Tip for exporting Maxima results to LaTeX

Leo Butler

February 14, 2023

Contents

1 Goal

Generate LaTeX code from Maxima code.

2 Setup

2.1 maxima-init.lisp

The command `org-babel-execute:maxima` in `lisp/ob-maxima.el` uses the Maxima command `batchload` to execute Maxima code. This is a very tight-lipped loader, so we over-write `batchload` with `batch`. We also `load` an init file:

```
#+name: maxima-init.lisp
#+begin_src maxima :tangle maxima-init.lisp :exports none
  (defun $batchload (file) (mfuncall '$batch file))
  ($load "./maxima-init.mac")
#+end_src
```

On tangling, this produces the `common-lisp` output file `maxima-init.lisp`. It will be pre-loaded into Maxima.

2.2 maxima-init.mac

Next, we need to create an init file for Maxima that will provide an output printer that produces LaTeX output. One option would be to use the `imaxima` printer. Here is another option that uses the `alt-display` package. The code replaces the default printer with `org_tex_display`. It also sets the `epilog` prompt, so that the final `#+begin_example` is terminated.

```
#+name: maxima-init.mac
#+begin_src maxima :tangle maxima-init.mac :exports none :eval none
load("alt-display.mac") $
set_prompt('epilog,printf(false,"~%#+end_example")) $
define_alt_display(org_tex_display(x),
  block([],
    printf(true,"#+end_example~%#+begin_export latex~%"),
    printf(true,"\\textcolor{blue}{(\\~a~d)} ",outchar,linenum-1),
    tex(second(x)),
    printf(true,"~&#+end_export~%#+begin_example~%(input) "))) $
set_alt_display(2,org_tex_display) $
display2d:true $
printf(true,"#+begin_example~%(input) ") $
linenum : 0 $
#+end_src
```

3 An example

Here is an example that computes the partial derivatives of a composite function.

3.1 Org code

```
#+name: chain-rule
#+begin_src maxima :exports both :cmdline -p ./maxima-init.lisp
  (gradef(f(u,v),f_1(u,v),f_2(u,v)), 'done);
  diff(f(x^2-y^2,x*y),x);
  diff(f(x^2-y^2,x*y),y);
#+end_src
```

3.2 Maxima code

```
(gradef(f(u,v),f_1(u,v),f_2(u,v)), 'done);
diff(f(x^2-y^2,x*y),x);
diff(f(x^2-y^2,x*y),y);
```

The first line defines the partial derivatives of $f(u, v)$ with repect to u and v. The second and third lines compute the partial derivatives of the composite $f(x^2 - y^2, xy)$.

3.3 Result of evaluation; LaTeX output

The `batch` printer echos each input line; it prints the output of each command line that ends in a semicolon (`;`). The result of a line ending in a dollar sign (`$`) is not printed. The `org_tex_display` printer wraps each echoed input line in an `example` block and prints the output as it would appear in an `imaxima` session.

```
(input)
read and interpret /tmp/babel-HzHnIr/maxima-GLLEO6.max
(gradef(f(u,v),f_1(u,v),f_2(u,v)),'done)
```

(%o1)

done

```
(input)
diff(f(x^2-y^2,x*y),x)
```

(%o2)

$$y\,f_2\left(x^2-y^2,x\,y\right)+2\,x\,f_1\left(x^2-y^2,x\,y\right)$$

```
(input)
diff(f(x^2-y^2,x*y),y)
```

(%o3)

$$x\,f_2\left(x^2-y^2,x\,y\right)-2\,y\,f_1\left(x^2-y^2,x\,y\right)$$

```
(input)
gnuplot_close()
```

3.4 Two annoyances

The initial line `read and interpret...` and that final, dangling input line with `gnuplot_close()` are nuisances. They can be easily suppressed, but that requires patching `ob-maxima.el`. That's another story.

4 Epilogue

After sending my initial draft to the `Org` mailing list, I received some feedback.

4.1 Eric Fraga's suggestion

On the mailing list, Eric Fraga suggested adding the prologue/epilogue:

```
#+PROPERTY: header-args:maxima :prologue "fpprintprec: 2; linel: 50;"
#+PROPERTY: header-args:maxima :epilogue "for j: 1 thru length(solution) do (\
            print(\"\"), print(\"Solution\", j), \
            print(\"\"), for i: 1 thru length(solution[j]) do grind(solution[j][i]))$"
```

Let's redo the example above with those settings and incorporating Eric's design that the results need to be collected in the `solution` list:

```
#+name: chain-rule-redo
#+header: :prologue "fpprintprec: 2; linel: 50;"
#+header: :epilogue "for j: 1 thru length(solution) do (print(\"\"), print(\"Solution\", j), prin
#+begin_src maxima :exports results :results table
(gradef(f(u,v),f_1(u,v),f_2(u,v)), 'done);
fx:diff(f(x^2-y^2,x*y),x);
fy:diff(f(x^2-y^2,x*y),y);
solution:[fx,fy];
#+end_src
```

Here is the result:

Solution 1

$(y*f_2(x^2-y^{2,x}*y)+2*x*f_1(x^2-y^{2,x}*y))[1]$
$(y*f_2(x^2-y^{2,x}*y)+2*x*f_1(x^2-y^{2,x}*y))[2]$

Solution 2

$(x*f_2(x^2-y^{2,x}*y)-2*y*f_1(x^2-y^{2,x}*y))[1]$
$(x*f_2(x^2-y^{2,x}*y)-2*y*f_1(x^2-y^{2,x}*y))[2]$

4.2 Max Nikulin's idea

Max Nikulin remarked that the `org` code in this file contains redundancy. He suggested trying:

```
#+begin_src elisp :exports results :results silent
  (require 'ob-org)
#+end_src

#+name: elisp-in-org
#+begin_src org :exports both :results replace
  ,#+name: elisp-block
  ,#+begin_src elisp :exports results
    '((1 2 3) (4 5 6))
  ,#+end_src
#+end_src
```

The block named `elisp-in-org` is exported to:

```
#+name: elisp-block
#+begin_src elisp :exports results
  '((1 2 3) (4 5 6))
#+end_src
```

The `elisp` code block exports to a named block of `elisp` code.
That exports to:

1	2	3
4	5	6

4.2.1 A bug // Feature request

`Org` does a good job of recursively evaluating code blocks, as can be seen by the examples here (Thanks, Max!). But, two things are not done quite correctly:

1. When editing `org` code blocks in an indirect buffer, it should be possible to recursively edit a code block. That does not appear to work at the moment.

2. Noweb expansion does not work correctly in combination with recursive evaluation of code blocks. In the `org` block named `fraga-example.org`, a noweb reference `<<chain-rule>>` to the Maxima code block `chain-rule` is not expanded correctly (it leaves a blank line).

5 How to reproduce the pdf

1. In this `org` file, do `C-c C-v t` to tangle the two code blocks.
2. Then, do `C-c C-e l p` to export to pdf. Each time you are prompted about evaluating a code block, answer `y` or `yes`.