QSoas reference

December 11, 2015

Contents

Important warning Some commands of `QSoas` may not be documented here. It means they are not supported in any way: their syntax may change without notice, they may disappear, they may turn out to do completely different things in the end. In short, **use them at your own risks** (and it would be a good idea to check out the code, in that case).

The most recent HTML version of this document can always be found there, together with the corresponding PDF version.

Commands, arguments and options (how to read this document)

QSoas works by entering commands inside the command prompt.

Most commands have arguments and options. Arguments and options are separated by spaces:

```
QSoas> command argument1 argument2 "argument 3" /option=option /option2="with spaces"
```

If you need to pass arguments or option values that have spaces, make sure you quote them using `"` or `'`, like in the above example.

Arguments are *italicized* in the documentation below. You need to provide all the arguments for a command to work, and if you don't, QSoas will prompt for them. Some arguments are followed by ..., which means that you can pass several space-separated arguments. This is the case for `load`, for instance:

```
QSoas> load file1 file2 file3
```

Some options are marked as "(default option)", which means that, if all arguments of the command are already specified, you can omit the `/option=` part of the option. For instance, to set the `temperature` to 300 K, you should be doing that:

```
QSoas> temperature /set=300
```

But, as `/set` is the default option, you can omit the `/set=` and write:

```
QSoas> temperature 300
```

In this documentation, all options and arguments have mouseover texts that give a short explanation of what kind of values are expected.

Some commands can be used through a short name (like `q` for `quit`), indicated as such in the present documentation.

Some commands are marked as *(interactive)*. This means that their use requires user input. If they are used in a script, the script pauses for user interaction.

Note about text files

Many commands of QSoas make use of "plain text files", i.e. files that simply contain unformatted text. These are for instance:

- files for defining fits with `load-fits`
- scripts to be run with `run`
- definitions of kinetic systems for `fit-kinetic-system`
- saved fit parameters

On windows, use Notepad to edit them. On Linux, `pico`, `nano`, `vi` or `emacs` are pretty good choices. On MacOS, use TextEdit, but make sure you hit Cmd+Shift+T to switch to "plain text" format; the default is rich text (i.e. text with formatting informations) in the RTF format, and `QSoas` does not understand RTF.

Buffer list (or dataset lists) arguments

Many commands, such as `flag`, `contract` and others take lists of buffers (or datasets) as arguments. This list can take several forms:

- A comma-separated list of buffer numbers (the ones given by `show-stack`), such as: `1,4,7` (0 is the current buffer, 1, the one just before, which you can reach using `undo`, etc.).
- Negative numbers refer to the "redo" stack: `-1` is the buffer you would get by running `redo`
- A number range, such as `1..7`, meaning all buffers from 1 to 7 included.
- A number range with a step, such as `1..7:3`, meaning `1,4,7`.
- `all` for all buffers on the stack.
- `displayed` for the currently displayed buffers.
- `latest` for the datasets produced by the last command (running a script counts as many commands); this can be different from `0` if the last command produced more than one dataset, or none.
- `latest:1` is the same as `latest`, `latest:2` represents the datasets produced by the command before the last one, etc...

It is also possible to make use of buffer flags set by `flag`:

- `flagged` stands for all flagged buffers (regardless of the name of the flag);
- `unflagged` for all buffers that don't have any flag;
- `flagged-` and `unflagged-` do the same, but with the buffers in the reverse order;
- `flagged:`*flagname* for all buffers that have the flag *flagname*;
- `unflagged:`*flagname* for all buffers that don't have the flag *flagname*;
- and the variants `flagged-:`*flagname* and `unflagged-:`*flagname* for the reversed order.

Note in this documentation, the terms "buffer" and "dataset" are synonyms.

Buffer columns

Some commands such as `bin` or `dataset-options` take buffer column names (or numbers) as arguments or options. There are three way to designate those:

- using a number: `1` is the x column, `2` is the y column, and so on
- using a number prefixed by `#`: this is a 0-based index, `#0` is then the x column
- by its name: `x`, `y`, `z`, `y2`, `y3` and so on. `y2` is equivalent to `z`
- `no` or `none` when you don't want to specify a number at all, such as for disabling the display of error bars with `dataset-options`.

Some commands (like `contract`) take column lists, which are comma-separated lists of columns (just like above), with the addition of range: `2..6` are columns 2 to 6 inclusive.

Regular expressions

Some commands, notably `load` and the related commands, make use of "regular expressions". Regular expressions are a way to describe how a text looks like, such as "numbers", "white spaces", "anything that looks like a date", etc. Here is how it works:

- A simple text just matches itself. For instance, using `/separator=,` for `load-as-text` means that the columns are separated by commas.
- `{blank-line}` matches a fully blank line.
- `{blank}` matches a series of blanks. This is the default separator for `load-as-text`.
- `{text-line}` matches a line that does not start by numbers (ignoring spaces).
- `/`*regex*`/`, which is taken as a Qt regular expression. For instance, `/[;,]/` means "either `;` or `,`". Please see the Qt documentation for more information.

General purpose commands

quit - Quit

`quit`
Short name: `q`
Exits QSoas, losing the current session. The full log of the session is always available in the `soas.log` file created in the initial directory. This is indicated at startup in the terminal.
To avoid accumulating very large log files, the log file gets renamed as `soas.log.1` when you start QSoas (and the older one as `soas.log.2`, and so on until `soas.log.5`).
If you want to save the entire state of QSoas before quitting so you can restart exactly from where you left, use `save-stack`.

credits - Credits

`credits /full=`*yes-no*

- `/full=`*yes-no*: Full text of the licenses – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

This command displays credits, copyright and license information of QSoas and all the dependencies linked to or built in your version. You'll get the full license text with `/full=true`.
It also lists publications whose findings/equations/algorithms were directly used in QSoas.

save-history - Save history

`save-history` *file*

- *file*: Output file – values: name of a file

Saves all the commands that were launched since the beginning of the session, to the given (text) file.
This can be used for saving a series of command that should be applied repetitively as a script.

cd - Change directory

`cd` *directory* `/from-home=`*yes-no* `/from-script=`*yes-no*
Short name: `G`

- *directory*: New directory – values: name of a directory
- `/from-home=`*yes-no*: If on, relative from the home directory – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/from-script=`*yes-no*: If on, cd relative from the current script directory – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Changes the current working directory. If `/from-home` is specified, the directory is assumed to be relative to the user's home directory. If `/from-script` is specified, the directory is assumed to be relative to that of the command file currently being executed by a `run` command (or in a startup script).

pwd - Working directory

`pwd`
Prints the full path of the current directory.
It is also indicated in the title of the QSoas window.

temperature - Temperature

`temperature /set=`*number*
Short name: `T`

- `/set=`*number* (default option): Sets the temperature – values: a floating-point number

Shows or sets the current temperature, in Kelvins. The temperature is used in many places, mostly in fits to serve as the initial value for the temperature parameter. To set the temperature, pass its new value using the `/set` option (the `/set=` part is optional):

```
QSoas> temperature 310
```

commands - Commands

`commands`
List all available commands, with a short help text. This also includes used-defined commands, such as custom fits loaded from a fit file and aliases.

help - Help on...

`help` *command* `/online=`*yes-no*
Short name: `?`

- *command*: The command on which to give help – values: the name of one of QSoas's commands
- `/online=`*yes-no*: Show the online documentation in a browser – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Gives all help available on the given command. By default, it spawns a browser to show the online help, unless you use `/online=false`.

save-output - Save output

`save-output` *file*

- *file*: Output file – values: name of a file

Save all text in the terminal to a plain text file. Equivalent to copy-pasting the contents of the terminal to a plain text file using a text editor.

print - Print

`print` `/file=`*file* `/title=`*text*
Short name: `p`

- `/file=`*file*: Save as file – values: name of a file
- `/title=`*text*: Sets the title of the page as printed – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Prints the current view, providing a usual print dialog. If you just want a PDF or PostScript file, just provide the file name as the `/file` option.
An optional title can be added using the `/title` option.
Important note: QSoas is not a data plotting system, it is a data analysis program. Don't expect miraculous plots !

define-alias - Define alias

`define-alias` *alias command* `/*=`*text*

- *alias*: The name to give to the new alias – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- *command*: The command to give an alias for – values: the name of one of QSoas's commands
- `/*=`*text*: All options – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

The `define-alias` commands allows one to defined a shortcut for a command one uses often with the same options. For instance, running:

```
QSoas> define-alias fit-2exp fit-exponential-decay /exponentials=2 /loss=true
```

creates a `fit-2exp` command that is equivalent to starting `fit-exponential-decay` with two exponentials by default and film loss on.
Alias can only be used to provide default values for options. It cannot provide default values for arguments.

display-aliases - Display aliases

`display-aliases`
Shows a list of all the currently defined aliases.

graphics-settings - Graphics settings

`graphics-settings` `/line-width=`*number* `/opengl=`*yes-no* `/antialias=`*yes-no*

- `/line-width=`*number*: Sets the base line width for all lines/curves – values: a floating-point number
- `/opengl=`*yes-no*: Turns on/off the use of OpenGL acceleration – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/antialias=`*yes-no*: Turns on/off the use of antialised graphics – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Gives the possibility to tweak a few settings concerning display. The settings are kept from one QSoas session to the next.
Turning on antialias (with `/antialias=true`) will make QSoas use antialiased drawings, which looks admittedly nicer, but requires much more computation time, to the point that drawing jagged curves may become particularly slow. Printing or exporting to PDF files through `print` always produces antialiased graphics, regardless of this option.
If you experience performance problems for displaying curves, use `/opengl=true`, as this will instruct QSoas to use hardware acceleration to display curves. It is off by default as some setups do not really benefit from that, and the OpenGL support is sometimes buggy and may result in crashes.

ruby-run - Ruby load

`ruby-run` *file*

- *file*: Ruby file to load – values: name of a file

This command loads and executes a Ruby file. For the time being, the main interest of this command is to define complex functions in a separate file.

Imagine you have a file `function.rb` containing the text:

```
def mm(x,vmax,km)
  return vmax/(1 + km/x)
end
```

After running

```
QSoas> ruby-run function.rb
```

You can use `mm` like any normal function for fitting:

```
QSoas> fit-arb mm(x,vmax,km)
```

or use it in `eval`:

```
QSoas> eval mm(1.0,2.0,3.0)
 => 0.5
```

break - Break

`break`

Exits from the current script. Has no effect if not inside a script.

system - System

`system` *command...* `/shell=`*yes-no* `/timeout=`*integer*

- *command...*: Arguments of the command – values: one or more files. Can include wildcards such as *, `[0-4]`, etc...
- `/shell=`*yes-no*: use shell (on by default on Linux/Mac, off in windows) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/timeout=`*integer*: timeout (in milliseconds) – values: an integer

The `system` command can be used to run external commands from QSoas. The output of the commands will be displayed in the terminal.

For the duration of the external command, QSoas will not respond to keyboard and mouse.

If `/use-shell` is on (the default on Linux and Mac, but off in Windows), the command will be processed by the shell before being run.

If a strictly positive `/timeout` is specified, the command will be killed if it takes longer than the timeout to execute.

timer - Timer

`timer`

The first call starts a timer, and the second one stops it, showing the amount of time that has elapsed since the previous call to `timer`. This can be used to benchmark costly computations, for instance.

Output file manipulation

Several commands (e.g. various data analysis commands and the fit commands) write data to the output file.

By default, the first time the output file is used, a `output.dat` file is created in the current directory. Another file can be used by providing its name to the `output` command.

output - Change output file

`output` `/file=`*file* `/overwrite=`*yes-no* `/reopen=`*yes-no* `/meta=`*words*

- `/file=`*file* (default option): name of the new output file – values: name of a file
- `/overwrite=`*yes-no*: if on, overwrites the file instead of appending (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/reopen=`*yes-no*: if on, forces reopening the file (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/meta=`*words*: when writing to output file, also prints the listed meta-data – values: several words, separated by ','

This command has several modes of operations. If *file* is provided (it is the default option, so you can omit `/file=`), then it opens *file* as the new output file. By default, if the file exists, new data are appended, and the old data are left untouched. You can force overwriting by specifiying `/overwrite=true`.

In the other mode, when only the `/meta` option is provided, it sets the list of meta-data that will automatically be added to the output file when outputting any data there. It is a comma-separated list of meta names. See more about meta-data there.

It is a bad idea to modify the output file while QSoas is still using it, as it messes up what QSoas think is in the output file. If you forgot you were using the output file and modified it, you can avoid problems by running:

```
QSoas> output /reopen=true
```

comment - Write line to output

`comment` *comment*

- *comment*: Comment line added to output file – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Writes the given line *comment* to the output file. Don't forget to quote if you need to include spaces:

```
QSoas> comment 'Switching to sample 2'
```

Data loading/saving

The main command for loading data is `load`.

load - Load

`load` *file...* `/style=`*style* `/flags=`*words* `/ignore-cache=`*yes-no* `/yerrors=`*column* `/histogram=`*yes-no* `/separator=`*pattern* `/decimal=`*text* `/skip=`*integer* `/comments=`*pattern* `/columns=`*integers* `/auto-split=`*yes-no*

Short name: `l`

- *file...*: the files to load – values: one or more files. Can include wildcards such as *, `[0-4]`, etc...
- `/style=`*style*: style for curves display – values: one of: `red-blue`
- `/flags=`*words*: flags for the newly created buffers – values: several words, separated by ','
- `/ignore-cache=`*yes-no*: if on, ignores cache (default off) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/yerrors=`*column*: name of the column containing y errors – values: the number/name of a column in a buffer
- `/histogram=`*yes-no*: whether to show as a histogram (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/separator=`*pattern*: separator between columns – values: plain text, or regular expressions enclosed within / / delimiters
- `/decimal=`*text*: decimal separator – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/skip=`*integer*: skip that many lines at beginning – values: an integer
- `/comments=`*pattern*: pattern for comment lines – values: plain text, or regular expressions enclosed within / / delimiters
- `/columns=`*integers*: columns loaded from the file – values: a comma-separated list of integers
- `/auto-split=`*yes-no*: if on, create a new dataset at every fully blank line (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Loads the given files and pushes them onto the data stack. QSoas features several backends for loading files ("backends" are roughly equivalent to "file formats"). In principle, QSoas is smart enough to figure out which one is correct, but you can force the use of a given backend by using the appropriate `load-as-` command. Using a backend directly also provides more control on the way files are loaded (this can also be done via the numerous options to `load`, which are forwarded to the appropriate backend). Currently available backends:

- `text` for plain space-separated text
- `csv` for CSV data
- `chi-txt` for file from CH Instruments potentiostats

Look in their documentation for more information. In particular, the options `/separator=`, `/decimal=`, `/skip=`, `/comments=`, `/columns=` and `/auto-split` are documented in the `load-as-text` command.

QSoas tells you which backend it used for loading a given file:

```
QSoas> load 03.dat
Loading file: './03.dat' using backend text
```

The command `load` caches the loaded file. If for some reason, the cache gets in the way, use the direct `load-as-` commands, or alternatively use `/ignore-cache=true`.

`load`, like all the other commands that take several files as arguments, understand unix-like wildcards:

```
QSoas> load *.dat
```

This command loads all the files ending by `.dat` files from the current directory.

```
QSoas> load [0-4]*.dat
```

This loads only those that start with a digit from 0 to 4, etc.

One can also set various dataset options while loading with `load` (and the `load-as-` commands), using the options `/yerrors=` and `/histogram=`. See the `dataset-options`, command for more information

The `/style=` option sets the color style when loading several curves:

```
QSoas> load *.dat /style=red-blue
```

This loads all the `.dat` files in the current directory, and displays them with a color gradient from red (for the first loaded file) to blue (for the last loaded file).

With the `/flags=` option, on can flag buffers as they get loaded. Using it has the same effect as running `flag` with the same option on loaded datasets.

load-as-text - Load files with backend 'text'

`load-as-text` *file...* `/separator=`*pattern* `/decimal=`*text* `/skip=`*integer* `/comments=`*pattern* `/columns=`*integers* `/auto-split=`*yes-no* `/style=`*style* `/flags=`*words* `/yerrors=`*column* `/histogram=`*yes-no*

- *file...*: the files to load – values: one or more files. Can include wildcards such as *, `[0-4]`, etc...
- `/separator=`*pattern*: separator between columns – values: plain text, or regular expressions enclosed within / / delimiters
- `/decimal=`*text*: decimal separator – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/skip=`*integer*: skip that many lines at beginning – values: an integer
- `/comments=`*pattern*: pattern for comment lines – values: plain text, or regular expressions enclosed within / / delimiters
- `/columns=`*integers*: columns loaded from the file – values: a comma-separated list of integers
- `/auto-split=`*yes-no*: if on, create a new dataset at every fully blank line (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/style=`*style*: style for curves display – values: one of: `red-blue`
- `/flags=`*words*: flags for the newly created buffers – values: several words, separated by ','
- `/yerrors=`*column*: name of the column containing y errors – values: the number/name of a column in a buffer
- `/histogram=`*yes-no*: whether to show as a histogram (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Loads files using the backend `text`, bypassing cache and automatic backend detection. `text` recognizes space-separated data (which includes tab-separated data). Most "plain text" files will be read correctly by this backend. By default, it loads all the columns of the file, but only displays the second as a function of the first. If you want to work on other columns, have a look at `expand`. Alternatively, you can specify the columns to load using the `/columns` option, see below.

Apart from the options of `dataset-options` and the `/style` and `/flags` options documented in the `load` command, the `text` backend accepts several options controlling the way the text files are interpreted:

- `/separator` specifies the text that separates the columns (blank spaces by default). You can use regular expressions.
- `/decimal` specifies the decimal separator for loading (default is the dot). This is for loading only.
- `/comments` specifies a regular expression describing comment lines (ie lines that get ignored). By default, line that don't start by a number are ignored.
- Give to `/skip` a number of text lines that should be ignored at the beginning of the text file.
- If `/auto-split` is `true`, then `QSoas` will create a new dataset everytime it hits a series of blank lines in the file.
- `/columns` is a series of numbers saying in which order the file columns will be used to make a dataset. For instance, `/columns=2,1` will swap X and Y at load time.

load-as-csv - Load files with backend 'csv'

`load-as-csv` *file...* `/separator=`*pattern* `/decimal=`*text* `/skip=`*integer* `/comments=`*pattern* `/columns=`*integers* `/auto-split=`*yes-no* `/style=`*style* `/flags=`*words* `/yerrors=`*column* `/histogram=`*yes-no*

- *file...*: the files to load – values: one or more files. Can include wildcards such as *, `[0-4]`, etc...
- `/separator=`*pattern*: separator between columns – values: plain text, or regular expressions enclosed within / / delimiters
- `/decimal=`*text*: decimal separator – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/skip=`*integer*: skip that many lines at beginning – values: an integer
- `/comments=`*pattern*: pattern for comment lines – values: plain text, or regular expressions enclosed within / / delimiters

- `/columns=`*integers*: columns loaded from the file – values: a comma-separated list of integers
- `/auto-split=`*yes-no*: if on, create a new dataset at every fully blank line (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/style=`*style*: style for curves display – values: one of: `red-blue`
- `/flags=`*words*: flags for the newly created buffers – values: several words, separated by ','
- `/yerrors=`*column*: name of the column containing y errors – values: the number/name of a column in a buffer
- `/histogram=`*yes-no*: whether to show as a histogram (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

The `csv` backend is essentially the same backend as the `text` one, but with the separators set by default to commas and semicolons, to parse CSV files. Hence, the options have the same meaning as for `load-as-text`.

load-as-chi-txt - Load files with backend 'chi-txt'

`load-as-chi-txt` *file...* `/separator=`*pattern* `/decimal=`*text* `/skip=`*integer* `/comments=`*pattern* `/columns=`*integers* `/auto-split=`*yes-no* `/style=`*style* `/flags=`*words* `/yerrors=`*column* `/histogram=`*yes-no*

- *file...*: the files to load – values: one or more files. Can include wildcards such as *, `[0-4]`, etc...
- `/separator=`*pattern*: separator between columns – values: plain text, or regular expressions enclosed within / / delimiters
- `/decimal=`*text*: decimal separator – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/skip=`*integer*: skip that many lines at beginning – values: an integer
- `/comments=`*pattern*: pattern for comment lines – values: plain text, or regular expressions enclosed within / / delimiters
- `/columns=`*integers*: columns loaded from the file – values: a comma-separated list of integers
- `/auto-split=`*yes-no*: if on, create a new dataset at every fully blank line (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/style=`*style*: style for curves display – values: one of: `red-blue`
- `/flags=`*words*: flags for the newly created buffers – values: several words, separated by ','
- `/yerrors=`*column*: name of the column containing y errors – values: the number/name of a column in a buffer
- `/histogram=`*yes-no*: whether to show as a histogram (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

This is a slightly modified version of `load-as-text` that handles better text files from CH Instruments (and is in particular able to detect at least some of their meta-data).

expand - Expand

`expand` `/perp-meta=`*text* `/flags=`*words*

- `/perp-meta=`*text*: defines meta-data from perpendicular coordinate – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/flags=`*words*: flags for the new buffers – values: several words, separated by ','

If a buffer contains several columns, QSoas only displays the second as a function of the first. `expand` splits the current buffer into as many buffers as there are Y columns, ie a X, Y1, Y2, Y3 buffer will be split into three buffers: X, Y1; X, Y2 and X, Y3.

If `/perp-meta` is specified, then the given meta-data will be defined for each buffer, based on the value of the perpendicular coordinates.

rename - Rename

`rename` *new-name*

Short name: `a`

- *new-name*: New name – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Changes the name of the current buffer. To help track the operations applied to a buffer, its name is modified and gets longer after each modification. Use `rename` to give it a more meaningful (and shorter) name.

If you need to rename a large number of buffers, you probably want to try `save-buffers` with `/mode=rename`.

save - Save

`save` *file* `/overwrite=`*yes-no* `/mkpath=`*yes-no*

Short name: `s`

- *file*: File name for saving – values: name of a file
- `/overwrite=`*yes-no*: If true, overwrite without prompting – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/mkpath=`*yes-no*: If true, creates all necessary directories – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Saves the current buffer to a file. This command will ask you before overwriting an existing file, unless `/overwrite=true` was specified.

It will also change the name of the file.

save-buffers - Save

`save-buffers` *buffers...* `/format=`*text* `/expression=`*text* `/mode=`*choice* `/mkpath=`*yes-no*

- *buffers...*: buffers to save – values: comma-separated lists of buffers in the stack, see buffers lists
- `/format=`*text*: overrides buffer names if present – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/expression=`*text*: a Ruby expression to make file names – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/mode=`*choice*: if using `/format` or `/expression`, whether to just `save`, to just `rename` or `both` (defaults to 'both') – values: one of: `both`, `rename`, `save`
- `/mkpath=`*yes-no*: if true, creates all necessary directories (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Saves the designated buffers to files.

Unlike the `save` command, this saves the buffers using their current names, and does not prompt for a file name. It is probably a good idea to use `rename` first, or use the possibilities below.

This command can rename the buffers before saving them, by using a `[printf](http://www.cplusplus.com/reference/cstdio/prin` like format, as in the following case, which renames the first 101 buffers to `Buffer-000.dat`, `Buffer-001.dat`, and so on:

```
QSoas> save-buffers /format=Buffer-%03d.dat 0..100
```

It is also possible to use a full-blown Ruby expression that will be aware of the buffer's meta-data:

```
QSoas> save-buffers '/expression="File-#{$meta["sr"]}.dat"'
```

This requires careful quoting: outer single quotes (') for QSoas and inner double quotes for Ruby. See more information about the informations available from the ruby code there.

If you only need to rename the buffers without saving them, use `/mode=rename`.

browse - Browse files

`browse` `/pattern=`*text* `/for-which=`*text*

Short name: `W`

- `/pattern=`*text* (default option): Files to browse – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/for-which=`*text*: Select on formula – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Browse all datafiles in the current directory (or those matching the wildcard pattern given to `/pattern`, see `load` for more information about wildcards). Very useful to find quickly the file you're looking for.

Using the `/for-which` option, one can display only a certain set of files based on their meta-data and/or statistics. See the dedicated section for more details.

Data display

overlay-buffer - Overlay buffers

`overlay-buffer` *buffers...* `/style=`*style*

Short name: `V`

- *buffers...*: Buffers to overlay – values: comma-separated lists of buffers in the stack, see buffers lists
- `/style=`*style*: Style for curves display – values: one of: `red-blue`

Plots one or several buffers on top of the current buffer.

See `load` for the description of the `/style` option.

hide-buffer - Hide buffers

`hide-buffer` *buffers...*

Short name: `H`

- *buffers...*: buffers to hide – values: comma-separated lists of buffers in the stack, see buffers lists

This does the reverse of the `overlay-buffer` command. Pass it the buffers you want to remove from the current view. Don't be afraid of passing it non-visible datasets, QSoas won't shout at you if you do.

overlay - Overlay

`overlay` *file...* `/style=`*style* `/flags=`*words* `/ignore-cache=`*yes-no* `/yerrors=`*column* `/histogram=`*yes-no* `/separator=`*pattern* `/decimal=`*text* `/skip=`*integer* `/comments=`*pattern* `/columns=`*integers* `/auto-split=`*yes-no*

Short name: `v`

- *file...*: the files to load – values: one or more files. Can include wildcards such as *, `[0-4]`, etc...
- `/style=`*style*: style for curves display – values: one of: `red-blue`
- `/flags=`*words*: flags for the newly created buffers – values: several words, separated by ','
- `/ignore-cache=`*yes-no*: if on, ignores cache (default off) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/yerrors=`*column*: name of the column containing y errors – values: the number/name of a column in a buffer
- `/histogram=`*yes-no*: whether to show as a histogram (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/separator=`*pattern*: separator between columns – values: plain text, or regular expressions enclosed within / / delimiters
- `/decimal=`*text*: decimal separator – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/skip=`*integer*: skip that many lines at beginning – values: an integer
- `/comments=`*pattern*: pattern for comment lines – values: plain text, or regular expressions enclosed within / / delimiters
- `/columns=`*integers*: columns loaded from the file – values: a comma-separated list of integers
- `/auto-split=`*yes-no*: if on, create a new dataset at every fully blank line (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

This command combines `overlay-buffer` and `load` in one go: loads the files given as arguments and adds them to the current plot; it has the same options as those commands.

clear - Clear view

`clear`

Removes all datasets except the current buffer from the display. Use to revert the effect of a previous overlay command, or can be useful if for some reason a command failed while not restoring the display (but that should not happen anyway).

points - Show points

`points`

Short name: `poi`

Shows datapoints (by default, datasets are plotted by connecting datapoints with a line). Beware that it may slow down display if you have a large number of data points.

zoom - Zoom

`zoom` **(interactive)**

Short name: `z`

Zooms on the current curve.

Click to delimit a region. Hit `x` to zoom in on the X axis, `X` to zoom out, `y` and `Y` for the Y axis, and `z`/`Z` for both at the same time. Hit `c` to reset the zoom.

Indepently of this function, you can use the mouse wheel **at any moment** to zoom in and out: * mouse wheel: zoom in and out vertically * Shift+mouse wheel: zoom in and out horizontally * Ctrl (or Cmd) + mouse wheel: zoom in and out (horizontally and vertically) * Shift+Ctrl + mouse wheel: reset zoom.

If you know the coordinates around which you'd like to zoom, you may want to use `limits` instead.

limits - Set limits

`limits` *left right bottom top*

- *left*: Left limit – values: a floating-point number
- *right*: Right limit – values: a floating-point number
- *bottom*: Bottom limit – values: a floating-point number
- *top*: Top limit – values: a floating-point number

This is the non-interactive version of `zoom`. You specify the left, right, bottom and top values of the currently displayed window directly on the command-line. There are two special values:

- `*` means "auto", or in other words the maximum needed to see all the curves for that specific side (left, right, bottom or top)
- = means "don't change"

Data stack manipulation

Data files are loaded and manipulated in a stack. Every time a file is loaded or a buffer modified, the new buffer is pushed onto the top of the stack, and becomes the current buffer (numbered 0). Older buffers have increasing numbers (the previous buffer is 1, the one before 2, and so on). There is also a "redo" stack populated by the `undo` command. Stack can be manipulated in different ways:

- the current buffer can be changed using `undo`/`redo`;
- buffers can be permanently removed from the stack using `drop`;
- the whole stack can be saved for later use with `save-stack` and restored using `load-stack`, or dropped altogether using `clear-stack`;
- contents of the stack can be displayed in the terminal using `show-stack` or in a dialog bog with `browse-stack`.
- an old buffer can be put back on the top of the stack with `fetch`.
- buffers can be flagged (`flag`) or unflagged (`unflag`) to be used later using the `flagged` buffer selector.

browse-stack - Browse stack

`browse-stack` **(interactive)**
Short name: `K`
Displays a dialog box to show the current contents of the stack, like for `browse`.

show-stack - Show stack

`show-stack` `/number=`*integer*
Short name: `k`

- `/number=`*integer* (default option): Display only that many buffers around 0 – values: an integer

Shows a small text summary of what the stack is made of. If your stack is large and you just need to look at a few buffers, use `/number=10` for instance (that will only show buffers `-9` to `9`).

undo - Undo

`undo` `/number=`*integer*
Short name: `u`

- `/number=`*integer* (default option): Number of operations to undo – values: an integer

Returns to the previous buffer, and push the current to the redo stack. If `/number=` is specified, repeat that many times.

redo - Redo

`redo` `/number=`*integer*
Short name: `r`

- `/number=`*integer* (default option): Number of operations to redo – values: an integer

Pops the last buffer from the redo stack and set it as the current buffer. `/number` has the same meaning as for `undo`.

save-stack - Save stack

`save-stack` *file*

- *file*: File name for saving stack – values: name of a file

Saves the contents of the stack for later use, in a private binary format.

load-stack - Load stack

`load-stack` *file*

- *file*: File name for saving stack – values: name of a file

Loads a saved stack, from a file that was created using `save-stack`.

clear-stack - Clear stack

`clear-stack`
Short name: `delstack`
Removes all the buffers from both normal and redo stack

fetch - Fetch an old buffer

`fetch` *buffers...*

- *buffers...*: Buffers to fetch – values: comma-separated lists of buffers in the stack, see buffers lists

Put back a *copy* of the given buffer on the top of the stack. Useful when you want to work again on a old buffer buried in the stack.

drop - Drop dataset

`drop /buffers=`*buffers*

- `/buffers=`*buffers* (default option): Buffers to drop – values: comma-separated lists of buffers in the stack, see buffers lists

Permanently deletes the current dataset (or the ones specified in the `/buffers` options) from the stack.

```
QSoas> drop 3..16
```

drops all the buffers from 3 to 16 included.
Important: it is *not* possible to recover a buffer once it has been dropped from the stack. `undo` won't work. Use it only on buffers you're sure you won't need again (such as computation intermediates).

flag - Flag datasets

`flag /buffers=`*buffers* `/for-which=`*text* `/flags=`*words* `/set=`*yes-no*

- `/buffers=`*buffers* (default option): Buffers to flag/unflag – values: comma-separated lists of buffers in the stack, see buffers lists
- `/for-which=`*text*: Select on formula – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/flags=`*words*: Buffers to flag/unflag – values: several words, separated by ','
- `/set=`*yes-no*: If on, clears all the previous flags – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Flags the given buffer (or the current one if none is supplied) for later use. All currently flagged buffers can be specified using the `flagged` argument to, for instance, `overlay-buffer`.
`QSoas` supports arbitrary text flags, by passing a comma-separated list of flags to the `/flags=` option. In the absence of that, the buffers are flagged with the flag name `default`. Buffers can hold an arbitrary number of flags. For instance:

```
QSoas> flag 0..5 /flags=exp1,fit
```

flags buffers 0 to 5 with the flags `exp1` and `fit`. Buffers are flagged 'in-place': the current buffer is not changed.
If the `/for-which` option is present, the flags are only applied to the datasets that match the specifications given. See more about that there.

unflag - Unflag datasets

`unflag /buffers=`*buffers* `/for-which=`*text* `/flags=`*words*

- `/buffers=`*buffers* (default option): Buffers to flag/unflag – values: comma-separated lists of buffers in the stack, see buffers lists
- `/for-which=`*text*: Select on formula – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/flags=`*words*: Buffers to flag/unflag – values: several words, separated by ','

Does the reverse of `flag`, that is removes all flags on the given datasets, or only those specified by the `/flags` option if the latter is present. The `/for-which` option words exactly in the same way as for `flag`.

auto-flag - Auto flag

`auto-flag /flags=`*words*

- `/flags=`*words* (default option): Flags – values: several words, separated by ','

This command can be used to automatically flag the datasets produced by any command afterwards, until a call to `auto-flag` without options:

```
QSoas> auto-flag /flags=stuff
 [ ... create new datasets. They will all be flagged stuff,
 until the following command ...]
QSoas> auto-flag
```

This can be used to flag automatically all the datasets produced by a script, for instance.

Basic data manipulation at the buffer level

apply-formula - Apply formula

`apply-formula` *formula* `/extra-columns=`*integer* `/use-stats=`*yes-no* `/use-meta=`*yes-no*
Short name: `F`

- *formula*: formula – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/extra-columns=`*integer*: number of extra columns to create – values: an integer
- `/use-stats=`*yes-no*: if on, the formula can use `$stats` to refer to statistics (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/use-meta=`*yes-no*: if on (by default), a `$meta` hash is available that contains the dataset meta-data – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Applies a formula to the current dataset. It should specify how the `x` and/or `y` values of the dataset are modified:

```
QSoas> apply-formula 'x = x**2'
QSoas> apply-formula 'y = sin(x**2)'
QSoas> apply-formula 'x,y = y,x'
```

The last bit swaps the *x* and *y* values of the buffer. The formula must be valid ruby code.
In addition to `x` and `y` (note the lowercase !), the formula can refer to:

- `i`, the index of the data point
- `seg`, the number of the current segment (starting from 0)
- `y2`, `y3`, etc when there are more than 2 columns in the dataset

`i` and `seg` cannot be modified, but `y2` and so on can.
Extra columns initially filled with 0 can be created by using the `/extra-columns` option:

```
QSoas> apply-formula /extra-columns=1 'y2 = y**2'
```

This creates a second column containing the square of the values of the Y column.
If `/use-stats=true` is used, a global variable `$stats` can be used within the Ruby expression that contains all the statistics displayed by `stats`. For instance, to normalize the Y values by dividing by the median, one would use:

```
QSoas> apply-formula /use-stats=true 'y = y/$stats["y_med"]'
```

Statistics by segments (see more about segments there) are available too, which means if you want to normalize by the medians of the first segment, you could do

```
QSoas> apply-formula /use-stats=true 'y = y/$stats[0]["y_med"]'
```

If `/use-meta` is `true` (the default), then a global variable `$meta` is defined that contains the value of the meta-data (what is shown by `show`). What you make of this will greatly depend of the meta-data QSoas has gathered from your file (and the ones you may have set manually using `set-meta`).

dx - DX

`dx`
Replaces the Y values by the values of delta X, i.e, `y[i] = x[i+1] - x[i]`. This is useful to see if the X values are equally spaced.

dy - DY

`dy`
Same as `dx` but for Y values: replaces the Y values by the values of delta Y.

zero - Makes 0

`zero` *value* `/axis=`*axis*

- *value*: – values: a floating-point number
- `/axis=`*axis*: which axis is zero-ed (default y) – values: one of: `x`, `y`

Given an X value, shifts the Y values so that the point the closest to the given X value has 0 as Y value.
If `/axis` is `x`, swap X and Y in the above description.

shiftx - Shift X values

`shiftx`
Shift X values so that the first point has a X value of 0.

norm - Normalize

`norm /positive=`*yes-no* `/map-to=`*numbers*

- `/positive=`*yes-no*: whether to normalize on positive or negative values (default true) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/map-to=`*numbers* (default option): Normalizes by mapping to the given segment – values: several floating-point numbers, separated by :

Normalize the current buffer by its maximum value (or by the absolute value of its most negative value if `/positive` is `false`).

If the `/map-to` option is specified, the original dataset is mapped linearly to the given segment:

```
norm /map-to=2:4
```

shifts and scales the original data so that the Y minimum is 2 and the Y maximum is 4.

deldp - Deldp

`deldp` **(interactive)**

With this command, you can click on given data points to remove them. Useful to remove a few spikes from the data. Middle click or `q` to accept the modifications, hit escape to cancel them.

edit - Edit dataset

`edit`

Pops up a spreadsheet-like window where you can view and edit the individual values of the current dataset. If you want to save your modification, press the "push new" button.

sort - Sort

`sort`

Sorts the buffer in increasing X values.

reverse - Reverse

`reverse`

Reverse the order of all the data points: the last one now becomes the first one, and so on. Though this has no effect on the look of the data, this will impact commands that work with indices, such as the display of `cut` and the multi-buffer processing commands (such as `subtract`, `div` and so on) with `/mode=indices`.

strip-if - Strip points

`strip-if` *formula*

- *formula*: Ruby boolean expression – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Removes all points for which the ruby expression returns `true`. This can be used for quite advanced data selection:

```
QSoas> strip-if 'x > 2'
```

This removes all points whose X value is over 2.

```
QSoas> strip-if 'x * y < 10 && x > 2'
```

This removes all the points for which both the X value is over 2 and the product of X and Y is below 10.

When reading data files that contain spurious data points (such as text lines containing no data within a file read with `load-as-text`), QSoas replaces the missing data by weird numbers called NaN (Not a Number). They can be useful at times, but mess up statistics and fits. To remove them, use:

```
QSoas> strip-if '(x != x) || (y != y)'
```

integrate - Integrate

`integrate /index=`*integer*

- `/index=`*integer*: Index of the point that should be used as y = 0 – values: an integer

Integrate just does the reverse of `diff` and integrates the current buffer. First data point is the one for which Y=0, unless an index is specified to the `/index` option, in which case the numbered point ends up being at 0.

diff - Derive

`diff`

Computes the 4th order accurate derivative of the buffer.

This is efficient to compute the derivative of smooth data, but it gives very poor results on noisy data. In general, for derivation, prefer `filter-fft`, `filter-bsplines` or `auto-reglin` that will give much better results.

diff2 - Derive twice

`diff2`

Computes the 4th order accurate second derivative of the buffer.

The same warnings apply as for `diff`.

dataset-options - Options

`dataset-options` `/yerrors=`*column* `/histogram=`*yes-no*

- `/yerrors=`*column*: name of the column containing y errors – values: the number/name of a column in a buffer
- `/histogram=`*yes-no*: whether to show as a histogram (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Sets options for the current dataset:

- with `/yerrors`, sets the display of errors on Y values, see there for more information on how to specify the columns;
- with `/histogram`, sets wether or not the dataset should be displayed as a histogram.

edit-errors - Edit errors

`edit-errors` **(interactive)**

Provides an interface for editing manually the errors attached to each point of the current buffer. This function will create a column containing errors if there is not one yet.

Pick left and right bounds and set the errors within the bounds with `i` and outside with `o`.

Splitting the dataset in bits (and back)

cut - Cut

`cut` **(interactive)**

Short name: `c`

Interactively cuts bits out of the buffer. Left and right mouse clicks set the left and right limits. Middle click or q quits leaving only the part that is within the region, while u leaves only the outer part. Hit escape to cancel.

chop - Chop Buffer

`chop` *lengths...* `/mode=`*choice* `/flags=`*words* `/set-segments=`*yes-no*

- *lengths...*: Lengths of the subsets – values: several floating-point numbers, separated by ,
- `/mode=`*choice*: Whether to cut on index or x values (default) – values: one of: `deltax`, `index`, `indices`, `xvalues`
- `/flags=`*words*: Buffers to flag/unflag – values: several words, separated by ‘,’
- `/set-segments=`*yes-no*: Whether to actually cut the dataset, or just to set segments where the cuts would have been – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Cuts the buffer into several parts based on the numbers given as arguments, and save them as separate buffers. The intepretation of the numbers depends on the value of the `/mode` option:

- `deltax` (default): the numbers are the length (in terms of X) of the sub-buffers
- `xvalues`: the numbers are the X values at which to split
- `index` (or `indices`): the numbers are the indices of the points at which to split

If `/set-segments` is on, the X values not used to create independent buffers but rather to set the position of the segments.

splita - Split first

`splita`

Returns the first part of the buffer, until the first change of sign of Δx.

Useful to get the forward scan of a cyclic voltammogram.

splitb - Split second

`splitb`

Returns the part of the buffer after the first change of sign of Δx.

Useful to get the backward scan of a cyclic voltammogram.

split-monotonic - Split into monotonic parts

`split-monotonic /flags=`*words* `/group=`*integer*

- `/flags=`*words*: Flags to set on the results – values: several words, separated by ‘,’
- `/group=`*integer*: Group that many segments into one dataset – values: an integer

Splits the buffer into buffers where all parts have X values that increase or decrease monotonically.

unwrap - Unwrap

`unwrap`
This command makes the X values of the current buffer monotonic by ensuring that the value of Δx always have the same sign, changing it if needs be.
This is useful for instance to convert a cyclic voltammogram from $i = f(E)$ to $i = f(t)$, after dividing by the scan rate.

cat - Concatenate

`cat` *first second…* `/add-segments=`*yes-no*
Short name: `i`

- *first*: First buffer – values: the number of a buffer in the stack
- *second…*: Second buffer(s) – values: comma-separated lists of buffers in the stack, see buffers lists
- `/add-segments=`*yes-no*: If on (default) segments are added between the old buffers – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Concatenates the buffers given as arguments, adding segment stops inbetween (unless `/add-segments=false` is used). This can be used to reverse the effect of the previous commands.
This does not change the number of columns. If you’re trying to gather several Y columns as a function of the same X, you should try `contract` instead.

Buffer’s meta-data and perpendicular coordinates

QSoas’s buffers hold more than just columns of numbers. When a file is loaded, QSoas also gathers as much information as possible about that file, such as original file name, file date, and, for files supported by QSoas, details about the experimental conditions recorded in that file. These are known as “meta-data”, and can be displayed using the `show` command.
Here are some meta-data of particular signification available to all buffers loaded from files:

- `file-date` is the date of the file
- `age` is the how old the file was in **seconds** when the current `QSoas` session was started.
- `commands` is the list of commands that have been applied to this buffer since its load/creation.

Upon saving using `save` all of a buffer’s meta-data are saved as comments in the text file.
Perpendicular coordinates make sense when a buffer has several Y columns. For instance, when data consists in spectra taken at different times after the beginning of an experiment, like in the tutorial (or at different solution potentials for a redox titration), then the X values will be the wavelength, and each Y column will correspond to a different time. Then the time is the *perpendicular coordinate*. One can set the perpendicular coordinate manually using `set-perp`.
Many commands make use of perpendicular coordinates, most notably `transpose` (that would convert columns of $y = f(\lambda)$ for different values of t above into columns of $y = f(t)$ for different values of λ), and all the multi-fit commands that show parameters as a function of the perpendicular coordinates when applicable.

Selecting datasets and files based on meta-data

Some commands, namely `flag`, `unflag` and `browse` accept a `/for-which` option to select the datasets (or files) they work on based on their properties. The value of the `/for-which` is a ruby formula that uses the global variables `$meta` and `$stats` variables. For instance, the following command flags all the datasets in the stack that have a maximum value above `1e-4`:

```
QSoas> flag all /for-which '$stats["y_max"] >= 1e-4'
```

Be mindful of the quotes: the outer `'` quotes are meant to protect the `$stats["y_max"] >= 1e-4` bit that is passed on to ruby. The quotes around `y_max` are necessary, as `$stats` is a dictionnary whose keys are ruby strings, hence the double quotes.

show - Show information

`show` *buffers...*

- *buffers...*: Buffers to show – values: comma-separated lists of buffers in the stack, see buffers lists

This command gives detailed information about the buffers given as arguments, such as the number of rows, columns, segments, but also the flags the buffer may have, and all its meta-data:

```
QSoas> show 0
Dataset 08.oxw: 2 cols, 4975 rows, 1 segments
Flags:
Meta-data: delta_t_0 = 950 gpes_file = D:\Vincent\140428\08 original-file = /home/vincent/Data/140428/08.oxw
age = 428907.581 steps = 1 title =
file-date = 2014-05-23T21:23:38 exp-time = 14:03:08 comments =
t_0 = 0 E_0 = -0.65 method = chronoamperometry
```

set-meta - Set meta-data

`set-meta` *name value*

- *name*: The name of the meta-data – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- *value*: The meta-data value – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Using `set-meta`, one can set the value of the named meta-data for the current buffer. *name* can have any value, it does not have to exist in the list of buffer's meta-data.

set-perp - Set perpendicular

`set-perp` *coords...*

- *coords...*: The values of the coordinates (one for each Y column) – values: several floating-point numbers, separated by ,

Sets the perpendicular coordinates for the Y columns, as comma-separated values. For it to be useful, there must be as many perpendicular coordinates as there are Y columns.

transpose - Transpose

`transpose`

This command transposes the matrix of the Y columns, while paying attention to the perpendicular coordinates. In short, if one starts from a series of Y columns representing spectra as a function of λ (the X column) for different values of time (each column at at different value of t), then after `transpose`, the new dataset contains columns describing the time evolution of the absorbance for different values of λ (one for each column).

tweak-columns - Tweak columns

`tweak-columns /remove=`*columns* `/flip=`*yes-no* `/flip-all=`*yes-no*

- `/remove=`*columns*: The column numbers to remove (X = 1, Y = 2, etc...) – values: a comma-separated list of columns
- `/flip=`*yes-no*: If true, flips all the Y columns – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/flip-all=`*yes-no*: If true, flips all the columns, including the X column – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

`tweak-columns` provides basic modifications of columns. If a list of columns is given to the `/remove` option, then the given columns are removed. If `/flip` is on, then all Y columns are reversed. If `/flip-all` is on, then all columns, including the X column, are reversed.

Data filtering/processing

QSoas provides different ways to process data to remove unwanted noise:

- Fourier transform filtering using `filter-fft` or `auto-filter-fft`.
- Data approximation using basis splines via `filter-bsplines` or `auto-filter-bs`.
- Splike removals using `remove-spikes` or even `deldp`.

In addition, QSoas provides ways to remove calculated "baselines":

- baselines interpolated from datapoints with `baseline`
- baselines interpolated between two segments using either a cubic function or an exponential function with `catalytic-baseline`

filter-fft - FFT filter

`filter-fft /derive=`*integer* **(interactive)**

- `/derive=`*integer*: The starting order of derivation – values: an integer

Filter data using FFT, ie the data is Fourier transformed, then a filter function is applied in the frequency domain and the result is backward transformed.

The cutoff can be changed using the mouse left/right buttons. The power spectrum can be displayed using the `p` key, and the derivative can be displayed with `d` (in which case you get the derivative of the signal when accepting the data).

Behind the scenes, a cubic baseline is computed and subtracted from the data to ensure that the data to which the FFT is applied has 0 value and 0 derivative on both sides. This greatly reduces artifacts at the extremities of the dataset. This baseline is computed using a small heuristic. You can display it using the `b` key.

filter-bsplines - B-Splines filter

`filter-bsplines /weight-column=`*column* **(interactive)**

- `/weight-column=`*column*: Use the weights in the given column – values: the number/name of a column in a buffer

Filters the data using B-splines: B-splines are polynomial functions of a given order defined over segments. The filtering process finds the linear combination of these splines functions that is the closest to the original data.

This approach amounts to taking the projection of the original data onto the subspace of the polynomial functions.

More information about the polynomial splines used can be found in the GSL documentation.

The result can be tuned by placing "nodes", ie the X positions of the segments over which the splines are defined. Put more nodes in an area where the data is not described properly by the smoothed function. Increasing the order (using `+`) may help too.

Like for `filter-fft`, you can derive the data as well pushing the `d` key.

Hitting the `o` key optimizes the position of the segments in order to minimize the difference between the data and the approximation. (be careful as this function may fail at times).

auto-filter-bs - Auto B-splines

`auto-filter-bs /number=`*integer* `/order=`*integer* `/weight-column=`*column* `/derivatives=`*integer*

Short name: `afbs`

- `/number=`*integer*: Number of segments – values: an integer
- `/order=`*integer*: Order of the splines – values: an integer
- `/weight-column=`*column*: Use the weights in the given column – values: the number/name of a column in a buffer
- `/derivatives=`*integer*: Compute derivatives up to the given – values: an integer

Filters the data using B-splines in a non-interactive fashion. Performs automatically an optimization step.

This is mostly useful in scripts.

auto-filter-fft - Auto FFT

`auto-filter-fft /cutoff=`*integer* `/derive=`*integer*

Short name: `afft`

- `/cutoff=`*integer*: value of the cutoff – values: an integer
- `/derive=`*integer*: differentiate to the given order – values: an integer

Filters data using FFT in a non-interactive fashion. Useful in scripts.

auto-reglin - Automatic linear regression

`auto-reglin /window=`*integer*

- `/window=`*integer*: Number of points (after and before) over which to perform regression – values: an integer

Performs a linear regression on a number of points around each point of the graph and creates a buffer from the resulting slopes, which results in a derivative buffer. This command is similar to but provides less noisy output than `diff`, and also similar to filtering with FFT (using `filter-fft`) and taking the derivative.

remove-spikes - Remove spikes

`remove-spikes /number=`*integer* `/factor=`*number* `/force-new=`*yes-no*

Short name: `R`

- `/number=`*integer*: looks at that many points – values: an integer
- `/factor=`*number*: threshold factor – values: a floating-point number
- `/force-new=`*yes-no*: creates a new buffer even if no spikes were removed (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Remove spikes using a simple heuristic. This command will not create a new buffer if not spikes were removed, unless you specify `/force-new=true`.

downsample - Downsample

`downsample /factor=`*integer*

- `/factor=`*integer*: Downsampling factor – values: an integer

Creates a buffer with about *factor* times less points than the original buffer (default 10 times less) by averaging the original X and Y values in groups of *factor*. This command averages the other columns too.

baseline - Baseline

`baseline` **(interactive)**
Short name: `b`
Draw a baseline by placing markers on the curve using the mouse (or off the curve, after using key `o`). Baseline is computing using one of several interpolation algorithms: C-splines, linear or polynomial interpolation and Akima splines (the latter usually follows best the accidents on the curve). Cycle between the various schemes by hitting `t`.
It is possible to leave saving not the interpolated data, but just the interpolation "nodes'' (ie the big dots), by pushing the `p` key. This has two advantages: first, one can load nodes from a buffer by hitting the `L` key and providing the buffer number (or just their X value by hitting `l`). Second, if one has the nodes and just the X values, one can generate the interpolated data using `interpolate`.

interpolate - Interpolate

`interpolate` *xvalues nodes* `/type=`*choice*

- *xvalues*: Buffer serving as base for X values – values: the number of a buffer in the stack
- *nodes*: Buffer containing the nodes X/Y values – values: the number of a buffer in the stack
- `/type=`*choice*: Interpolation type – values: one of: Akima spline, C-spline, linear, polynomial

Given a buffer containing *xvalues* and another one containing the X/Y position of interpolation nodes saved using `p` from within `baseline`, this command regenerates the interpolated values, for the given X values.
Through this approach, one can draw a baseline, save the points, generate the baseline-subtracted data using `interpolate` from within a script. This has the advantage that one can always have a close look at the quality of the baseline, and tweak it if need be.

catalytic-baseline - Catalytic baseline

`catalytic-baseline` **(interactive)**
Short name: `B`
Draws a so-called "catalytic" baseline. There are several types of baselines, but they all share the following features:

- they are defined by 4 points
- the first two points correspond to points where the baseline sticks to the data
- the last two points give a "direction"

There are two baselines implemented for now:

- a cubic baseline, that goes through the first two points and is parallel to the slope of the last two
- an exponential baseline, that goes through the first two points and has the same ratio as the data for the last two points

auto-correlation - Auto-correlation

`auto-correlation`
Short name: `ac`
Computes the auto-correlation function of the data, using FFT.

bin - Bin

`bin /boxes=`*integer* `/column=`*column* `/log=`*yes-no*

- `/boxes=`*integer*: – values: an integer
- `/column=`*column*: – values: the number/name of a column in a buffer
- `/log=`*yes-no*: – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Creates an histogram by binning the Y values (or the values of the column given by the `/column` option, see above) into various boxes (whose number can be controlled using the `/boxes` option). The new buffer has for X values the center of the boxes and as Y values the number of data points that were in the boxes.

Segments

It is possible to split a buffer into logical segments without changing the contents of the buffer, but the position of the segment boundaries are marked by a vertical line. They can be used for different purposes: for segment-by-segment operations, step-by-step film loss correction (using `film-loss`) or buffer splitting (using `segments-chop`).

Segments can be detected using `find-steps`, or set manually using `set-segments` or `chop`.

find-steps - Find steps

`find-steps` `/average=`*integer* `/threshold=`*number* `/set-segments=`*yes-no*

- `/average=`*integer*: Average over that many points – values: an integer
- `/threshold=`*number*: Detection threshold – values: a floating-point number
- `/set-segments=`*yes-no*: Whether or not to set the dataset segments – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

This function detects "jumps" in the data (such as potential changes in a chronoamperometry experiment, for instance), and display them both to the terminal output and on the data display.

By default, this function only shows the segments it finds, but if the option `/set-segments` is on, the segments are set to that found by `find-steps` (removing the ones previously there).

set-segments - Set segments

`set-segments` **(interactive)**

Interactively prompts for the addition/removal of segments. A left click adds a segment where the mouse is, while a right click removes the closest segment.

segments-chop - Chop into segments

`segments-chop`

Cuts the buffer into several ones based on the segments defined in the current buffer. This way, the effect of a `chop` `/set-segment=true` followed by `segments-chop` is the same as the `chop` without `/set-segment=true`.

film-loss - Film loss

`film-loss` **(interactive)**

Applies stepwise film loss correction (in the spirit of the K_m experiments in Fourmond *et al*, **Anal. Chem.**, *2009*). For that, the current buffer must be separated into segments, using `set-segments`, for instance. `QSoas` then zooms on the first segment. Right and left clicking around the final linear decay will set the value of the film loss rate constant for this step. Push space to switch to the next step, and when you have done everything, push `q` to obtain the corrected data.

Operations involving several buffers

It is possible to combine several buffers into one by applying mathematical operations (subtraction, division and the like). Each of these processes involve matching a point in a given buffer to a point in another buffer. There are two ways to do that, chosen by the `/mode` option:

- with `/mode=xvalues`, the default, uses the values of X (ie the closest X value is picked). **Warning** this doesn't give the expected result for buffers with several times the same X values, like cyclic voltammograms.
- with `/mode=indices`, points are matched on a one-to-one basis, ie point 1 of buffer 1 to point 1 of buffer 2, irrespective of the X values.

In addition to that, the operations can make use of the segments defined on each buffer (see `find-steps` and `set-segments`). If segments are defined and `/use-segments=true`, then the operations are applied segment-by-segment, with the first point of each segment matching the corresponding point in the other buffer. This mode is particularly useful to manipulate series of potential steps with different time intervals.

div - Divide

`div` *first…* *second* `/mode=`*choice* `/use-segments=`*yes-no*

- *first…*: First buffer(s) – values: comma-separated lists of buffers in the stack, see buffers lists
- *second*: Second buffer – values: the number of a buffer in the stack
- `/mode=`*choice*: Whether operations try to match x values or indices – values: one of: `indices`, `xvalues`
- `/use-segments=`*yes-no*: If on, operations are performed segment-by-segment – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Divides all buffers by the last one. This is useful to get rid of film loss when one has an independent measure of film loss, see Fourmond *et al*, **Anal. Chem.** 2009 for more information.

subtract - Subtract

`subtract` *first...* *second* `/mode=`*choice* `/use-segments=`*yes-no*
Short name: `S`

- *first...*: First buffer(s) – values: comma-separated lists of buffers in the stack, see buffers lists
- *second*: Second buffer – values: the number of a buffer in the stack
- `/mode=`*choice*: Whether operations try to match x values or indices – values: one of: `indices`, `xvalues`
- `/use-segments=`*yes-no*: If on, operations are performed segment-by-segment – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Subtracts the last buffer from all the other ones (there can be more than one *first* buffer). Useful for standard baseline removal.

average - Average

`average` *buffers...* `/mode=`*choice* `/use-segments=`*yes-no* `/split=`*yes-no* `/count=`*yes-no*

- *buffers...*: Buffers – values: comma-separated lists of buffers in the stack, see buffers lists
- `/mode=`*choice*: Whether operations try to match x values or indices – values: one of: `indices`, `xvalues`
- `/use-segments=`*yes-no*: If on, operations are performed segment-by-segment – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/split=`*yes-no*: If on, buffers are automatically split into monotonic parts before averaging. – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/count=`*yes-no*: If on, a last column contains the number of averaged points for each value – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

In a manner similar to `subtract` and `div`, the `average` command averages all the buffers given into one, with the same segment-by-segment capacities.

An additional feature of `average`, though is its ability to first split the buffers into monotonic parts before averaging (when `/split` is on). That is even the default in the case when only one buffer is provided. This proves useful for averaging the forward and return scan in a cyclic voltammogram.

merge - Merge buffers on X values

`merge` *first...* *second* `/mode=`*choice* `/use-segments=`*yes-no*

- *first...*: First buffer(s) – values: comma-separated lists of buffers in the stack, see buffers lists
- *second*: Second buffer – values: the number of a buffer in the stack
- `/mode=`*choice*: Whether operations try to match x values or indices – values: one of: `indices`, `xvalues`
- `/use-segments=`*yes-no*: If on, operations are performed segment-by-segment – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Merge the second buffer with the first one, and keep Y of the second as a function of Y of the first. The algorithm for finding which point in the second corresponds to a given one in the first is the same as that of the other buffers.

If more than two buffers are specified, the last one gets merged with each of those before.

contract - Group buffers on X values

`contract` *buffers...* `/mode=`*choice* `/use-segments=`*yes-no* `/perp-meta=`*text* `/use-columns=`*columns*

- *buffers...*: Buffers to contract – values: comma-separated lists of buffers in the stack, see buffers lists
- `/mode=`*choice*: Whether operations try to match x values or indices – values: one of: `indices`, `xvalues`
- `/use-segments=`*yes-no*: If on, operations are performed segment-by-segment – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: Define the perpendicular coordinate from meta-data – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/use-columns=`*columns*: If specified, use only the given columns for the contraction – values: a comma-separated list of columns

As may be anticipated from the name, `contract` does the reverse of `expand`, ie it regroups in one buffer several values of Y that run against the same values of X. The result is a buffer that contains as many Y columns as the total of Y columns of all the arguments. X matching between the buffers is done as for the other operations such as `div` or `subtract`.

You can specify a column list using `/use-columns` (see above for more information about column lists), in which case the other columns from the buffers are ignored.

Data inspection facilities

find-peaks - Find peaks

`find-peaks /window=`*integer* `/include-borders=`*yes-no* `/which=`*choice* `/output=`*yes-no* `/peaks=`*integer*

- `/window=`*integer*: Width of the window – values: an integer
- `/include-borders=`*yes-no*: Whether or not to include borders – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/which=`*choice*: Selects which of minima and/or maxima to display – values: one of: `both`, `max`, `min`
- `/output=`*yes-no*: Whether peak information should be written to the output file (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/peaks=`*integer*: Display only that many peaks (by order of intensity) – values: an integer

Find all the peaks of the current dataset. Peaks are local extrema over a window of a number of points given by `/window` (8 by default). If `/output` is on, then the peak data is written to the output file. This function will find many peaks on noisy data, you can limit to the first *n* ones by using `/peaks=`*n* (peaks are ranked by amplitude).

By default, if a point at either end of the dataset is an extremum, it is not included, unless you use `/include-borders=true`.

Peaks are indicated on the buffer using lines, and their position is written to the terminal, if `/output` is on (off by default).

echem-peaks - Find peaks pairs

`echem-peaks`

This function tries to find "pairs" of peaks that may be the anodic and cathodic peaks of a redox couple, and outputs useful information about those.

1 - Find peak

`1 /window=`*integer* `/include-borders=`*yes-no* `/which=`*choice*

- `/window=`*integer*: Width of the window – values: an integer
- `/include-borders=`*yes-no*: Whether or not to include borders – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/which=`*choice*: Selects which of minima and/or maxima to display – values: one of: `both`, `max`, `min`

Essentially equivalent to

```
QSoas> find-peaks /peaks=1 /output=true
```

2 - Find two peaks

`2 /window=`*integer* `/include-borders=`*yes-no* `/which=`*choice*

- `/window=`*integer*: Width of the window – values: an integer
- `/include-borders=`*yes-no*: Whether or not to include borders – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/which=`*choice*: Selects which of minima and/or maxima to display – values: one of: `both`, `max`, `min`

Essentially equivalent to

```
QSoas> find-peaks /peaks=2 /output=true
```

stats - Statistics

`stats /buffer=`*buffer* `/output=`*yes-no* `/use-segments=`*yes-no* `/meta=`*words*

- `/buffer=`*buffer* (default option): An alternative buffer to get information on – values: the number of a buffer in the stack
- `/output=`*yes-no*: whether to write stats to output file (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/use-segments=`*yes-no*: Makes statistics segment by segment (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/meta=`*words*: When writing to output file, also print the listed meta-data – values: several words, separated by ','

`stats` displays various statistics about the current buffer (or the one specified as the `/buffer` option). The exact list of statistics displayed grows slowly with time and needs. It is currently, for each column:

- the first and last values (displayed with the index)
- the `_min` and `_max` values
- the average
- `_var`, the variance
- the `_norm`, ie the square root of the sum of the squares
- for Y columns, the integral is also displayed, indexed by `_int`.

Statistics can be written to the output file with `/to-file=true`. If you specify `/use-segments=true`, the statistics are also displayed segment-by-segment (and written out to the output file if asked for). If you want some meta-data to be written to the output file together with the statistics, provide them as a comma-separated list to the `/meta` option, or, alternatively, use the `/meta` option of the `output` command.

cursor - Cursor

`cursor` **(interactive)**
Short name: `cu`
Position a cursor on the curve to know its exact X and Y positions.
Using the right mouse button, it is also possible to position a reference point. After that the difference in X,Y coordinates between the cursor and the reference point is also displayed (and the ratios).
Cursor positions can be saved to the output file by pressing the space bar.
Hitting `u` subtracts the Y value of the current point to the Y values of the buffer and returns. Hitting `v` divides by the current Y value.

reglin - Linear regression

`reglin` **(interactive)**
Short name: `reg`
Linear regression. Using the left and right mouse buttons, select a region whose slope is of interest. The terminal shows the a and b parameters (the equation is $ax + b$), and also the effective first order rate constant, ie the k_{eff} parameter of the equation

$$f_0 \exp\left[-k_{eff}\,(x - x_0)\right]$$

whose first-order expansion gives the same linear approximation, ie:

$$f_0 \times \left[1 - k_{eff}(x - x_0))\right] = ax + b$$

Using the space bar it is possible to save the values displayed in the terminal to the output file.

Fits

QSoas was designed with a particular emphasis on fitting data. It allows complex fits, and in particular multi-buffer fits, when functions with shared parameters are fit to different buffers. Fits fall into two different categories:

- mono-buffer fits, ie fits that apply to one buffer, but that can be applied to several buffers at the same time with shared parameters
- multi-buffer fits, ie fits that need at least two buffers to work

Fits can be used through several commands: for all fits there are a `mfit-`*fit* and a `sim-`*fit* command, and for mono-buffer fits, there is a `fit-`*fit* in addition.

- The `fit` command fits a single buffer, when the fits allows that. It takes no argument
- The `mfit` command fits several buffers at the same time. It takes the numbers of the buffers it will work on.
- The `sim` command takes a saved parameters file and a series of buffers, and pushes the data computed from the parameters on the stack using the X values of the buffers given as arguments (their Y values are not used).

In addition to these commands, QSoas provides commands to combine fits together, to fit derivatives of the signals, and to define fits with distributions of parameters.
All fits commands share several common options, which are detailed here:

- With the `/extra-parameters` option, on defines additional parameters to the fit, that can be used to define parameters by formulas
- Passing the name of a saved parameters file to the `/parameters` option preloads the given parameters at the beginning of the fit.
- The `/set-from-meta` option makes it possible to set a value of parameters from meta-data. For instance, running a fit with `/set-from-meta=v=sr` will set the value of the parameter `v` to the value of the meta-data `sr` (if present). Specify more of those by separating them with commas.
- The `/debug` and `/debug2` options are essentially for debugging the fits and the fit engines, you should not need them.

combine-fits - Combine fits

`combine-fits` *name formula fits...*

- *name*: The name of the new fit – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- *formula*: How to combine the various fits – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- *fits...*: The fit to combine together – values:

Creates a new fit named *name* based on other fits, combined through a formula. The formula use `y1`, `y2` and so on to refer to the fits. You specify the fit names by removing the `fit-` or `mfit-` prefix. For instance, to fit a sum of lorentzians and gaussians, one just has to do:

```
QSoas> combine-fits lg 'y1 + y2' lorentzian gaussian
```

This creates a new fit, `lg`, and hence three new commands, `fit-lg`, `mfit-lg` and `sim-lg`. The fit is a sum of a lorentzian fit (`y1`) and a gaussian fit (`y2`). The new fit shares the options of all the original fits.

The newly-defined fit only lasts for the current session, if you need something more persistent, consider setting up a startup file using `startup-files`.

define-derived-fit - Create a derived fit

`define-derived-fit` *existing-fit* `/mode=`*choice*

- *existing-fit*: name of the fit to make a derived fit of – values: the name of a fit (without the fit- prefix)
- `/mode=`*choice*: Whether one fits only the derivative, both the derivative and the original data together or separated – values: one of: `combined`, `deriv-only`, `separated`

Defines a new fit based on the already defined *fit* (just the fit name, that is without the `fit-` prefix) that will fit:

- only the derivative if `/mode=deriv-only`, in which case is is named `fit-deriv-only-`*fit*;
- a multibuffer fit for the original function in one buffer and the derivative in the second if `/mode=separated` (the default mode), in which case the fit is named `mfit-deriv-`*fit*;
- both the original function and the derivative in a single buffer (the derivative is assumed to be the data after the first discontinuity in the X values) if `/mode=combined`, in which case the new fit is named `fit-deriv-combined-`*fit*;

This function is explained in more details in the tutorial.

define-distribution-fit - Define fit with distribution

`define-distribution-fit` *name existing-fit parameter* `/distribution=`*choice*

- *name*: name of the new fit – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- *existing-fit*: name of fit to make a derived fit from – values: the name of a fit (without the fit- prefix)
- *parameter*: the parameter over which to integrate – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/distribution=`*choice*: The default distribution – values: one of: `gaussian`, `k0`, `lorentzian`, `uniform`

Defines a new fit called *name* based on the fit *fit* in which the data is the result of the integration of the original fit over a distribution of *parameter*.

The choice for the distribution is given through the `/distribution=` option (also available to the created fit), and is one of:

- `gaussian`: gaussian distribution of the parameters
- `lorentzian`: lorentzian distribution of the parameters
- `k0`: "dispersion of interfacial rate distribution", defined by a uniform probability between two values for the logarithm of the parameter, see Léger, *et al*, **J. Phys. Chem. B** 2002 for more details
- `uniform`: for a uniform probability between two values

Of course, even for theoretically infinite distributions (`gaussian` and `lorentzian` distributions above), `QSoas` does not integrate over the whole real axis, which is why these distributions get an extra parameter, fixed by default, that indicates the extent of the integration interval in dimensionless units (independent of the value of the parameter). In principle, these values are chosen as a good compromise between accuracy and computing time, but they can be tuned should you need it.

Exponential fits

There are several ways to fit exponentials to data. The simplest is by far the `fit-exponential-decay` that fits a decay with an arbitrary number of exponentials to the data.

fit-exponential-decay - Fit: Multi-exponential fits

`fit-exponential-decay` `/exponentials=`*integer* `/absolute=`*yes-no* `/loss=`*yes-no* `/slow=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* **(interactive)**

- `/exponentials=`*integer*: Number of exponentials – values: an integer
- `/absolute=`*yes-no*: whether the amplitude is absolute or relative to the asymptote (defaults to true) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/loss=`*yes-no*: wether the sum of exponentials should be multiplied by an exp(-kt) function (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/slow=`*yes-no*: whether there is a very slow phase (that shows up as a linear change in Y against time, defaults: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’

Fits to the formula:

$$\left(A_\infty + \sum_{i=1}^{n} A_i \exp(-(x - x_0)/\tau_i) + b\,x\right) \exp(-k_{loss}(x - x_0))$$

$b\,x$ is only present if the `/slow` option is on, and k_{loss} is not 0 only if `/loss` is on. If `/relative` is on, the parameter of the fit is α_i (defined by $A_i = \alpha_i A_\infty$) rather than A_i. `/relative=true` should not be used to fit data that tend to 0.

mfit-exponential-decay - Multi fit: Multi-exponential fits

`mfit-exponential-decay` *datasets…* `/exponentials=`*integer* `/absolute=`*yes-no* `/loss=`*yes-no* `/slow=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *datasets…*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/exponentials=`*integer*: Number of exponentials – values: an integer
- `/absolute=`*yes-no*: whether the amplitude is absolute or relative to the asymptote (defaults to true) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/loss=`*yes-no*: wether the sum of exponentials should be multiplied by an exp(-kt) function (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/slow=`*yes-no*: whether there is a very slow phase (that shows up as a linear change in Y against time, defaults: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance

Multi-buffer version of the `fit-exponential-decay` fit.

sim-exponential-decay - Simulation: Multi-exponential fits

`sim-exponential-decay` *parameters datasets…* `/exponentials=`*integer* `/absolute=`*yes-no* `/loss=`*yes-no* `/slow=`*yes-no* `/override=`*text* `/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *parameters*: File to load parameters from – values: name of a file
- *datasets…*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/exponentials=`*integer*: Number of exponentials – values: an integer
- `/absolute=`*yes-no*: whether the amplitude is absolute or relative to the asymptote (defaults to true) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/loss=`*yes-no*: wether the sum of exponentials should be multiplied by an exp(-kt) function (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/slow=`*yes-no*: whether there is a very slow phase (that shows up as a linear change in Y against time, defaults: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ‘,’
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Simulation command for the `fit-exponential-decay` fit.

fit-multiexp-multistep - Fit: Multi-step and multi-exponential

`fit-multiexp-multistep` `/exponentials=`*integer* `/steps=`*integers* `/independent=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* **(interactive)**

- `/exponentials=`*integer*: Number of exponentials – values: an integer
- `/steps=`*integers*: Step list with numbered conditions – values: a comma-separated list of integers
- `/independent=`*yes-no*: Whether irreversible loss is independent on each step – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','

This fit is an extension of the `exponential-decay` fit when the experiment consists in several steps in which the time constants are expected to change, but some may be common to different steps. The steps are specified using the `/steps` option. Specifying `/steps=0,1,0` means that there are three steps, but there are only two distinct sets of time constants, a first one (`0`, used for step 1 and 3), and a second one (`1`, used only for step 2).

In each of the steps, the formula fitted to the data is:

$$y(t) = I^{k(j)}\left(1 - \sum_{i=1}^{n} \alpha_i^j \exp(-(t - t_0^j)/\tau_i^{k(j)})\right) \exp(-k_{loss}^{k(j)}(t - t_0^j)) \times \alpha_j$$

Where j is the step number, $k(j)$ is the number of the corresponding time constants, t_0^j is the beginning of the step j, the α_i^j are the relative amplitudes of the exponential phases, the $I^{k(j)}$ are the asymptotic values of $y(t)$ on each step (in the absence of film loss) and α_j is defined recursively by $\alpha_0 = 1$ and $\alpha_j = \alpha_{j-1} \exp(-k_{loss}^{k(j-1)}(t_0^j - t_0^{j-1}))$. This is done so as to keep track of the film loss over the whole.

mfit-multiexp-multistep - Multi fit: Multi-step and multi-exponential

`mfit-multiexp-multistep` *datasets...* `/exponentials=`*integer* `/steps=`*integers* `/independent=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/exponentials=`*integer*: Number of exponentials – values: an integer
- `/steps=`*integers*: Step list with numbered conditions – values: a comma-separated list of integers
- `/independent=`*yes-no*: Whether irreversible loss is independent on each step – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

This is the multibuffer version of the `multiexp-multistep` fit.

sim-multiexp-multistep - Simulation: Multi-step and multi-exponential

`sim-multiexp-multistep` *parameters datasets...* `/exponentials=`*integer* `/steps=`*integers* `/independent=`*yes-no* `/override=`*text* `/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *parameters*: File to load parameters from – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/exponentials=`*integer*: Number of exponentials – values: an integer
- `/steps=`*integers*: Step list with numbered conditions – values: a comma-separated list of integers

- `/independent=`*yes-no*: Whether irreversible loss is independent on each step – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ','
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

This is the computation command for the `multiexp-multistep` fit.

Arbitrary fits

QSoas provides ways to fit arbitrary formulas (written in Ruby) to data. While it is possible to do that on a case-by-case basis using `fit-arb`, it is also possible to store formulas in a plain text file and load them using `load-fits` or define a new one using `custom-fit`.

fit-arb - Fit: Arbitrary fit

`fit-arb` *formulas* `/with=`*time-dependent parameters* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* **(interactive)**

- *formulas*: |-separated formulas for the fit – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/with=`*time-dependent parameters*: Make certain parameters depend upon time – values: several specifications of time dependent parameters (like `co:2,exp`), seperated by ';'. Available types: steps, exp, rexp
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','

Fits *formula* (a piece of Ruby code) to the current buffer.
Parameters are auto-detected. Some parameters are treated specifically:

- `x_0` and `y_0` are fixed by default and initialized to the first X or Y value of the buffer the fit applies to;
- `temperature` is also fixed and set to the current temperature
- Using `fara` counts as using `temperature` excepted that its value is $f = F/RT$. You never get `fara` as a fit parameter.

If you often use the same formula for `fit-arb`, you should consider using `custom-fit` or writing it in a file and loading that file with `load-fits`.

Starting from `QSoas` version 2.0, you can use the `/with=` option to make some of the parameters dependent on time in a flexible fashion. See time dependent parameters below for more information.

mfit-arb - Multi fit: Arbitrary fit

`mfit-arb` *formulas datasets...* `/with=`*time-dependent parameters* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *formulas*: |-separated formulas for the fit – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- *datasets...*: datasets that will be fitted to – values: comma-separated lists of buffers in the stack, see buffers lists
- `/with=`*time-dependent parameters*: Make certain parameters depend upon time – values: several specifications of time dependent parameters (like `co:2,exp`), seperated by ';'. Available types: steps, exp, rexp
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Same as `fit-arb`, but for multiple buffers.
Using `mfit-arb`, it is possible to specify several formulas, separated by `|`.
If only one formula is specified, the same formula is applied to all buffers (with, as usual, the possibility to select which parameters are global or buffer-local).
If more than one formula is specified, the exact same number of buffers should be supplied; the first formula applies to the first buffer, the second formula to the second buffer, and so on... For instance, if you run:

```
QSoas> mfit-arb a*x+b|a*x+c|a*x+d 0 1 2
```

Buffer 0 get fits with `a*x+b`, 1 with `a*x+c` and 2 with `a*x+d`.
In this specific case, though, you could also have run

```
QSoas> mfit-arb a*x+b 0 1 2
```

and have `a` common to all buffer, but `b` buffer-specific.

load-fits - Load fits

`load-fits` *file*

- *file*: File containing the fits to load – values: name of a file

Load fits of arbitrary functions from a plain text file, and create the corresponding `fit-`, `mfit-` and `sim-` functions, that can be used with `define-derived-fit` or `combine-fits` for instance. Files should look like this:

```
# Comments are allowed
michaelis: vmax/(1 + km/x)
sigm-log: log((exp(a_red*log(10.0)) +exp(a_ox*log(10.0)) * \
          exp(-fara*(x-e0)))/ \
          (1 + exp(-fara*(x-e0))))
```

Comments are allowed, as are line continuations with \.

custom-fit - Define fit

`custom-fit` *name formula*

- *name*: Name for the new fit – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- *formula*: Mathematical expression for the fit – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Directly defines a custom fit with the given *name* and *formula*. Equivalent to having a line
in a file loaded by `load-fits`.

Peak fits

The fits in this section can be used to fit various "peaks" obeying to different distributions, such as the

- gaussian distribution `fit-gaussian`
- lorentzian distribution `fit-lorentzian`

For all these fits, you can specify the number of "peaks" using a common `/number` option. For each peak, there is a position, an amplitude and a width parameter. If you are more interested in the total surface under the peak rather than the amplitude of the peak, the fits provide a `/use-surface` argument that changes the amplitude parameter into a surface one.

fit-gaussian - Fit: One or several gaussians

`fit-gaussian` `/use-surface=`*yes-no* `/number=`*integer* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* **(interactive)**

- `/use-surface=`*yes-no*: Whether to use a surface or an amplitude parameter – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/number=`*integer*: Number of distinct peaks – values: an integer
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’

Fits a number of gaussians, given by:

$$\sum_i \frac{A_i}{\sqrt{2\pi{\sigma_i}^2}} \exp\left[-\frac{(x-x_i)^2}{2{\sigma_i}^2}\right]$$

More information in the GSL documentation.

The `/number` option controls the number of different peaks, while using `/use-surface=true` fits the surface of the peak instead of the amplitude.

mfit-gaussian - Multi fit: One or several gaussians

`mfit-gaussian` *datasets...* `/use-surface=`*yes-no* `/number=`*integer* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/use-surface=`*yes-no*: Whether to use a surface or an amplitude parameter – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/number=`*integer*: Number of distinct peaks – values: an integer
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance

Multi-buffer variant of the `fit-gaussian` fit.

sim-gaussian - Simulation: One or several gaussians

`sim-gaussian` *parameters datasets...* `/use-surface=`*yes-no* `/number=`*integer* `/override=`*text* `/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *parameters*: File to load parameters from – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/use-surface=`*yes-no*: Whether to use a surface or an amplitude parameter – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/number=`*integer*: Number of distinct peaks – values: an integer
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ‘,’
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Simulation command for the `fit-gaussian` fit.

fit-lorentzian - Fit: A Lorentzian (also named Cauchy distribution)

`fit-lorentzian` `/use-surface=`*yes-no* `/number=`*integer* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* **(interactive)**

- `/use-surface=`*yes-no*: Whether to use a surface or an amplitude parameter – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/number=`*integer*: Number of distinct peaks – values: an integer
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file

- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','

Fits a number of lorentzians, given by:

$$\sum_i \frac{A_i}{a_i \pi \left(1 + \left[(x - x_i)/a_i\right]^2\right)}$$

More information in the GSL documentation.

The `/number` option controls the number of different peaks, while using `/use-surface=true` fits the surface of the peak instead of the amplitude.

mfit-lorentzian - Multi fit: A Lorentzian (also named Cauchy distribution)

`mfit-lorentzian` *datasets...* `/use-surface=`*yes-no* `/number=`*integer* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/use-surface=`*yes-no*: Whether to use a surface or an amplitude parameter – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/number=`*integer*: Number of distinct peaks – values: an integer
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Multi-buffer variant of the `fit-lorentzian` fit.

sim-lorentzian - Simulation: A Lorentzian (also named Cauchy distribution)

`sim-lorentzian` *parameters datasets...* `/use-surface=`*yes-no* `/number=`*integer* `/override=`*text* `/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *parameters*: File to load parameters from – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/use-surface=`*yes-no*: Whether to use a surface or an amplitude parameter – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/number=`*integer*: Number of distinct peaks – values: an integer
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ','
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Simulation command for the `fit-lorentzian` fit.

Redox titration fits

fit-nernst - Fit: Nerstian behaviour

`fit-nernst` `/states=`*integers* `/species=`*integer* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=` **(interactive)**

- `/states=`*integers*: Number of redox states for each species – values: a comma-separated list of integers
- `/species=`*integer*: Number of distinct species (regardless of their redox state) – values: an integer

- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','

Fits the absorbance (or anything else) of a number of chemical species present under several redox states as a function of the potential, using the Nernst equation. The number of species is given to the `/species` option, while the number of redox states for each species is given to the `/states` option. Alternatively, if you need distinct species with a different number of redox states, you can specify a comma-separated list of number of states to `/states`, in which case `/species` is ignored. For instance, to fit two species, one presentin 4 redox states and the other in two redox states, one can use:

```
QSoas> fit-nernst /states=4,2
```

The species are designated using a lowercase letter suffix, while the redox state is designated using `red`, `int` or `ox` when there are 3 states or less, or with a number for more than three states.

Note: be aware that if there is more than one species, the system is intrinsically overdetermined, which is why `QSoas` automatically fixs the absorbance of the reduced species of all but the first one to 0 (but you can change that).

This fit is useful to fit the results of a the redox titration at a single wavelength. If several wavelength are available, separate them into several buffers as a function of the potential and fit them using `mfit-nernst`, while keeping the redox potentials (and electron numbers) global and only the absorbances as buffer-local.

mfit-nernst - Multi fit: Nerstian behaviour

`mfit-nernst` *datasets...* `/states=`*integers* `/species=`*integer* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/states=`*integers*: Number of redox states for each species – values: a comma-separated list of integers
- `/species=`*integer*: Number of distinct species (regardless of their redox state) – values: an integer
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Multi-buffer version of `fit-nernst`. To be used for fitting multi-wavelength redox titrations.

sim-nernst - Simulation: Nerstian behaviour

`sim-nernst` *parameters datasets...* `/states=`*integers* `/species=`*integer* `/override=`*text* `/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *parameters*: File to load parameters from – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/states=`*integers*: Number of redox states for each species – values: a comma-separated list of integers
- `/species=`*integer*: Number of distinct species (regardless of their redox state) – values: an integer
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ','
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Simulation command for `fit-nernst`

Adsorbed ideal redox species

fit-adsorbed - Fit: Adsorbed species

`fit-adsorbed` `/species=`*integer* `/2el=`*integer* `/distinct=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* **(interactive)**

- `/species=`*integer*: Number of 1-electron species – values: an integer
- `/2el=`*integer*: Number of true 2-electron species – values: an integer
- `/distinct=`*yes-no*: If true (default) then all species have their own surface concentrations – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’

Fits a series of “ideal adsorbed peaks” to the current buffer. The actual formula is the following:

$$i = \frac{F^2 \nu}{R\,T}\left(\sum_k {i_k^1}^{el} + \sum_{k'} {i_{k'}^2}^{el}\right)$$

The number of 1-electron peaks is given by the `/species` option (defaults to 1) and that of the 2-electrons peaks is given by the `/2el` option (defaults to 0).

The current for 1-electron peaks is given by:

$${i_k^1}^{el} = \frac{\Gamma_k n_k\, e_k}{(1+e_k)^2}$$

with $e_k = \exp\frac{F\,n_k(E-E_k^0)}{R\,T}$, with E_k^0 the potential of the couple and n_k the apparent number of electrons. The latter only affects the width of the peaks, the stoechiometry is always 1 electron.

The current for the 2-electrons peaks is given by Pilchon and Laviron, **J. Electronanal. Chem.**, *1976*:

$${i_{k'}^2}^{el} = \Gamma_{k'}\kappa_{k'}\frac{\epsilon_{k'} + 4/\kappa_{k'} + 1/\epsilon_{k'}}{(\epsilon_{k'} + \kappa_{k'} + 1/\epsilon_{k'})^2}$$

With $\epsilon_{k'} = \exp\frac{F\,(E-E_{k'}^0)}{R\,T}$, $E_{k'}^0$ being the 2-electrons reduction potential (i.e. the average of those of the 1-electron couples) and $\kappa_{k'} = \exp\frac{F\,\Delta E_{k'}^0}{R\,T}$, $\Delta E_{k'}^0$ being the difference in the reduction potentials of the 1-electron couples (it is positive if the intermediate species is thermodynamically stable).

The Γ parameters are the number of moles of the molecules adsorbed on the electrode (there is no explicit surface). If the option `/distinct=false` is used, the same value of Γ is used for all couples, while in the other case (the default), each couple has its own value of Γ (which corresponds to unrelated species). ν is the voltammetric scan rate (in volts per second).

mfit-adsorbed - Multi fit: Adsorbed species

`mfit-adsorbed` *datasets...* `/species=`*integer* `/2el=`*integer* `/distinct=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/species=`*integer*: Number of 1-electron species – values: an integer
- `/2el=`*integer*: Number of true 2-electron species – values: an integer
- `/distinct=`*yes-no*: If true (default) then all species have their own surface concentrations – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance

Multi-buffer version of the `adsorbed` fit.

sim-adsorbed - Simulation: Adsorbed species

`sim-adsorbed` *parameters datasets...* `/species=`*integer* `/2el=`*integer* `/distinct=`*yes-no* `/override=`*text* `/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *parameters*: File to load parameters from – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/species=`*integer*: Number of 1-electron species – values: an integer
- `/2el=`*integer*: Number of true 2-electron species – values: an integer
- `/distinct=`*yes-no*: If true (default) then all species have their own surface concentrations – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ','
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Simulation command for the `adsorbed` fit.

Differential equations fits

fit-ode - Fit: Fit an ODE system

`fit-ode` *system* `/with=`*time-dependent parameters* `/choose-t0=`*yes-no* `/adaptive=`*yes-no* `/step-size=`*number* `/min-step-size=`*number* `/prec-relative=`*number* `/prec-absolute=`*number* `/sub-steps=`*integer* `/stepper=`*choice* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* **(interactive)**

- *system*: Path to the file describing the ODE system – values: name of a file
- `/with=`*time-dependent parameters*: Make certain parameters depend upon time – values: several specifications of time dependent parameters (like `co:2,exp`), seperated by ';'. Available types: steps, exp, rexp
- `/choose-t0=`*yes-no*: If on, one can choose the initial time – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/adaptive=`*yes-no*: whether or not to use an adaptive stepper (on by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/step-size=`*number*: initial step size for the stepper – values: a floating-point number
- `/min-step-size=`*number*: minimum step size for the stepper – values: a floating-point number
- `/prec-relative=`*number*: relative precision required – values: a floating-point number
- `/prec-absolute=`*number*: absolute precision required – values: a floating-point number
- `/sub-steps=`*integer*: If this is not 0, then the smallest step size is that many times smaller than the minimum delta t – values: an integer
- `/stepper=`*choice*: algorithm used for integration (default: rkf45) – values: one of: bsimp, msadams, msbdf, rk1imp, rk2, rk2imp, rk4, rk4imp, rk8pd, rkck, rkf45
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','

Using this command, one can fit the results of integrating a system of differential equations (as with the command `ode`) to a dataset. The system is a file given as the *system* argument. For more details, please refer to the documentation of the `ode` command. The parameters whose values are not defined in the *system* file become the fit parameters. If there is no optional third section in the *system* file, the *y* value of the function is by default a linear combination of the variables of the system.

As with the `kinetic-system` fit, some of the parameters of the system can be varied automatically as a function of time, using the `/with=` option. See time dependent parameters below for more information.

mfit-ode - Multi fit: Fit an ODE system

`mfit-ode` *system datasets...* `/with=`*time-dependent parameters* `/choose-t0=`*yes-no* `/adaptive=`*yes-no* `/step-size=`*number* `/min-step-size=`*nu* `/prec-relative=`*number* `/prec-absolute=`*number* `/sub-steps=`*integer* `/stepper=`*choice* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *system*: Path to the file describing the ODE system – values: name of a file

- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/with=`*time-dependent parameters*: Make certain parameters depend upon time – values: several specifications of time dependent parameters (like `co:2,exp`), seperated by ';'. Available types: steps, exp, rexp
- `/choose-t0=`*yes-no*: If on, one can choose the initial time – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/adaptive=`*yes-no*: whether or not to use an adaptive stepper (on by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/step-size=`*number*: initial step size for the stepper – values: a floating-point number
- `/min-step-size=`*number*: minimum step size for the stepper – values: a floating-point number
- `/prec-relative=`*number*: relative precision required – values: a floating-point number
- `/prec-absolute=`*number*: absolute precision required – values: a floating-point number
- `/sub-steps=`*integer*: If this is not 0, then the smallest step size is that many times smaller than the minimum delta t – values: an integer
- `/stepper=`*choice*: algorithm used for integration (default: rkf45) – values: one of: bsimp, msadams, msbdf, rk1imp, rk2, rk2imp, rk4, rk4imp, rk8pd, rkck, rkf45
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Multibuffer version of the `ode` fit.

sim-ode - Simulation: Fit an ODE system

`sim-ode` *system parameters datasets...* `/with=`*time-dependent parameters* `/choose-t0=`*yes-no* `/adaptive=`*yes-no* `/step-size=`*number* `/min-step-size=`*number* `/prec-relative=`*number* `/prec-absolute=`*number* `/sub-steps=`*integer* `/stepper=`*choice* `/override=`*text* `/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *system*: Path to the file describing the ODE system – values: name of a file
- *parameters*: File to load parameters from – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/with=`*time-dependent parameters*: Make certain parameters depend upon time – values: several specifications of time dependent parameters (like `co:2,exp`), seperated by ';'. Available types: steps, exp, rexp
- `/choose-t0=`*yes-no*: If on, one can choose the initial time – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/adaptive=`*yes-no*: whether or not to use an adaptive stepper (on by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/step-size=`*number*: initial step size for the stepper – values: a floating-point number
- `/min-step-size=`*number*: minimum step size for the stepper – values: a floating-point number
- `/prec-relative=`*number*: relative precision required – values: a floating-point number
- `/prec-absolute=`*number*: absolute precision required – values: a floating-point number
- `/sub-steps=`*integer*: If this is not 0, then the smallest step size is that many times smaller than the minimum delta t – values: an integer
- `/stepper=`*choice*: algorithm used for integration (default: rkf45) – values: one of: bsimp, msadams, msbdf, rk1imp, rk2, rk2imp, rk4, rk4imp, rk8pd, rkck, rkf45
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ','
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Simulation command for the `ode` fit.

Kinetic systems

It is possible with QSoas to fit kinetic traces that follow the concentration of one or more species that are part of a full kinetic system. For that, you need to write a simple text file of the following form:

```
A <=>[k_i][k_a] I1
I1 ->[k_i2 * o2] I2
```

This describes a kinetic system with three species, `A`, `I1` and `I2`, with a reversible reaction from `A` to `I1` with a forward rate of `k_i` and a backward rate of `k_a`, and an irreversible reaction from `I1` to `I2` with a rate of `k_i2 * o2`.

The files can contain comment lines starting with a `#`, and can contain an arbitrary large number of reactions.

QSoas automatically detects the parameters from the fit, here `k_i`, `k_a`, `k_i2` and `o2`, and the initial concentrations of `A`, `I1` and `I2`, namely `c0_A`, `c0_I1` and `c0_I2`.

It is possible to assign special time dependence to any of the parameters by using the `/with` option:

```
QSoas> fit-kinetic-system /with=o2:3,exp kinetic-file.txt
```

This gives to `o2` the value of the sum of three exponential decays shifted in time (see formula below); this possibility is documented in greater detail below.

To define a new fit one could combine with others using `combine-fits`, use `define-kinetic-system-fit`.

fit-kinetic-system - Fit: Full kinetic system

`fit-kinetic-system` *system* `/with=`*time-dependent parameters* `/choose-t0=`*yes-no* `/adaptive=`*yes-no* `/step-size=`*number* `/min-step-size=`*n*
`/prec-relative=`*number* `/prec-absolute=`*number* `/sub-steps=`*integer* `/stepper=`*choice* `/extra-parameters=`*text* `/parameters=`*file*
`/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* **(interactive)**

- *system*: Path to the file describing the system – values: name of a file
- `/with=`*time-dependent parameters*: Make certain parameters depend upon time – values: several specifications of time dependent parameters (like `co:2,exp`), seperated by ';'. Available types: steps, exp, rexp
- `/choose-t0=`*yes-no*: If on, one can choose the initial time – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/adaptive=`*yes-no*: whether or not to use an adaptive stepper (on by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/step-size=`*number*: initial step size for the stepper – values: a floating-point number
- `/min-step-size=`*number*: minimum step size for the stepper – values: a floating-point number
- `/prec-relative=`*number*: relative precision required – values: a floating-point number
- `/prec-absolute=`*number*: absolute precision required – values: a floating-point number
- `/sub-steps=`*integer*: If this is not 0, then the smallest step size is that many times smaller than the minimum delta t – values: an integer
- `/stepper=`*choice*: algorithm used for integration (default: rkf45) – values: one of: bsimp, msadams, msbdf, rk1imp, rk2, rk2imp, rk4, rk4imp, rk8pd, rkck, rkf45
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','

Fits a full kinetic system. The fitted value is a linear combination of all the concentrations, with the coefficients given by parameters of name `y_A` (for the coefficient for the concentration of species `A`, for instance).

mfit-kinetic-system - Multi fit: Full kinetic system

`mfit-kinetic-system` *system datasets...* `/with=`*time-dependent parameters* `/choose-t0=`*yes-no* `/adaptive=`*yes-no* `/step-size=`*number*
`/min-step-size=`*number* `/prec-relative=`*number* `/prec-absolute=`*number* `/sub-steps=`*integer* `/stepper=`*choice* `/extra-parameters=`*text*
`/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *system*: Path to the file describing the system – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/with=`*time-dependent parameters*: Make certain parameters depend upon time – values: several specifications of time dependent parameters (like `co:2,exp`), seperated by ';'. Available types: steps, exp, rexp
- `/choose-t0=`*yes-no*: If on, one can choose the initial time – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/adaptive=`*yes-no*: whether or not to use an adaptive stepper (on by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

- `/step-size=`*number*: initial step size for the stepper – values: a floating-point number
- `/min-step-size=`*number*: minimum step size for the stepper – values: a floating-point number
- `/prec-relative=`*number*: relative precision required – values: a floating-point number
- `/prec-absolute=`*number*: absolute precision required – values: a floating-point number
- `/sub-steps=`*integer*: If this is not 0, then the smallest step size is that many times smaller than the minimum delta t – values: an integer
- `/stepper=`*choice*: algorithm used for integration (default: rkf45) – values: one of: bsimp, msadams, msbdf, rk1imp, rk2, rk2imp, rk4, rk4imp, rk8pd, rkck, rkf45
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Multi-buffer variant of the `kinetic-system` fit.

sim-kinetic-system - Simulation: Full kinetic system

`sim-kinetic-system` *system parameters datasets...* `/with=`*time-dependent parameters* `/choose-t0=`*yes-no* `/adaptive=`*yes-no* `/step-size=`*numb*
`/min-step-size=`*number* `/prec-relative=`*number* `/prec-absolute=`*number* `/sub-steps=`*integer* `/stepper=`*choice* `/override=`*text*
`/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *system*: Path to the file describing the system – values: name of a file
- *parameters*: File to load parameters from – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/with=`*time-dependent parameters*: Make certain parameters depend upon time – values: several specifications of time dependent parameters (like `co:2,exp`), seperated by ';'. Available types: steps, exp, rexp
- `/choose-t0=`*yes-no*: If on, one can choose the initial time – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/adaptive=`*yes-no*: whether or not to use an adaptive stepper (on by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/step-size=`*number*: initial step size for the stepper – values: a floating-point number
- `/min-step-size=`*number*: minimum step size for the stepper – values: a floating-point number
- `/prec-relative=`*number*: relative precision required – values: a floating-point number
- `/prec-absolute=`*number*: absolute precision required – values: a floating-point number
- `/sub-steps=`*integer*: If this is not 0, then the smallest step size is that many times smaller than the minimum delta t – values: an integer
- `/stepper=`*choice*: algorithm used for integration (default: rkf45) – values: one of: bsimp, msadams, msbdf, rk1imp, rk2, rk2imp, rk4, rk4imp, rk8pd, rkck, rkf45
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ','
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Computation for the `kinetic-system` fit.

define-kinetic-system-fit - Define a fit based on a kinetic mode

`define-kinetic-system-fit` *file name*

- *file*: System to load – values: name of a file
- *name*: Name of the newly created fit – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

In the `fit-kinetic-system` fit, one has to provide systematically the name of the file that contains the kinetic system. This can be annoying at times, and it makes it impossible to use these fits with `combine-fits` or `define-derived-fit`.

This command makes it possible to define a new fit for the kinetic system contained in *file*. The kinetic system is read once: further changes in the file will **not** be taken into account.

Time-dependent parameters

Some fits, namely `arb`, `ode` and `kinetic-system` (and all the custom fits defined using `custom-fit` or `define-kinetic-system-fit`) have a built-in possibility to have some parameters depend on time (instead of being constant). For instance, this possibility can be used in kinetic systems to impose an external dependence on various parameters: maybe a rate constant will only have a non-zero value during a given period of time, in the case of a light-dependent process, or another parameter is modulated sinusoidally...

The time-dependent parameters are defined using the `/with=` option to the fits. This option takes a `;`-separated list of specifications of the form: *parameter*:*number*,*type*,*options*... where *parameter* is the name of the parameter that will depend on time, *type* is the type of the dependence (see below), *number* (not always needed) is the number of "features" in the dependence (very type-dependent), and *options* can additionnally be used for some types.

`QSoas` recognizes the following time-dependences:

- `exp`, where the given parameter, p is given by:

$$p = \sum_{i=1}^{n} p_i \exp\left(-\frac{t - t_i^p}{\tau_i^p}\right) \times H(t - t_i^p)$$

where H is the heavyside step function (1 for positive argument, 0 else) and n is the number given just after `:` (in command below, that means you will have three different steps). You may wish to have all τ values common, which you do by adding `,common` in the spec:

```
QSoas> fit-kinetic-system /with=o2:3,exp,common kinetic-file.txt
```

This kind of functions were used to analyse the inhibition of NiFe hydrogenase by CO and O2, see for instance Liebgott *et al*, **Nat. Chem. Biol.**, 2010.

- `steps`, where the given parameter, takes a series of values (1 more than the *number* given) at the given times.
- `rexp`, that combines the `steps` and `exp`: the time is divided in *number* segments (preceded by an initial segment in which the parameter is fixed) in which the parameter is given by:

$$p = \left(p(t = t_i) - p_\infty^i\right) \exp -\frac{t - t_i}{\tau_i} \qquad \text{for} \qquad t_i < t \leq t_{i+1}$$

As for `exp`, the time constant can be chosen to be common to all the segments by adding `,common` after the spec.

Another way to look at the different types of time-dependent parameters available in your version of `QSoas` is to run the file `make-all.cmds` from the `tests/time-dependent-parameters` directory of the source code archive.

Slow scans fits

These specific fits were used in the context of the interpretation of cyclic voltammograms of adorbed nickel-iron hydrogenase that undergo inactivations under oxidizing conditions. For more information, refer to Abou-Hamdam *et al*, **PNAS** 2012 (which you probably should cite anyway if you use them).

fit-slow-scan-hp - Fit: Slow scan test

`fit-slow-scan-hp /bi-exp=`*yes-no* `/scaling=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* **(interactive)**

- `/bi-exp=`*yes-no*: Whether we have a bi-exponential or a mono-exponential – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/scaling=`*yes-no*: Use an additional scaling factor – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ','

Fit for the "high-potential" part of a slow voltammetric scan where inactivation occurs with rate constants that do not depend on time. The current for the active form is assumed to depend linearly on the potential.

Formula:

$$i_{forward} = (a + b \times E)\left[(A_1 - A_e) \times \exp\left\{-(EE_1)/(\nu\tau)\right\} + A_e\right]$$

$$i_{backward} = (a + b \times E)\left[(A_1 - A_e) \times \exp\left\{-(2E_v - E_1 - E)/(\nu\tau)\right\} + A_e\right]$$

where E_v is the vertex potential, E_1 is the initial potential, τ the rate constant of decrease, A_1 the amount of initially active enzyme, A_e the equilibrium concentration of active species and ν the scan rate.

fit-slow-scan-lp - Fit: Slow scan test

`fit-slow-scan-lp` `/explicit-rate=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*wo* **(interactive)**

- `/explicit-rate=`*yes-no*: Whether the scan rate is an explicit parameter of the fit – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’

Fit for the “low-potential” part of a slow voltammetric scan where the enzyme reactivates with a rate constant that depends exponentially on the potential:

$$k_a = k \exp(-\alpha f E)$$

The overall formula is:

$$i(E) = \underbrace{(A \times E + B)}_{\text{stationnary current}} \underbrace{\left(1 - (1 - y_1) \exp^{-\frac{k}{\alpha f \nu}(\exp^{\alpha f E} - \exp^{\alpha f E_1})}\right)}_{\text{active fraction}}$$

E_1 is the initial potential, ν the scan rate

mfit-slow-scan-hp - Multi fit: Slow scan test

`mfit-slow-scan-hp` *datasets...* `/bi-exp=`*yes-no* `/scaling=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/bi-exp=`*yes-no*: Whether we have a bi-exponential or a mono-exponential – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/scaling=`*yes-no*: Use an additional scaling factor – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance

Multi-buffer variant of the `fit-slow-scan-hp` fit.

mfit-slow-scan-lp - Multi fit: Slow scan test

`mfit-slow-scan-lp` *datasets...* `/explicit-rate=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/explicit-rate=`*yes-no*: Whether the scan rate is an explicit parameter of the fit – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’

- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Multi-buffer variant of the `fit-slow-scan-lp` fit.

sim-slow-scan-lp - Simulation: Slow scan test

`sim-slow-scan-lp` *parameters datasets...* `/explicit-rate=`*yes-no* `/override=`*text* `/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *parameters*: File to load parameters from – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/explicit-rate=`*yes-no*: Whether the scan rate is an explicit parameter of the fit – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ','
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Computation for the `slow-scan-lp` fit.

sim-slow-scan-hp - Simulation: Slow scan test

`sim-slow-scan-hp` *parameters datasets...* `/bi-exp=`*yes-no* `/scaling=`*yes-no* `/override=`*text* `/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *parameters*: File to load parameters from – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/bi-exp=`*yes-no*: Whether we have a bi-exponential or a mono-exponential – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/scaling=`*yes-no*: Use an additional scaling factor – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ','
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Computation for the `slow-scan-hp` fit.

Wave shape fits

These fits correspond to the various models described in Fourmond *et al*, **JACS** 2013:

- `fit-eecr-wave` is the *EEC* model
- `fit-ecr-wave` is the *EC* model
- `fit-eecr-relay-wave` is the *EECR* model

Please refer to that paper for more information.

fit-eecr-wave - Fit: Fit of an EECR catalytic wave

`fit-eecr-wave` `/plateau=`*yes-no* `/oxidation=`*yes-no* `/use-eoc=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* **(interactive)**

- `/plateau=`*yes-no*: Whether to use the general expression or only that valid when plateaus are not reached – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/oxidation=`*yes-no*: If on, use the oxidation current as reference – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/use-eoc=`*yes-no*: Whether to use explicitly the bias or compute it using the open circuit potential – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’

Fits the so-called EEC model in Fourmond *et al*, **JACS** 2013 to the data.
If `/plateau` is off (default), then the equation used is that for the limit where the dispersion of values of k_0 large $\beta d_0 \to \infty$, while the full expression is used in the other case.
If `/oxidation` is true, the reference current is the oxidation current (and not the reduction current as is the default).
If `/use-eoc` is true, then the open circuit potential is used as a parameter rather than the ratio k_2/k_{-2}.

mfit-eecr-wave - Multi fit: Fit of an EECR catalytic wave

`mfit-eecr-wave` *datasets...* `/plateau=`*yes-no* `/oxidation=`*yes-no* `/use-eoc=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/plateau=`*yes-no*: Whether to use the general expression or only that valid when plateaus are not reached – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/oxidation=`*yes-no*: If on, use the oxidation current as reference – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/use-eoc=`*yes-no*: Whether to use explicitly the bias or compute it using the open circuit potential – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance

Multi-buffer variant of `fit-eecr-wave`.

sim-eecr-wave - Simulation: Fit of an EECR catalytic wave

`sim-eecr-wave` *parameters datasets...* `/plateau=`*yes-no* `/oxidation=`*yes-no* `/use-eoc=`*yes-no* `/override=`*text* `/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *parameters*: File to load parameters from – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/plateau=`*yes-no*: Whether to use the general expression or only that valid when plateaus are not reached – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/oxidation=`*yes-no*: If on, use the oxidation current as reference – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/use-eoc=`*yes-no*: Whether to use explicitly the bias or compute it using the open circuit potential – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ‘,’
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Simulation for the `eecr-wave` fit.

fit-eecr-relay-wave - Fit: Fit of an EECR+relay catalytic wave

`fit-eecr-relay-wave` `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* **(interactive)**

- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’

Fits the so-called EEC with relay model in Fourmond *et al*, **JACS** 2013 to the data.

mfit-eecr-relay-wave - Multi fit: Fit of an EECR+relay catalytic wave

`mfit-eecr-relay-wave` *datasets...* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance

Multi-buffer version of the `eecr-relay-wave` fit.

sim-eecr-relay-wave - Simulation: Fit of an EECR+relay catalytic wave

`sim-eecr-relay-wave` *parameters datasets...* `/override=`*text* `/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *parameters*: File to load parameters from – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ‘,’
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Simulation command for the `eecr-relay-wave` fit.

fit-ecr-wave - Fit: Fit of an ECR catalytic wave

`fit-ecr-wave` `/plateau=`*yes-no* `/oxidation=`*yes-no* `/use-eoc=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* **(interactive)**

- `/plateau=`*yes-no*: Whether to use the general expression or only that valid when plateaus are not reached – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/oxidation=`*yes-no*: If on, use the oxidation current as reference – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/use-eoc=`*yes-no*: Whether to use explicitly the bias or compute it using the open circuit potential – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’

Fits the so-called EC model in Fourmond *et al*, **JACS** 2013 to the data.
See the `eecr-wave` fit for a description of the `/oxidation`, `/plateau` and `/use-eoc` options.

mfit-ecr-wave - Multi fit: Fit of an ECR catalytic wave

`mfit-ecr-wave` *datasets...* `/plateau=`*yes-no* `/oxidation=`*yes-no* `/use-eoc=`*yes-no* `/extra-parameters=`*text* `/parameters=`*file* `/debug=`*yes-no* `/debug2=`*yes-no* `/set-from-meta=`*words* `/weight-buffers=`*yes-no* `/perp-meta=`*text* **(interactive)**

- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/plateau=`*yes-no*: Whether to use the general expression or only that valid when plateaus are not reached – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/oxidation=`*yes-no*: If on, use the oxidation current as reference – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/use-eoc=`*yes-no*: Whether to use explicitly the bias or compute it using the open circuit potential – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/parameters=`*file*: Pre-loads parameters – values: name of a file
- `/debug=`*yes-no*: turn on debugging (for QSoas developers only, default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/debug2=`*yes-no*: extremely verbose debugging (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/set-from-meta=`*words*: sets parameter values from meta-data – values: several words, separated by ‘,’
- `/weight-buffers=`*yes-no*: whether or not to weight buffers (off by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/perp-meta=`*text*: if specified, it is the name of a meta-data that holds the perpendicular coordinates – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance

Multi-buffer version of the `ecr-wave` fit.

sim-ecr-wave - Simulation: Fit of an ECR catalytic wave

`sim-ecr-wave` *parameters datasets...* `/plateau=`*yes-no* `/oxidation=`*yes-no* `/use-eoc=`*yes-no* `/override=`*text* `/extra-parameters=`*text* `/flags=`*words* `/reexport=`*yes-no*

- *parameters*: File to load parameters from – values: name of a file
- *datasets...*: the buffers whose X values will be used for simulations – values: comma-separated lists of buffers in the stack, see buffers lists
- `/plateau=`*yes-no*: Whether to use the general expression or only that valid when plateaus are not reached – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/oxidation=`*yes-no*: If on, use the oxidation current as reference – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/use-eoc=`*yes-no*: Whether to use explicitly the bias or compute it using the open circuit potential – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/override=`*text*: A comma-separated list of parameters to override – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/extra-parameters=`*text*: Define supplementary parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ‘,’
- `/reexport=`*yes-no*: Do not compute data, just re-export fit parameters and errors – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Simulation command for the `ecr-wave` fit.

Computation/simulations functions

The commands in this section generate data “from scracth”, though most require a buffer to work on, to act as a model for the X values. You can create a buffer for those commands using `generate-buffer`.

Evaluation functions

While it is not its primary purpose, QSoas provides various functions to evaluate the result of mathematical operations.

eval - Ruby eval

`eval` *code* `/use-dataset=`*yes-no*

- *code*: Any ruby code – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/use-dataset=`*yes-no*: If on (the default) and if there is a current dataset, the $meta and $stats hashes are available – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Evaluates the given code as a Ruby expression:

```
QSoas> eval 2*3
 => 6
```

It runs in the same environment as the `apply-formula` and the custom fits (excepted, of course, that there are no `x` and `y` variables). It can be useful to check that a function has been correctly defined in a file loaded by `ruby-run`. You can also use that as a calculator.

find-root - Finds a root

`find-root` *formula seed* `/max=`*number*

- *formula*: An expression of 1 variable (not an equation !) – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- *seed*: Initial X value from which to search – values: a floating-point number
- `/max=`*number* (default option): If present, uses dichotomy between seed and max – values: a floating-point number

Find the root of the given `x`-dependent expression using an iterative algorithm, using *seed* as the initial value. If the `/max` option is specified, then the search proceeds using dichotomy between the two values (*seed* and *max*).

```
QSoas> find-root 'x**2 - 3' 1
Found root at: 1.73205
```

Do not use a equal sign. The returned value is that for which the expression equates 0.

integrate-formula - Integrate expression

`integrate-formula` *formula a b* `/integrator=`*choice* `/prec-relative=`*number* `/prec-absolute=`*number*

- *formula*: An expression of 1 variable (not an equation !) – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- *a*: Left bound of the segment – values: a floating-point number
- *b*: Right bound of the segment – values: a floating-point number
- `/integrator=`*choice*: The algorithm used for integration – values: one of: gauss15, gauss21, gauss31, gauss41, gauss51, gauss61, qng
- `/prec-relative=`*number*: Relative precision required for integration – values: a floating-point number
- `/prec-absolute=`*number*: Absolute precision required for integration – values: a floating-point number

Computes the integral of the given expression of `x` between bounds *a* and *b*:

```
QSoas> integrate-formula x**2 10 22
Integral value: 3216 estimated error: 3.57048e-11  in 31 evaluations over 1 intervals
```

generate-buffer - Generate buffer

`generate-buffer` *start end* `/samples=`*integer* `/number=`*integer* `/formula=`*text* `/flags=`*words*

- *start*: The first X value – values: a floating-point number
- *end*: The last X value – values: a floating-point number
- `/samples=`*integer*: number of data points – values: an integer
- `/number=`*integer*: generates that many datasets – values: an integer
- `/formula=`*text* (default option): Formula to generate the Y values – values: arbitrary text. If you need spaces, do not forget to quote them with ‘, for instance
- `/flags=`*words*: Flags to set on the results – values: several words, separated by ‘,’

Generates a buffer with *samples* samples (by default 1000) uniformly spaced between *start* and *end*.
If formula is provided, it sets Y values according to this formula (else Y is take equal to X).

```
QSoas> generate-buffer -10 10 sin(x)
```

Simulation functions

kinetic-system - Kinetic system evolver

`kinetic-system` *reaction-file parameters* `/dump=`*yes-no* `/adaptive=`*yes-no* `/step-size=`*number* `/min-step-size=`*number* `/prec-relative=`*nu*
`/prec-absolute=`*number* `/sub-steps=`*integer* `/stepper=`*choice* `/annotate=`*yes-no*

- *reaction-file*: File describing the kinetic system – values: name of a file
- *parameters*: Parameters of the model – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/dump=`*yes-no*: if on, prints a description of the system rather than solving (default: false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/adaptive=`*yes-no*: whether or not to use an adaptive stepper (on by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/step-size=`*number*: initial step size for the stepper – values: a floating-point number
- `/min-step-size=`*number*: minimum step size for the stepper – values: a floating-point number
- `/prec-relative=`*number*: relative precision required – values: a floating-point number
- `/prec-absolute=`*number*: absolute precision required – values: a floating-point number
- `/sub-steps=`*integer*: If this is not 0, then the smallest step size is that many times smaller than the minimum delta t – values: an integer
- `/stepper=`*choice*: algorithm used for integration (default: rkf45) – values: one of: bsimp, msadams, msbdf, rk1imp, rk2, rk2imp, rk4, rk4imp, rk8pd, rkck, rkf45
- `/annotate=`*yes-no*: If on, a last column will contain the number of function evaluation for each step (default false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Simulates the evolution over time of the kinetic system given in the *reaction-file* (see the section kinetic system for the syntax of the reaction files).

This commands will use the current buffer as a source for X values.

The result is a multi-column buffer containing the concentration of all the species in the different columns.

parameters is a list of assignments evaluated at the beginning of the time evolution to set the parameters of the system. (all parameters not set this way default to 0). This list is evaluated as Ruby code, so you should separate the assignments with `;`.

For instance, if the reaction file (`system.sys`) contains:

```
A <=>[ki][ka] I
```

You can run the following commands to simulate the time evolution of the system with initial concentration of A equal to 1 (the parameter `c0_A`), of I equal to 0 (the parameter `c0_I`, here not specified so assumed to be 0) and with `ki` and `ka` equal to 1:

```
QSoas> generate-buffer 0 10
QSoas> kinetic-system system.sys 'c0_A = 1;ka = 1; ki = 1'
```

The *parameters* argument is actually a Ruby expression that is evaluated once before the computations start, so you can use more complex formulas.

ode - ODE solver

`ode` *file* `/parameters=`*text* `/annotate=`*yes-no* `/dump=`*yes-no* `/adaptive=`*yes-no* `/step-size=`*number* `/min-step-size=`*number*
`/prec-relative=`*number* `/prec-absolute=`*number* `/sub-steps=`*integer* `/stepper=`*choice*

- *file*: File containing the system – values: name of a file
- `/parameters=`*text* (default option): Values of the parameters – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance
- `/annotate=`*yes-no*: If on, a last column will contain the number of function evaluation for each step – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/dump=`*yes-no*: If on, do not integrate, just dumps the parse contents of the ODE file – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/adaptive=`*yes-no*: whether or not to use an adaptive stepper (on by default) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/step-size=`*number*: initial step size for the stepper – values: a floating-point number
- `/min-step-size=`*number*: minimum step size for the stepper – values: a floating-point number
- `/prec-relative=`*number*: relative precision required – values: a floating-point number
- `/prec-absolute=`*number*: absolute precision required – values: a floating-point number
- `/sub-steps=`*integer*: If this is not 0, then the smallest step size is that many times smaller than the minimum delta t – values: an integer
- `/stepper=`*choice*: algorithm used for integration (default: rkf45) – values: one of: bsimp, msadams, msbdf, rk1imp, rk2, rk2imp, rk4, rk4imp, rk8pd, rkck, rkf45

`ode` solves ordinary differential equations. The equation definition file is structure in three parts, separated by at least one fully blank line, the last one being optional.

The first section defines the "initial conditions"; there are as many integrated variables as there are lines in this section. This section is only evaluated at the beginning of the system.

The second section defines the derivatives; they are evaluated several times for each time step.

Here is the contents of the file (say `sine.ode`) one would use to obtain $\sin t$ and $\cos t$ as solutions.

```
sin = 0
cos = 1

d_sin = cos
d_cos = -sin
```

After running the commands

```
QSoas> generate-buffer 0 10
QSoas> ode sine.ode
```

One has a buffer with one X column (representing the t values), and two Y columns, $\sin t$ and $\cos t$ (in the order in which they are given in the "initial conditions" section).

The optional third section can be used to control the exact output of the program. The above example can be completed thus:

```
sin = 0
cos = 1

d_sin = cos
d_cos = -sin

[sin, cos, sin**2 + cos**2]
```

Using this gives 3 Y columns: $\sin t$, $\cos t$ and $\sin^2 t + \cos^2 t$, that should hopefully be very close to 1.

Details of the integrations procedures can be tweaked using the parameters:

- `/stepper`: the ODE stepper algorithm. You can find more about them in the GSL documentation
- `/prec-relative` and `/prec-absolute` control the precision. A step will be deemed precise enough if the error estimate is **either** smaller than the relative precision or than the absolute precision
- `/adaptive` controls whether an adaptative step size is used (the values of t in the resulting buffer are always those asked, but there may be more intermediate steps). You should seldom need to turn it off

If `/annotate` is on, a last column is added that contains the number of the evaluations of derivatives for each step (useful for understanding why an integration takes so long, for instance).

The system of equations may contain undefined variables; one could have for instance used:

```
d_sin = omega * cos
d_cos = -omega * sin
```

Their values are set to 0 by default. You can change their values using the `/parameters` option:

```
QSoas> ode sine.ode /parameters="omega = 3"
```

Scripting facilities

QSoas provides facilities for scripting, ie running commands unattended, for instance for preparing series of data files for fitting or further use. There are a few commands useful only in this context, such as the following

Scripting commands

run - Run commands

`run` *file...* `/silent=`*yes-no* `/add-to-history=`*yes-no* `/cd-to-script=`*yes-no*

Short name: `@`

- *file...*: First is the command files, following are arguments – values: one or more files. Can include wildcards such as *, `[0-4]`, etc...
- `/silent=`*yes-no*: Whether or not display is updated during the load – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/add-to-history=`*yes-no*: Whether the commands run are added to the history (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

- `/cd-to-script=`*yes-no*: If on, automatically change the directory to that oof the script – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Run commands saved in a file. If a compulsory argument is missing, QSoas will prompt the user.

Arguments following the name of the script are passed to the script as "special variables" `${1}`, and `${2}` etc.

Imagine you are often doing the same processing a given type of data files, say, simply filtering them. You just have to write a script `process.cmd` containing:

```
load ${1}
auto-filter-fft
```

And run it this way:

```
QSoas> @ process.cmd data_file.dat
```

If you start to use `run` regularly, you may be interested in the other scripting commands, such as `run-for-each`, `run-for-datasets` and `startup-files`

startup-files - Startup files

`startup-files /add=`*file* `/rm=`*integer* `/run=`*yes-no*

- `/add=`*file* (default option): Adds the given startup file – values: name of a file
- `/rm=`*integer*: Removes the numbered file – values: an integer
- `/run=`*yes-no*: If on, runs all the startup files right now – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Using this command, you can instruct `QSoas` to execute command files at startup. Without options, this command displays the list of command files that `QSoas` will read at the next startup.

Files given to the `/add` options are added at the end of the list.

To remove a fil from the list, obtain its number by running startup-fileswithout any option, then use startup-files again with the option /rm=.

You can re-run all startup files by running:

```
QSoas> startup-files /run=true
```

run-for-each - Runs a script for several arguments

`run-for-each` *script arguments...* `/arg2=`*file* `/arg3=`*file* `/arg4=`*file* `/arg5=`*file* `/arg6=`*file* `/range-type=`*choice* `/silent=`*yes-no* `/add-to-history=`*yes-no*

- *script*: The script file – values: name of a file
- *arguments...*: All the arguments for the script file to loop on – values: one or more files. Can include wildcards such as `*`, `[0-4]`, etc...
- `/arg2=`*file*: Second argument to the script – values: name of a file
- `/arg3=`*file*: Third argument to the script – values: name of a file
- `/arg4=`*file*: Fourth argument to the script – values: name of a file
- `/arg5=`*file*: Fifth argument to the script – values: name of a file
- `/arg6=`*file*: Sixth argument to the script – values: name of a file
- `/range-type=`*choice*: If on, transform arguments into ranged numbers – values: one of: `lin`, `log`
- `/silent=`*yes-no*: Whether or not display is updated during the load – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/add-to-history=`*yes-no*: Whether the commands run are added to the history (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`

Runs the given script file successively for each argument given. For instance, running:

```
QSoas> run-for-each process-my-file.cmds file1 file2 file3
```

Is equivalent to running successively

```
QSoas> @ process-my-file.cmds file1
QSoas> @ process-my-file.cmds file2
QSoas> @ process-my-file.cmds file3
```

The arguments may not be file names, although automatic completion will only complete file names. If the script you want to run requires more than one argument, you can specify them (for all the runs) using the options `/arg2`, `/arg3` and so on:

```
QSoas> run-for-each process-my-file.cmds /arg2=other file1 file2
```

Is equivalent to running:

```
QSoas> @ process-my-file.cmds file1 other
QSoas> @ process-my-file.cmds file2 other
```

If you specify either `/range-type=lin` or `/range-type=log`, the parameters are interpreted differently, and are expected to be of the type `1..10:20`, which means 20 numbers between 1 and 10 (inclusive), that are spaced either linearly or logarithmically, depending on the value of the option.

run-for-datasets - Runs a script for several datasets

`run-for-datasets` *script datasets...* `/silent=`*yes-no* `/add-to-history=`*yes-no* `/arg1=`*file* `/arg2=`*file* `/arg3=`*file* `/arg4=`*file* `/arg5=`*file* `/arg6=`*file*

- *script*: The script file – values: name of a file
- *datasets...*: All the arguments for the script file to loop on – values: comma-separated lists of buffers in the stack, see buffers lists
- `/silent=`*yes-no*: Whether or not display is updated during the load – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/add-to-history=`*yes-no*: Whether the commands run are added to the history (defaults to false) – values: a boolean: `yes`, `on`, `true` or `no`, `off`, `false`
- `/arg1=`*file*: First argument to the script – values: name of a file
- `/arg2=`*file*: Second argument to the script – values: name of a file
- `/arg3=`*file*: Third argument to the script – values: name of a file
- `/arg4=`*file*: Fourth argument to the script – values: name of a file
- `/arg5=`*file*: Fifth argument to the script – values: name of a file
- `/arg6=`*file*: Sixth argument to the script – values: name of a file

Runs the given script file for each of the datasets given. Before each invocation of the script, the dataset is pushed back to the top of the stack, as if by `fetch`.

noop - No op

`noop` *ignored...* `/*=`*text*

- *ignored...*: Ignored arguments – values: several words, separated by ''
- `/*=`*text* (default option): Ignored options – values: arbitrary text. If you need spaces, do not forget to quote them with ', for instance

Does not do anything, but slurps all arguments and options given.

Non-interactive commands

In addition to purely scripting commands, many commands do not require user interaction, provided all their arguments are given. They are listed here:

- `1`
- `2`
- `apply-formula`
- `auto-correlation`
- `auto-filter-bs`
- `auto-filter-fft`
- `auto-flag`
- `auto-reglin`
- `average`
- `bin`
- `break`
- `browse`
- `cat`
- `cd`
- `chop`
- `clear`
- `clear-stack`
- `combine-fits`
- `commands`
- `comment`
- `contract`
- `credits`
- `custom-fit`
- `dataset-options`
- `define-alias`
- `define-derived-fit`
- `define-distribution-fit`
- `define-kinetic-system-fit`

- diff
- diff2
- display-aliases
- div
- downsample
- drop
- dx
- dy
- echem-peaks
- edit
- eval
- expand
- fetch
- find-peaks
- find-root
- find-steps
- flag
- generate-buffer
- graphics-settings
- help
- hide-buffer
- integrate
- integrate-formula
- interpolate
- kinetic-system
- limits
- load
- load-as-chi-txt
- load-as-csv
- load-as-text
- load-fits
- load-stack
- merge
- noop
- norm
- ode
- output
- overlay
- overlay-buffer
- points
- print
- pwd
- quit
- redo
- remove-spikes
- rename
- reverse
- ruby-run
- run
- run-for-datasets
- run-for-each
- save
- save-buffers
- save-history
- save-output
- save-stack
- segments-chop
- set-meta

- set-perp
- shiftx
- show
- show-stack
- sim-adsorbed
- sim-ecr-wave
- sim-eecr-relay-wave
- sim-eecr-wave
- sim-exponential-decay
- sim-gaussian
- sim-kinetic-system
- sim-lorentzian
- sim-multiexp-multistep
- sim-nernst
- sim-ode
- sim-slow-scan-hp
- sim-slow-scan-lp
- sort
- split-monotonic
- splita
- splitb
- startup-files
- stats
- strip-if
- subtract
- system
- temperature
- timer
- transpose
- tweak-columns
- undo
- unflag
- unwrap
- zero

Ruby code

QSoas internally uses Ruby for the interpretation of all formulas. This means in particular that all formulas must be valid ruby code.

Basically, the Ruby syntax ressembles that of other symbolic evaluation programs (it is quite close to the one from gnuplot), with the following restrictions:

- Parameter names **cannot start with an uppercase letter**, as those have a special meaning to the Ruby interpreter: anything that starts with an uppercase letter is assumed to be a constant.
- Don't abbreviate floating point numbers: `2.` and `.4` are invalid, use `2.0` and `0.4` instead.
- **Case matters**: `Pi` is π, while `pi` is nothing defined.

In addition to standard mathematical functions from the Math module (that contains, among others, the error function `erf` and the `gamma` function), the following special functions are available:

- `airy_ai`: Airy Ai function (three modes) $AiryAi(x)$
- `airy_ai_deriv`: First derivative of Airy Ai function (three modes) $\mathrm{d}AiryAi(x)/\mathrm{d}x$
- `airy_bi`: Airy Bi function (three modes) $AiryBi(x)$
- `airy_bi_deriv`: First derivative of Airy Bi function (three modes) $\mathrm{d}AiryBi(x)/\mathrm{d}x$
- `atanc`: $\frac{\tan^{-1} x}{x}$
- `atanhc`: $\frac{\tanh^{-1} x}{x}$
- `bessel_j0`: Regular cylindrical Bessel function of 0th order, $J_0(x)$
- `bessel_j1`: Regular cylindrical Bessel function of first order, $J_1(x)$
- `bessel_jn`: Regular cylindrical Bessel function of n-th order, $J_n(x)$
- `bessel_y0`: Irregular cylindrical Bessel function of 0th order, $Y_0(x)$
- `bessel_y1`: Irregular cylindrical Bessel function of first order, $Y_1(x)$

- `bessel_yn`: Irregular cylindrical Bessel function of n-th order, $Y_n(x)$
- `dawson`: Dawson integral, $\exp(-x^2)\int_0^x \exp(t^2)\mathrm{d}t$
- `debye_1`: Debye function of order 1, $D_1 = (1/x)\int_0^x \mathrm{d}t(t/(e^t-1))$
- `debye_2`: Debye function of order 2, $D_2 = (2/x^2)\int_0^x \mathrm{d}t(t^2/(e^t-1))$
- `debye_3`: Debye function of order 3, $D_3 = (3/x^3)\int_0^x \mathrm{d}t(t^3/(e^t-1))$
- `debye_4`: Debye function of order 4, $D_4 = (4/x^4)\int_0^x \mathrm{d}t(t^4/(e^t-1))$
- `debye_5`: Debye function of order 5, $D_5 = (5/x^5)\int_0^x \mathrm{d}t(t^5/(e^t-1))$
- `debye_6`: Debye function of order 6, $D_6 = (6/x^6)\int_0^x \mathrm{d}t(t^6/(e^t-1))$
- `dilog`: The dilogarithm, $Li_2(x) = -\Re\left(\int_0^x \mathrm{d}s \log(1-s)/s\right)$
- `expint_e1`: Exponential integral $E_1(x) = \int_x^\infty \frac{\exp -t}{t}\mathrm{d}t$
- `expint_e2`: Exponential integral $E_2(x) = \int_x^\infty \frac{\exp -t}{t^2}\mathrm{d}t$
- `expint_en`: Exponential integral $E_n(x) = \int_x^\infty \frac{\exp -t}{t^n}\mathrm{d}t$
- `lambert_W`: Principal branch of the Lambert function $W_0(x)$
- `lambert_Wm1`: Secondary branch of the Lambert function $W_{-1}(x)$

Some physical/mathematical constants are available; their name starts with an **uppercase** letter.

- `Alpha`: The fine structure constant, α – 0.00729735
- `C`: The speed of light in vacuum, c – 2.99792e+08
- `Eps_0`: The permeability of vacuum, ϵ_0 – 1.25664e-06
- `F`: Faraday's constant, F – 96485.3
- `H`: The Planck constant, h – 6.62607e-34
- `Hbar`: $\hbar = h/2\pi$ – 1.05457e-34
- `Kb`: Boltzmann's constant – 1.38065e-23
- `M_e`: The mass of the electron, m_e – 9.10938e-31
- `M_mu`: The mass of the mu, m_μ – 1.88353e-28
- `M_n`: The mass of the neutron, m_n – 1.67493e-27
- `M_p`: The mass of the proton, m_p – 1.67262e-27
- `Mu_0`: The permittivity of vacuum, μ_0 – 8.85419e-12
- `Mu_B`: The Bohr Magneton, μ_B – 9.27401e-24
- `Na`: The Avogadro number, N_A – 6.02214e+23
- `Pi`, `PI`: π – 3.14159
- `Q_e`: The absolute value of the charge of the electron, e – 1.60218e-19
- `R`: Molar gas constant, R – 8.31447
- `Ry`: The Rydberg constant, Ry – 2.17987e-18
- `Sigma`: The Stefan-Boltzmann radiation constant – 5.6704e-08