~~Postgkeyll~~ Postgkyl (?) Tutorial

Gkeyll Tutorial Invitational

Petr

November 2, 2022

COLLEGE OF ENGINEERING
KEVIN T. CROFTON DEPARTMENT OF
AEROSPACE AND OCEAN ENGINEERING
VIRGINIA TECH.

Gkeyll

Postprocessing is important!

Postprocessing is an often overlooked part of computation science but it is important!

A good postprocessing tool should...

1. be possible to install
2. make simple things easy
3. make complex things possible
4. be usable remotely on supercomputers

Credit for `Postgkyl` goes to Ammar, Noah, Jimmy, Mana, and recently Jason!

Postprocessing is important!

Postprocessing is an often overlooked part of computation science but it is important!

A good postprocessing tool should...

1. be possible to install
2. make simple things easy
3. make complex things possible
4. be usable remotely on supercomputers

Credit for `Postgkyl` goes to Ammar, Noah, Jimmy, Mana, and recently Jason!

`Postgkyl` is complex

The goal of this tutorial is to show key concepts, basic commands, and new features, so you know what to ask about.

Installation is mostly painless

Installation procedure did slightly change due to ADIOS 1.13 issues with the latest versions of NumPy. This issue was fixed in the ADIOS 1 repository (https://github.com/ornladios/ADIOS). Our documentation (https://gkyl.readthedocs.io/en/latest/install.html#postgkyl-install) is up-to-date.

New Conda Installation

```
conda install -c gkyl -c conda-forge postgkyl
```

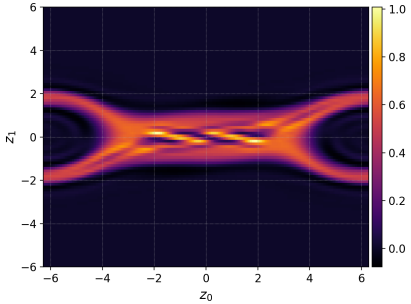
Sadly, this is not available for the new AMD64 Macs. ADIOS has to be installed manually using the `wrappers/numpy/` from the ADIOS repository.

Outline

Outline

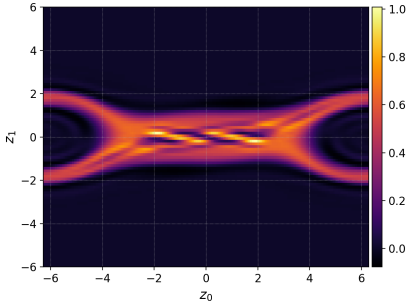
Using postgkyl in a command line is easy

```
pgkyl ts_elc_120.bp plot
```



Using postgkyl in a command line is easy

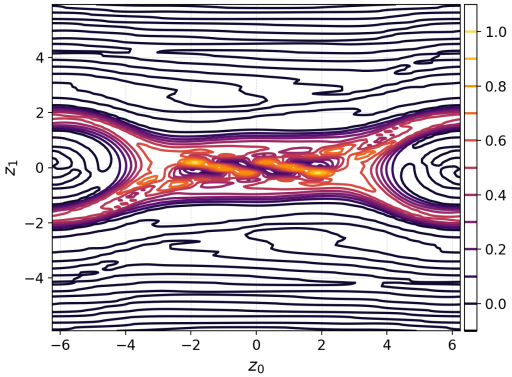
```
pgkyl ts_elc_120.bp plot
```



Too long? `pgkyl ts_elc_120.bp pl`

As usual, commands have parameters

```
pgkyl ts_elc_120.bp plot --contour
```



Help is automatically available

`pgkyl plot --help`

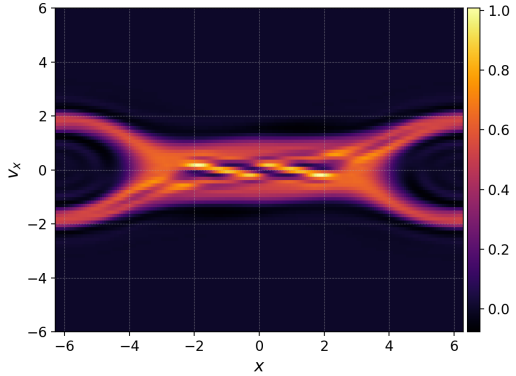
```
Usage: pgkyl plot [OPTIONS]

  Plot active datasets, optionally displaying the plot and/or saving it to PNG
  files. Plot labels can use a sub-set of LaTeX math commands placed between
  dollar ($) signs.

Options:
  -u, --use TEXT                  Specify the tag to plot.
  -f, --figure TEXT               Specify figure to plot in; either number or
                                  'dataset'.
  -s, --squeeze                   Squeeze the components into one panel.
  -b, --subplots                  Make subplots from multiple datasets.
```

This allows for publication-ish level quality plots

```
pgkyl ts_elc_120.bp pl -x '$x$' -y '$v_x$' --saveas 'two_stream.png'
--dpi 200
```



Outline

Postgkyl commands are Python function wrappers

```
import postgkyl as pg
data = pg.Data('ts_elc_120.bp')
```

[1]:
```
import postgkyl as pg
data = pg.Data('ts_elc_120.bp')
values = data.getValues()
print(type(values), values.shape)
print(values)
```

```
<class 'numpy.ndarray'> (64, 32, 8)
[[[ 4.1920672607903512e-08  1.0679391082388565e-09
    5.7975057381756998e-08 ...  1.1284141504513793e-08
    1.6904731793777075e-10 -3.0839022169101122e-10]
  [ 7.3547898532576669e-08 -6.0878231916690897e-09
    2.8190484930433895e-07 ...  2.1787790557862201e-07
    3.6956121684888986e-10 -1.0749171264064001e-08]
  [-4.5007521912656645e-07 -2.7435820337013622e-08
    3.9261544059914320e-07 ...  1.0764238463368680e-06
   -9.0163631933690787e-10 -9.0180903621736144e-08]
  ...
```

Some people might prefer other postprocessing tools

`Postgkyl` can write comma-separated text files

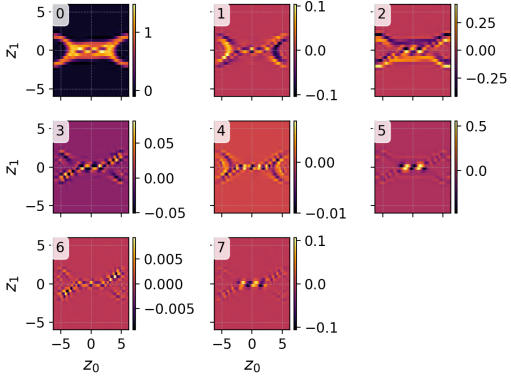
```
pgkyl ts_elc_120.bp write -f twostream.txt
```

This command can also create ADIOS files and, for example, put multiple `Gkeyll` outputs into one.

Outline

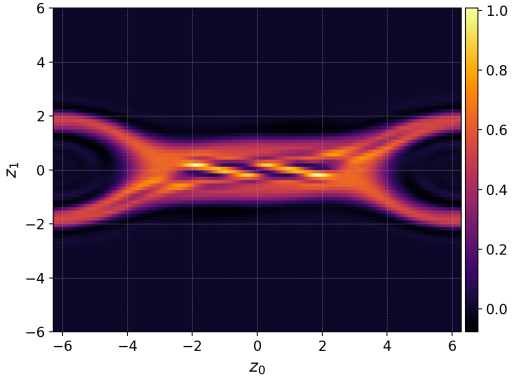
OK, maybe it isn't *that* easy... it really does this

```
pgkyl ts_elc_120.bp plot
```



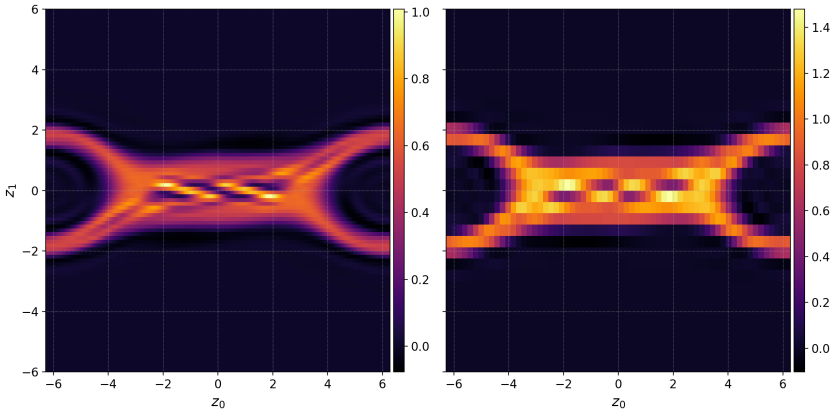
Commands can be chained indefinitely!

```
pgkyl ts_elc_120.bp interpolate plot
```



Commands can be chained indefinitely!

```
pgkyl ts_elc_120.bp interpolate plot
```



Side note: `interpolate` might require parameters

`interpolate` requires specifying a polynomial order and basis

```
pgkyl ts_elc_120.bp interpolate -p 2 -b ms plot
```

Modern g2 ADIOS output files include this information and `Postgkyl` can access it

There is a hidden `load` command

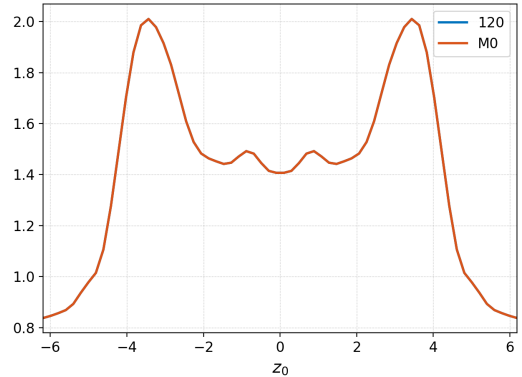
When a name is specified which does not match any existing command but match a file, a hidden `load` command is invoked

This is good to know for two reasons:

1. `help` can be used: `pgkyl load --help`
2. Loading can be chained as any other command

Commands apply to (by default all) previously loaded data

```
pgkyl ts_elc_120.bp select -c0 integrate 1 ev 'f 2 sqrt /'
ts_elc_M0_120.bp select -c0 plot -f0
```



With great power comes great responsibility

```
pgkyl ts_elc_120.bp interpolate integrate 1 ts_elc_M0_120.bp
interpolate plot -f0
```

```
Traceback (most recent call last):
  File "/home/pcagas/anaconda/bin/pgkyl", line 33, in <module>
    sys.exit(load_entry_point('postgkyl', 'console_scripts', 'pgkyl')())
  File "/home/pcagas/anaconda/lib/python3.8/site-packages/click/core.py
    ", line 1128, in __call__
    return self.main(*args, **kwargs)
  File "/home/pcagas/anaconda/lib/python3.8/site-packages/click/core.py
    ", line 1053, in main
    rv = self.invoke(ctx)
  File "/home/pcagas/anaconda/lib/python3.8/site-packages/click/core.py
    ", line 1690, in invoke
    rv.append(sub_ctx.command.invoke(sub_ctx))
  File "/home/pcagas/anaconda/lib/python3.8/site-packages/click/core.py
    ", line 1395, in invoke
    return ctx.invoke(self.callback, **ctx.params)
```

The info command comes to the rescue

```
pgkyl ts_elc_120.bp info interpolate integrate 1 ts_elc_M0_120.bp
interpolate plot -f0
```

```
(default#0)
 Time: 6.000000e+01
 Frame: 120
 Number of components: 8
 Number of dimensions: 2
 Grid: (uniform)
   Dim 0: Num. cells: 64; Lower: -6.283185e+00; Upper: 6.283185e+00
   Dim 1: Num. cells: 32; Lower: -6.000000e+00; Upper: 6.000000e+00
 Maximum: 1.479932e+00 at (41, 15) component 0
 Minimum: -4.786810e-01 at (31, 15) component 5
 DG info:
   Polynomial Order: 2
   Basis Type: serendipity (modal)
 Created with Gkeyll:
   Changeset: fc837ce8fc65+
```

The info command comes to the rescue

```
pgkyl ts_elc_120.bp interpolate info integrate 1 ts_elc_M0_120.bp
interpolate plot -f0
```

```
(default #0)
 Time: 6.000000e+01
 Frame: 120
 Number of components: 1
 Number of dimensions: 2
 Grid: (uniform)
   Dim 0: Num. cells: 192; Lower: -6.283185e+00; Upper: 6.283185e+00
   Dim 1: Num. cells: 96; Lower: -6.000000e+00; Upper: 6.000000e+00
 Maximum: 1.008332e+00 at (67, 49)
 Minimum: -7.723360e-02 at (184, 38)
 DG info:
   Polynomial Order: 2
   Basis Type: serendipity (modal)
 Created with Gkeyll:
   Changeset: fc837ce8fc65+
```

The info command comes to the rescue

```
pgkyl ts_elc_120.bp interpolate integrate 1 info ts_elc_M0_120.bp
interpolate plot -f0
```

```
(default#0)
 Time: 6.000000e+01
 Frame: 120
 Number of components: 1
 Number of dimensions: 2
 Grid: (uniform)
   Dim 0: Num. cells: 192; Lower: -6.283185e+00; Upper: 6.283185e+00
   Dim 1: Num. cells: 1; Lower: 0.000000e+00; Upper: 0.000000e+00
 Maximum: 1.425147e+00 at (43, 0)
 Minimum: 5.912497e-01 at (0, 0)
 DG info:
   Polynomial Order: 2
   Basis Type: serendipity (modal)
 Created with Gkeyll:
   Changeset: fc837ce8fc65+
```

The info command comes to the rescue

```
pgkyl ts_elc_120.bp interpolate integrate 1 ts_elc_M0_120.bp info
interpolate plot -f0
```

```
(default#0)
 Time: 6.000000e+01
 Frame: 120
 Number of components: 1
 Number of dimensions: 2
 Grid: (uniform)
   Dim 0: Num. cells: 192; Lower: -6.283185e+00; Upper: 6.283185e+00
   Dim 1: Num. cells: 1; Lower: 0.000000e+00; Upper: 0.000000e+00
 Maximum: 1.425147e+00 at (43, 0)
 Minimum: 5.912497e-01 at (0, 0)
 DG info:
   Polynomial Order: 2
   Basis Type: serendipity (modal)
 Created with Gkeyll:
   Changeset: fc837ce8fc65+
```

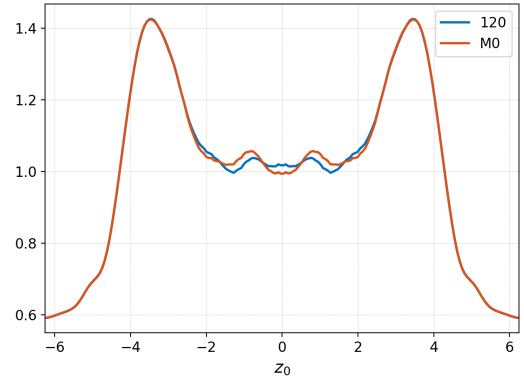
The `info` command comes to the rescue

```
18 M0 (default#1)
19  Time: 6.000000e+01
20  Frame: 120
21  Number of components: 3
22  Number of dimensions: 1
23  Grid: (uniform)
24    Dim 0: Num. cells: 64; Lower: -6.283185e+00; Upper: 6.283185e+00
25  Maximum: 2.010040e+00 at (14,) component 0
26  Minimum: -6.286080e-02 at (53,) component 1
27  DG info:
28    Polynomial Order: 2
29    Basis Type: serendipity (modal)
30  Created with Gkeyll:
31    Changeset: fc837ce8fc65+
32    Build Date: Apr 13 2022 21:03:13
```

Outline

Datasets (i.e. loaded data) can be active or inactive

```
pgkyl ts_elc_120.bp interpolate integrate 1 ts_elc_M0_120.bp activate
-i 1 interpolate activate plot -f0
```



The info can be again useful for understanding

```
pgkyl ts_elc_120.bp interpolate integrate 1 info -ca ts_elc_M0_120.bp
activate -i 1 interpolate activate plot -f0
```

```
(default#0)
```

The info can be again useful for understanding

```
pgkyl ts_elc_120.bp interpolate integrate 1 ts_elc_M0_120.bp info -ca
activate -i 1 interpolate activate plot -f0
```

```
120 (default#0)
M0 (default#1)
```

The info can be again useful for understanding

```
pgkyl ts_elc_120.bp interpolate integrate 1 ts_elc_M0_120.bp activate
-i 1 info -ca interpolate activate plot -f0
```

```
120 (default#0)
M0 (default#1)
```

The info can be again useful for understanding

```
pgkyl ts_elc_120.bp interpolate integrate 1 ts_elc_M0_120.bp activate
-i 1 interpolate activate info -ca plot -f0
```

```
120 (default#0)
M0 (default#1)
```

Relatively newer addition to Postgkyl are tags

```
pgkyl ts_elc_120.bp -t f ts_elc_M0_120.bp -t M0 interpolate integrate
-u f 1 plot -f0
```

Relatively newer addition to Postgkyl are tags

```
pgkyl ts_elc_120.bp -t f ts_elc_M0_120.bp -t M0 info -ca interpolate
integrate -u f 1 plot -f0
```

```
120 (f#0)
M0 (M0#0)
```

Relatively newer addition to Postgkyl are tags

```
pgkyl ts_elc_120.bp -t f ts_elc_M0_120.bp -t M0 interpolate integrate
-u f 1 info -ca plot -f0
```

```
120 (f#0)
M0 (M0#0)
```

You might argue that the f tag is not valid after integration...

```
pgkyl ts_elc_120.bp -t f ts_elc_M0_120.bp -t M0 interpolate integrate
-u f -t M0 1 plot -u M0 -f0
```

You might argue that the f tag is not valid after integration...

```
pgkyl ts_elc_120.bp -t f ts_elc_M0_120.bp -t M0 interpolate integrate
-u f -t M0 1 info -ca plot -u M0 -f0
```

```
120 (f#0)
M0 (M0#0)
(M0#1)
```

You might argue that the f tag is not valid after integration...

```
pgkyl ts_elc_120.bp -t f ts_elc_M0_120.bp -t M0 interpolate integrate
-u f -t M0 -l ':-)' 1 info -ca plot -u M0 -f0
```

```
120 (f#0)
M0 (M0#0)
:-) (M0#1)
```

Postgkyl can load multiple files at once!

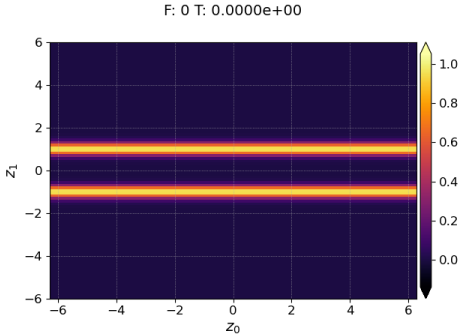
We can use wildcard characters! `pgkyl ts_elc_*.bp`

Keep in mind there is a difference between `pgkyl ts_elc_*.bp` and `pgkyl 'ts_elc_*.bp'`

1. Ordering; with quotes `'ts_elc_10.bp'` comes after `'ts_elc_2.bp'`
2. Shell wildcard support, `'ts_elc_[0-9]*.bp'`
3. Application of tags, `pgkyl 'ts_elc_[0-9]*.bp' -t elc`

Why would we want to load a bunch of files?

Batch processing: `pgkyl 'ts_elc_[0-9]*.bp' interp wr -f ts`
Animations! `pgkyl 'ts_elc_[0-9]*.bp' interp anim`



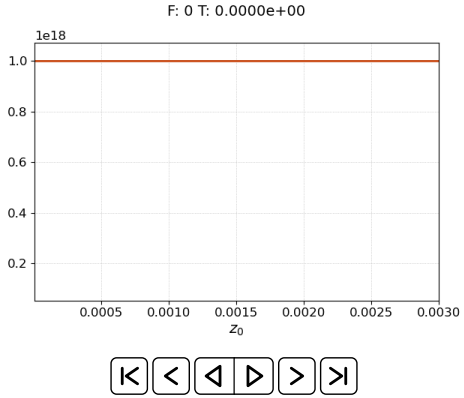
Side note: the previous frame

```
pgkyl 'ts_elc_[0-9]*.bp' interp anim --saveframes ts
```

```
1  Batch processing: \texttt{pgkyl 'ts\_elc\_[0-9]*.bp' interp wr -f
2    ts}
3
4  Animations! \texttt{pgkyl 'ts\_elc\_[0-9]*.bp' interp anim}
5
6  \begin{figure}
7    \animategraphics[controls={play,step,stop},width=0.45\linewidth]{10}
8                    {fig/ts_}{0}{120}
9  \end{figure}
```

The animate command can now group tags

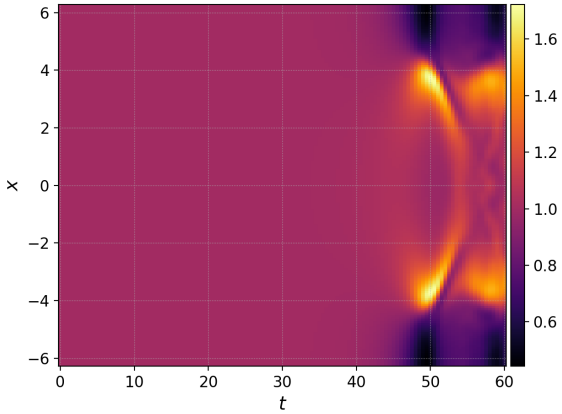
```
pgkyl 'rt-vm-sheath02-1x1v-p2_elc_M0_*.bp' -t elc
'rt-vm-sheath02-1x1v-p2_ion_M0_*.bp' -t ion interp anim --grouptags
```



Outline

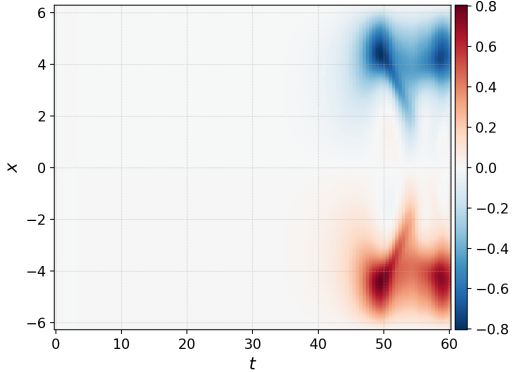
The collect command can group data in time

```
pgkyl 'ts_elc_M0_[0-9]*.bp' interp collect plot -x '$t$' -y '$x$'
```



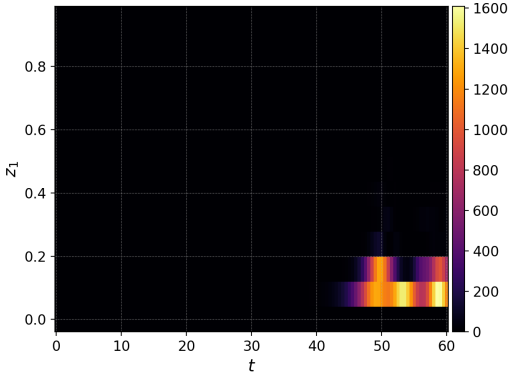
The collect command can group data in time

```
pgkyl 'ts_field_[0-9]*.bp' interp sel -c0 collect plot -d -x '$t$' -y
'$x$'
```

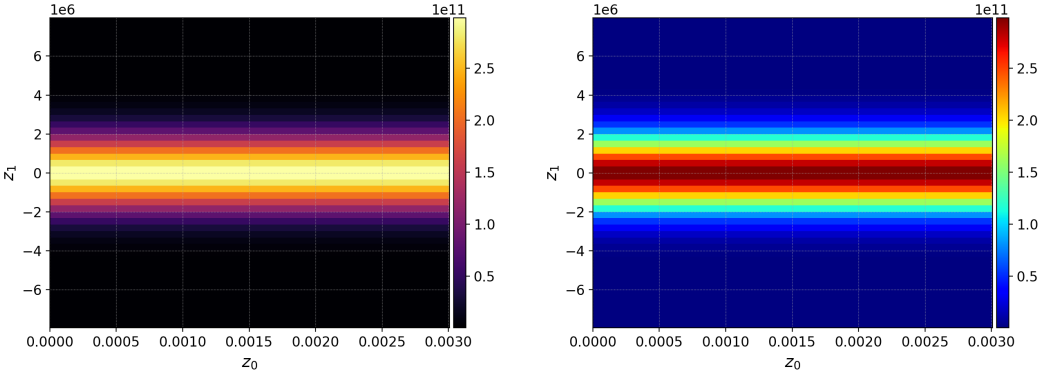


This can be used to plot spectrograms in a terminal

```
pgkyl 'ts_field_[0-9]*.bp' interp sel -c0 fft -p collect sel --z1 :1.0
plot -x '$t$'
```



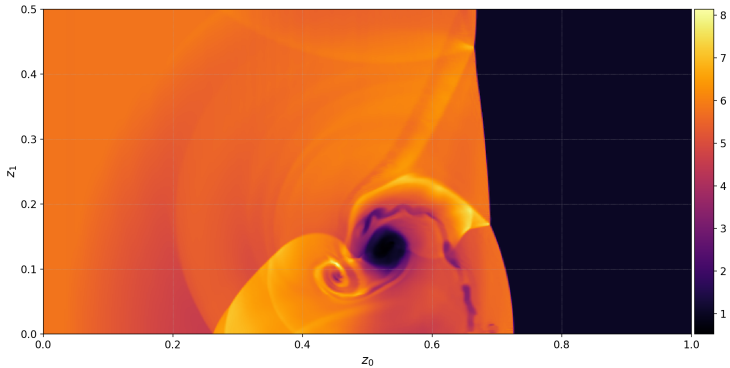
Postgkyl does not use jet color-map by default



Outline

This might happen to you as well...

Ammar: I am on a plane, I have this density data, and want to create a synthetic Schlieren data before the plane touches the ground.[1]



[1] This might have happened slightly differently :-)

In case you don't know like I didn't

We want to plot $\sqrt{\nabla\rho \cdot \nabla\rho}$

There is no `Postgkyl` command to calculate that.

In case you don't know like I didn't

We want to plot $\sqrt{\nabla\rho \cdot \nabla\rho}$

There is no `Postgkyl` command to calculate that.

There actually kind of is

```
pgkyl ts_elc_120.bp select -c0 integrate 1 ev 'f 2 sqrt /'
ts_elc_M0_120.bp select -c0 plot -f0
```

The ev is a very flexible command for general operations

ev uses Reverse Polish Notation (RPN), also know as Polish postfix notation (https://en.wikipedia.org/wiki/Reverse_Polish_notation)

Let's explain it on the previous example: `ev 'f 2 sqrt /'`

1. `f` adds data to the stack
2. 2 adds 2 to the stack
3. `sqrt` removes the top of the stack (2) and adds a square root of it back; now the stack has `f` and $\sqrt{2}$
4. `/` removes two top things from the stack (`f` and $\sqrt{2}$) and divides them; returns the result ($\mathtt{f}/\sqrt{2}$) to the stack

There are several ways to pass data to ev

When *not* using tags, `f[id#][component#]` stands for specified component(s) of a dataset. Python notation does work her, e.g., `f[#][1:4]` selects a momentum components of a classical fluid data. When selecting all, `[:]` can be omitted. This does take into account active/inactive data and only the result is set to be active.

```
pgkyl rt-euler-shock-bubble-hllc-fluid_40.gkyl ev 'f[0][1] f[0][0] /'
info -ca
(default#0)
```

```
pgkyl rt-euler-shock-bubble-hllc-fluid_40.gkyl
rt-euler-shock-bubble-hllc-fluid_40.gkyl ev 'f[1][1] f[0][0] /' info
-ca
(default#0)
(default#1)
f[1][1] f[0][0] / (default#2)
```

There are several ways to pass data to ev

With tags, situation is similar but tag name replaces the `f`. In this case active/inactive status is ignored.

```
pgkyl rt-euler-shock-bubble-hllc-fluid_40.gkyl -t r0
rt-euler-shock-bubble-hllc-fluid_40.gkyl -t r1 ev -t r2 'r0[0][1]
r1[0][0] /' info -ca
(r0#0)
(r1#0)
r0[1][1] r1[0][0] / (r2#0)
```

There are several ways to pass data to ev

Batch operations are supported

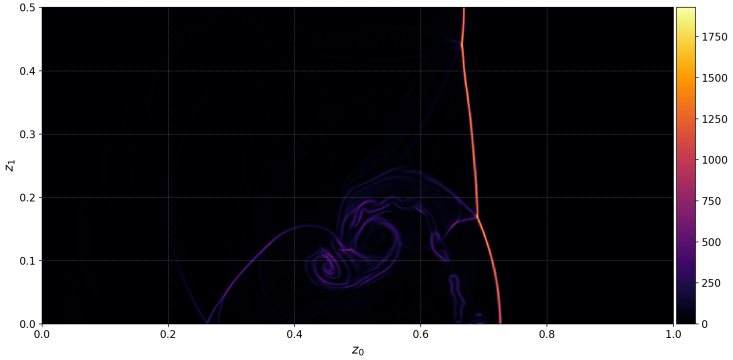
```
pgkyl rt-euler-shock-bubble-hllc-fluid_40.gkyl
rt-euler-shock-bubble-hllc-fluid_40.gkyl ev 'f[:][1] f[0][0] /' info
-ca
(default#0)
(default#1)
f[:][1] f[0][0] / (default#2)
f[:][1] f[0][0] / (default#3)
```

Similar operations might make more sense for g2 outputs which can include meta data like mass.

```
pgkyl 'fluid_data_[0-9]*.bp' ev 'f[:][0] f[0].mass /'
```

Now we can get the Schlieren data

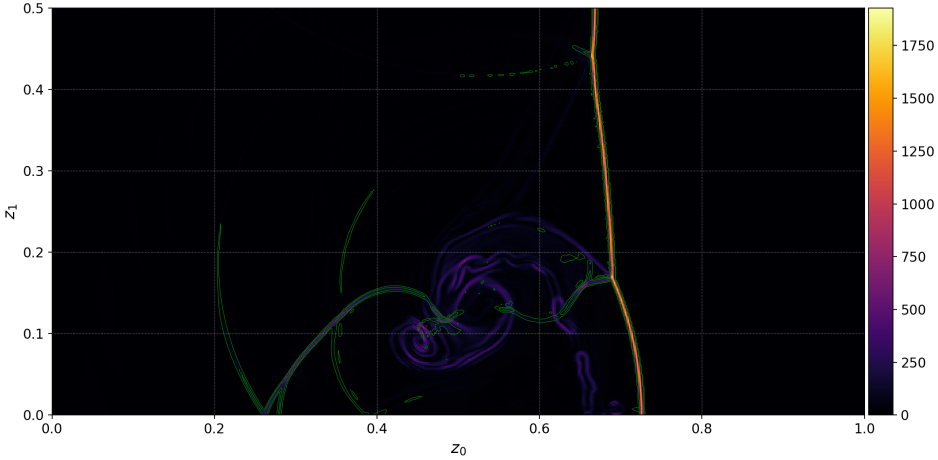
```
pgkyl rt-euler-shock-bubble-hllc-fluid_40.gkyl ev 'f[0][0] grad f[0][0] grad dot sqrt' pl -a
```



Too easy... let's add shock detector contours

```
pgkyl rt-euler-shock-bubble-hllc-fluid_40.gkyl -t inp sel -u inp
-c1,2,3 -t rhou sel -u inp -c0 -t rho ev -t vel "rhou rho /" ev -t
div2 "vel div sq" ev -t Omega "vel vel dot sqrt 0.1 * 1.0 200 / /" ev
-t Theta "div2 div2 Omega sq + 1e-10 + /" ev -t curl "v el curl" ev
-t curl2 "curl curl dot" ev -t D0 "div2 div2 curl2 + 1e-10 + /" ev -t
alpha "D0 Theta *" ev -t schlieren 'inp[0][0] grad inp[0][0] grad dot
sqrt' pl -u schlieren -a --no-legend --no-show -f0 --figsize 12,6 pl
-u alpha -c --clevels 0.1:1:3 --linewidth 0.5 --color "g" -a -f0
--no-legend
```

Too easy. . . let's add shock detector contours



Need more help?

The documentation is behind but we are trying!

Submit GitHub issues (https://github.com/ammarhakim/postgkyl/issues) and tag me when appropriate (which is sadly quite often) and/or contact me directly on Slack