applets/appletsmooth/appl_smooth2.html.*

tcalc

tcalc [-stddev|-sum|-avg|-max|-min]

tcalc calculates a single value from a passed column of data.

Options:

-stddev	Calculates the standard deviation of the passed data column
-sum	Calculates the sum
-avg	Calculates the mean value
-max	Returns the maximum value
-min	Returns the minimum value

tsnort

tsnort -asc|-desc|-rev -op [file]

tsnort *numerically* sorts a column of data in the user-specified direction (if a direction is not specified, it is sorted in an ascending manner). Alternatively, an external single-column file *with an identical number of rows of data as the passed column* can be used to determine the sort order of the passed column. tsnort is so named to not conflict with the gcc "tsort" topological sort utility.

Options:

-asc	Sort the data column in ascending order (default)
-desc	Sort in descending order
-rev	Reverses the order of the data column
`-by [sort file]`	Additional file containing a single column of data *with an identical number of elements as the data to be sorted* that will dictate the final order of the passed data.

twrite

tinsert -col N [-start N -end N -w|-comma|-delimiter "?" (-out [output file])] (-orig [parent file] | [parent file])

twrite replaces a column of data in a file with the piped column of data. If the column does not exist, the file will be padded with tabs until the specified column number is met. Note that data cannot extend past the last row of the destination file. If the original file is specified with the -orig flag, the file itself *will not* be modified, but the resulting text will be sent to stdout (unless the output file is specified with the -out option).

Options:

-col N	Write to column N of `[output file]` (required)
-w	Break on whitespace
-comma	Break on commas
-delimiter "x"	Break on delimiter "x"
-start N	Start writing from row N of source
-end N	End writing at row N of source
-orig [parent file]	The original file to insert the data into. If omitted, will simply read and write to the file specified by the last argument.
`-out [output file]`	File to write modified data to. Omitting this option will result in the changes being written to standard output.
`[parent file]`	File to read from and (if no `-out` file is specified) write the modified data back to.

texchange

texchange ([-w|-comma|-delimiter "?"] [file]])

texchange inverts a file containing rows and columns of data by exchanging rows for columns and visa versa. For example, if you had a file with 40 rows and 3 columns, texchange will convert that to a file with 3 rows and 40 columns. Note that although it can understand a wide number of delimiters (just like tread), it will *always* output tab-delimited data

(See tread for delimiter options)

tinterpolate

tinterpolate [x_data] [xy_data]

tinterpolate linearly interpolates y values from a tab (or whitespace) delimited xy dataset to a file of provided x values. This utility is very good for prepping data for submission to tscale, as it will match the number of x values.

tscale

tscale (-upper=N -lower=N -limit=N -grid=N -iter=N) (-log) (-save [file]) (-v) [match_file] [scaled_file]

tscale finds a scaling coefficient that most closely matches one dataset to another. This is useful for overlaying and comparing data with different amplitudes. Note that the number of values in each file must be identical!

Options:

-upper N	The upper bound on the scaling coefficient (default is 100)

-lower N	The lower bound on the scaling coefficient (default is 0)
-limit N	The scaling coefficient resolution to meet (default is 0.0001)
-grid N	The grid size to use when searching, larger numbers are good for highly discontinuous datasets (default is 10)
-iter N	The number of iterations after which to abort (default is 1000)
-log	Use a log SSE to calculate difference, instead of standard SSE, good for datasets with high dynamic range (e.g. SAXS data)
-save [file]	The file to save the scaled values to (optional)
`[match_file]`	The file containing a column of data that the `scale_file` data should be fitted to.
`[scale_file]`	The file containing a column of data that should be scaled to match the data in `match_file`.

tpad

tpad -col N [-w|-comma|-delimiter "?"] (-down) (-fill)

tpad find and potentially fills gaps in columns of sequential data values. See the examples section for usage details.

Options:

(see tread for delimiter options)

-col N	The column of data to watch
-fill	Should missing column values be filled in?

Examples

Assume the file "`file1.dat`", containing the following data:

```
1,1
2,3
3,7
4,11
7,13
```

And the file "`operator.dat`":

```
-4
-3
-2
-1
0
```

```
$>tread -comma -col 0 -start 1 -end 4 file1.dat
2
3
4

$>tread -comma -col 0 -start 1 -end 4 file1.dat | tcalc -avg
3

$>tread -comma -col 0 file1.dat | tmath -add 3
4
5
6
7
10

$>tread -comma -col 0 file1.dat | tmath -f add operator.dat
-3
-1
1
```

```
3
7

$>tread -comma -col 0 file1.dat | tmath -f add operator.dat | \
tmath -f multiply operator.dat
12
3
-2
-3
0

$>tread -comma -col 0 file.dat | tsnort -desc -by operator.dat
7
4
3
2
1

$>tread -comma -col 0 file1.dat | tsmooth -exp 0.5 | \
tinsert -col 2 -start 1 -out smoothed.dat
```

Result: (`smoothed.dat`):

```
1     1
2     3     1.46875
3     7     2.18750
4     11    3.00000
7     13    3.81250

$>texchange -comma file1.dat
1     2     3     4     7
1     3     7     11    13

$>tpad -comma -col 0 -fill file1.dat
1     1
2     3
3     7
4     11
5
6
7     13
```

Note: by using unix redirection to files, individual columns or transformations can be easily saved.

```
$>tread -comma -col 0 file.dat | tmath -f add 5 > added.dat
$>tread -col 0 added.dat | tmath -f multiply 10 > multiplied.dat
```

Result:
(`added.dat`):

```
6
7
8
9
12
```

(`multiplied.dat`):

```
60
70
80
90
120
```

```
$>tscale added.dat multiplied.dat
Converged to within 0.0001, scaling factor for added.dat to match multiplied.dat: 10
```