NAME

xy2xy − transform xy-files.

SYNOPSIS

xy2xy [−ahHdDivVz] [−su] [−qt] [−l*length*] [−f*progfile*] [−p*program*] [−o*outfile*] [−−] [−] [*filename*...]

It is possible to have whitespace between a command line option and its parameter.

DESCRIPTION

xy2xy is a program to apply various data transformations to files with xy-pairs. Each inputfile may have more than one set or chunk of xy-pairs.

Examples of transformations are filtering, normalizing, shifting and scaling. It is also possible to do some calculations on the data and report a resulting value.

The required transformations and calculations are specified through three programs via option **−p** or **−f**: *P1*, *P2* and *P3*. Program *P2* runs for each datapoint (the loop). *P1* runs before the loop over all datapoints, while *P3* runs after this loop.

This section is divided into the following subsections: Initialization, Options, Processing, Programming, Formal specification, xy2xy log-messages, xy2xy error messages and Program exit status.

Initialization

First **xy2xy** collects the options and builds a filelist from the (wildcard) filename specifications on the commandline. When no filenames are specified, **xy2xy** expects a list of filenames from standard input. If no filenames are specified here, **xy2xy** displays the usage screen (xy2xy <nul).

The filelist can be sorted, which is the default, or left as it comes by specifying option **−u** (see note below). **xy2xy** warns about duplicate filenames and normally discards them. By specifying the **−t** option you can accept duplicate filenames.

Note: when MS-DOS is requested to present a filelist from a wildcard file specification, this list is unsorted.

Then **xy2xy** allocates the internal x- and y-buffers. The length of these buffers can be specified through option **−l** on the commandline. If it is not specified there, **xy2xy** tries to determine the buffer length from the first chunk of the first file (that it is able to read succesfully). If the bufferlength cannot be determined from a file, **xy2xy** allocates buffers for 1024 points. It may be necessary to use option **−l** if the chunks do not all have the same number of points and the first chunk is smaller than following chunks.

Options

xy2xy can be executed with the following options:

−a	print author information,
−h	print overview of options,
−H	print program description,
−d	print debug information (−d: stdout, −D: stderr),
−dd	print also filename list built,
−ddd	print also xy-pair each datapoint processed,
−i	select files specified on commandline interactively,
−v	verbose; print summary per file processed (−v: stdout, −V: stderr),
−vv	print also *P1*, *P2* and *P3*
−z	suppress informative messages,
−s	sort list with filenames (default),

−u	leave list with filenames unsorted,
−q	unique: duplicate filenames discarded (default),
−t	duplicate filenames accepted,
−l*length*	buffer length,
−f*progfile*	file with programs *P1*, *P2* and *P3*
−p*program*	program: "*P1* | *P2* | *P3*"; from back to front, programs may be omitted,
−o*outfile*	output filename,
−−	end option section,
−	process xy-pairs from standard input.
filename	e.g. `*.xy` for all .xy datafiles in the current directory.

If you want to use − as the first filename, −− must precede it to end the option section. When a filelist from standard input is used, − cannot be used for *filename.*

Option **−v** increases the verbosity level, while option **−z** lowers it, so **−vz** is effectively a no-operation, and **−Vz** can be used to direct the normal messages to standard error (stderr).

Processing

The following pseudo code describes **xy2xy**'s processing.

```
for each filename in list of filenames do
    for each chunk in filename do
        read this chunk's xy-pairs into internal buffer
        f  = filename [[ cannot yet be used ]]
        c  = chunk-number
        N  = number of points, this chunk
        N1 = N − 1
        N2 = N / 2
        evaluate program P1
        for i in 0 .. N−1 do
            xi = x[i]
            yi = y[i]
            evaluate program P2
            x[i] = xi
            y[i] = yi
        done
        evaluate program P3
        write the (modified) xy-pairs to the output file
    done
done
```

Note 1: **xy2xy** reads lines from the current chunk from the current xy-file and saves the xy-pairs in two internal buffers. Only lines yielding two valid (floating point) numbers are used. Other lines are simply skipped, unless they contain one number or more than two numbers. In this case **xy2xy** concludes that the file is not a valid xy-file.

Note 2: *P1..P3* are the programs specified through the **−f***progfile* or the **−p***program* option.

When all xy-files have been processed and no error occured, the output file is closed and a summary is printed, unless it is suppressed with the **−z** option.

Programming

Transformations of and calculalations on the xy-data are specified through three programs: *P1*, *P2* and *P3*. The programs all run in the same context, e.g.: variables from *P1* can be used in *P2* and *P3*. The following small example illustrates what these programs may look like.

```
#
# asymmtry.prg - calculate a value to represent the asymmetry of a curve.
#

#
# P1 - before loop
#

Smooth(        );             # apply FIR filter
Add   ( -y[N2] );             # shift midpoint to zero
Abs   (        );             # we want to compare areas

%%                            # %% is the section separator

#
# P2 - within loop
#

A1 += y[    i/2 ];            # sum left half's area
A2 += y[ N2+i/2 ];            # sum right half's area

%%                            # %% is the section separator

#
# P3 - after loop
#

print( 'Asymmetry: ', abs(A2 - A1) / (A2 + A1) );    # print relative difference

#
# end of file
#
```

Programming language The programming language resembles the expression statements of the C-language. There are no control-flow statements like if, for etc.. Variables spring into existence when assigned a value. Two datatypes exist: numbers and strings. Further there are predefined values/variables and functions that either operate on the y-buffer or on their argument(s) (see below). Comments are introduced by a '#' and extend to the end of the line.

Numbers Numbers look like 123, 1.23, 1.23e4. A number can have a trailing multiplication specifier like n for nano (10-9) and M for mega (1e+6). So 1m is 0.001, or 1e−3.

Number multipliers		
f	femto	1e−15
p	pico	1e−12
n	nano	1e−9
u	micro	1e−6
m	mili	1e−3
k	kilo	1e3
M	Mega	1e6
G	Giga	1e9
T	Tera	1e12

There are also octal (077) and hexadecimal number formats ($FF, or 0xFF).

Strings Strings look like "A string's ..." and '"Hello", she said.'. Currently, strings can only be used as arguments of the print function.

Variables Variable names look like `a`, `B1`, `thePoint`, `_another`. Only the first 32 characters are significant. When a variable is assigned a value for the first time through one of the assignment operators, it springs into existence with a value of zero. So using a statement like `a += 1;` has the defined behaviour to assign one to `a` the first time. Thus it is not necessary to use the `=` assignment operator to initialize `a` first. When a variable is used before it is defined, an error message is printed like: "*filename*, line *ll*: name '*x*' not found." and a value of zero is substituted for it.

Operators The following table summarizes the available operators with their associativity in decreasing precedence .

Precedence and associativity of operators

Operators						Associativity
`()`	`[]`					l − r
`!`	`+`	`−`	`++`	`--`		r − l
`*`	`/`	`%`				l − r
`+`	`−`					l − r
`<`	`<=`	`>`	`>=`			l − r
`==`	`!=`					l − r
`&&`						l − r
`||`						l − r
`=`	`+=`	`−=`	`*=`	`/=`	`%=`	r − l
`,`						l − r

Unary + and − have higher precedence than the binary form.

Predefined items The following variables or values, arrays and functions are predefined.

Predefined variables/values:

f	current filename [[cannot yet be used]]
c	current chunk number
N	number of xy-pairs read from xy-file into databuffers
N1	N − 1
N2	N / 2
i	current index: 0..N−1 (within *P2*)
xi	current x-value: x[i] (within *P2*)
yi	current y-value: y[i] (within *P2*)

Predefined arrays:

x	buffer with x-coordinates: Array[0..N1] of double;
y	buffer with y-coordinates: Array[0..N1] of double;

Predefined functions:

abs(x)	return absolute value of x
cos(x)	return cosine of x
exp(x)	return e^x
ln(x)	return natural logarithmic of x
log(x)	return 10-log of x

print(x ...)	print value(s) and/or string(s)
sin(x)	return sine of x
sqrt(x)	return square root of x
Abs()	transform y-buffer to its absolute values
Add(x)	add x to all elements of y-buffer
dB()	transform y-buffer values to dB (20 log y)
Derive()	transform y-buffer to its derivative
Integrate()	transform y-buffer to its integral
Invert()	invert all elements of y-buffer
Log10()	transform y-buffer to its 10-log
Max()	return maximum value in y-buffer
Mean()	return mean value of all elements in y-buffer
Min()	return minimum value in y-buffer
Multiply(x)	multiply all elements of y-buffer with x
Normalize()	normalize the y-buffer: y = (y − ymin) / (ymax − ymin)
Reciproce()	transform y-buffer to its reciproce
printStat()	print statistical values of y-buffer: min, max, sum, mean, var, sdv and rms.
Rms()	return root-mean-square (RMS) value of y-buffer (see below)
Sdv()	return standard deviation of y-buffer (see below)
Smooth()	apply an FIR filter to the y-buffer
Sqrt()	transform y-buffer to its square root
Sum()	return sum of all elements in y-buffer
Square()	transform y-buffer to its square
Var()	return variance of y-buffer (see below)

Notes: the names of the functions that operate on the entire y-buffer start with a capital, except for functions dB and printStat.

$$rms = \sqrt{\frac{\sum_{i=0}^{N-1} (y_i)^2}{N}}$$

$$var = \frac{\sum_{i=0}^{N-1} (y_i)^2}{N} - \left(\frac{\sum_{i=0}^{N-1} (y_i)}{N} \right)^2$$

$$sdv = \sqrt{\frac{variance\ N}{N-1}}$$

Formal specification

The following syntax is recognized:

```
input ::=
   stmt_list \n
   stmt_list EOI
   EOI

stmt_list ::=
   expr_list ; stmt_list
   expr_list
   ;

expr_list ::=
   or_expression , expr_list
   or_expression

or_expression ::=
   or_expression || and_expression
   and_expression

and_expression ::=
   and_expression && eq_expression
   eq_expression

eq_expression ::=
   eq_expression == cmp_expression
   eq_expression != cmp_expression
   cmp_expression

cmp_expression ::=
   cmp_expression <  expression
   cmp_expression <= expression
   cmp_expression >  expression
   cmp_expression >= expression
   expression

expression ::=
   expression + term
   expression - term
   term

term ::=
   term * primary
   term / primary
   term % primary
   primary

primary ::=
   NUM
   NAM
   NAM  = expr_list
   NAM += expr_list
   NAM -= expr_list
   NAM *= expr_list
   NAM /= expr_list
   NAM %= expr_list
   NAM ++
   NAM --
   ++ NAM
   -- NAM
   + primary
   - primary
   ! primary
   ( expr_list )
   NAM [ expr_list ]
   NAM ( expr_list )
```

Where the terminal symbols mean:

```
EOI          ::= end-of-input

NAM          ::= letter (letter | digit)*
letter         = [_a-zA-Z]
digit          = [0-9]

NUM          ::= (octal | decimal | hexadecimal | float) suffix?
octal          = 0 [0-7]+
decimal        = [0-9]+
hexadecimal    = hexadecimal1 | hexadecimal2
hexadecimal1   = $ [0-9a-fA-F]+
hexadecimal2   = 0 [xX] [0-9a-fA-F]+
float          = ([0-9]+ | [0-9]+ . [0-9]* | [0-9]* . [0-9]+) ([eE] [-+]? [0-9]+)?
suffix         = [fpnumkMGT]
```

The optional suffix of a NUM specifies a factor of femto to Tera (10−15..10+12).

xy2xy log-messages

Here is an example of **xy2xy** log-messages, obtained with option **−vv**.

```
xy2xy  version 1.2, May 22 2001  Transform xy-files.

P1: '@9;Smooth();@11;Add(-y[N2]);'
P2: '@20;A1+=abs(y[i/2]);@21;A2+=abs(y[N2+i/2]);'
P3: '@30;print('Asymmetry: ',abs(A2-A1)/(A2+A1));'

Asymmetry: 0.0260405
 .\test1xye.xye: chunk 1, 1024 lines, 1024 xy-pairs added to file nul.
Asymmetry: 0.0258662
 .\test3xye.xye: chunk 1, 1024 lines, 1024 xy-pairs added to file nul.

xy2xy: processed 2 files, 2 chunks, 1024 points each column to file nul.
```

xy2xy error messages

Most error messages **xy2xy** can issue, concern file operations that fail. Other error messages are about proceessing data:

filename, line *lll*: cannot process lines of length 200 and longer.
filename, line *lll*: not an xy-file, only one column found.
filename, line *lll*: not an xy-file, more than two columns found.

progname, line *lll*: name '*name*' not found.
progname, line *lll*: divide by 0.
progname, line *lll*: ignoring superfluous comma's.
progname, line *lll*: '*number*' is not a left value.
progname, line *lll*: '*name*' is not a right value.
progname, line *lll*: '*number*' is not a function.
progname, line *lll*: '*number*' is not an array.
progname, line *lll*: '*token*' expected, '*token*' found.
progname, line *lll*: primary expected; token '*token*' found.
progname, line *lll*: unrecognized token '*character*'.
progname, line *lll*: unrecognized operator token '*character*'.

Fbuffer: array index out of range (*index*).
Fbuffer: divide by zero.
Fbuffer: buffer is empty.

Program exit status

When a file cannot be found, or the file cannot be properly processed, the program stops and issues an error message. The failure to process a file is reflected in the programs exit status (see DIAGNOSTICS below).

ENVIRONMENT

No environment variables are used.

FILES

xy2xy uses and creates the following files:

stdin **xy2xy** can read a list of files from standard input (stdin). The filenames must be separated by spaces, tabs or newlines. If the list of filenames does not come from standard input, xy-pairs may be read from standard input by specifying '−' for filename. See also *filename*.

stdout when no output file has been specified with option **−o**, **xy2xy** writes the result to standard output (stdout),

filename input file with one integer or floating point xy-pair per line, separated by spaces or tabs. The file can contain more than one set or chunk of xy-pairs. These chunks must be separated by a line with a single formfeed character (^L^M, or Alt+12,Enter from the keyboard, or \f\n in C-code).

progfile program file specified with option **−f**. This file has three sections, one for each of *P1*, *P2* and *P3*. The sections are separated by a '%%'. See further section Description, subsection Programming.

DIAGNOSTICS

xy2xy can return the following exit values:

0 success: program execution has been successfully completed,

1 commandline error: an invalid option is specified,

2 processing error: a file could not be opened or closed, an error occurred while writing to an output file,

3 interruption: the user interrupted the program,

4 internal error: an unexpected situation in program behaviour occurred.

SEE ALSO

sts-y2xy(1), **xy2xyy**(1), **xyy2xy**(1).

EXAMPLE

1. Apply the program in file progfile.prg to all .xye files, collect the result in file allxye.xyp, log messages to file allxye.log:

```
C:\>xy2xy −v −f progfile.prg −o allxye.xyp ∗xye.xye >allxye.log
```

2. Run to calculate a value: discard new xy-files, collect normal *and* error messages to file allxye.log (via stderr: capital -V and 2>), save result in file allxye.out (Windows NT):

```
C:\>xy2xy −V −f progfile.prg −o nul ∗xye.xye 2>allxye.log >allxye.out
```

3. As above, but specify program on the commandline (Windows NT):

```
C:\>xy2xy −V −p"Smooth()|sum+=yi|print(f,': ',sum/N)" −o nul ∗xye.xye 2>allxye.log >allxye.out
```

4. Collect list of filenames from a file:

```
C:\>type filelist.txt | xy2xy −v −f progfile.prg −o filelist.xyp >filelist.log
```

5. Read file two.xy from standard input (−):

```
C:\>type two.xy | xy2xy −v −f progfile.prg −o one-thr.xyp one.xy − three.xy >one-thr.log
```

6. Read output of program someprog from standard input (−):

```
C:\>someprog | xy2xy −v −f progfile.prg −o some.xyp −− − other.xy >some.log
```

7. Loop through a list of filenames and use Windows NT's filename editing mechanism (%~ni) to determine the names for the new xy-files:

```
C:\>for %i in (*.xye) do xy2xy -vv -p"Smooth()" -o%~ni.xys %i >>allxys.log
```

Note: double % must be used in a batch file.
See also: C:\>help for

LIMITS

xy2xy cannot process files with lines of 200 characters and longer.

BUGS

It is a bit of a problem to use strings in a program on the commandline (**−p** option). The commandline quickly spans more than one line. What happens is not completely clear, but possibly the line brakes within a string, causing mayhem. This is not so much an error of **xy2xy**, but more a problem with the MS-DOS/Windows NT command interpreter command.com or cmd.exe.

(real bugs to be determined.)

AUTHOR

M.J. Moene (moene@biophys.LeidenUniv.nl)

Huygens Laboratorium
Niels Bohrweg 2
2333 CA Leiden
The Netherlands