Performance analysis

Let's make a fun little investigation. Assume we have some data of a signal and we want to make an algorithm that tracks the 1-d signal. This presentation is a little bit too short to make a real algorithm and data in so we will fake it instead

Synthesize the data

To generate the true signal and the estimate we will use a block of elisp code.

```
(mapcar (lambda (i)
          (list i (+ (random 4) (- i 2))))
        (number-sequence 1 20))
```

Truth	Estimate
1	2
2	1
3	2
4	5
5	5
6	7
7	7
8	7
9	7
10	8
11	9
12	12
13	12
14	13
15	13
16	16
17	16
18	19
19	19
20	20

Table 1: Truth and Estimate

In order to get the table of data more comprehensive we utilize another block to post process the result. This block adds a header to the table.

```
(cons 'hline (cons '("Truth" "Estimate") (cons 'hline tbl)))
```

It's good to see the data in the table but I often find that some kind of visualization is more powerful. It's too bad I don't know how to plot in elisp... but I do know how to do it in Python.

Visualize estimate and truth

Here is some python code which can plot data using the matplotlib. So as a variable to this code block we will pass the numbers from the `est-truth-data` block.

```
import numpy as np
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt

# Convert list to numpy array
truth = np.asarray(data)[:, 0]
est = np.asarray(data)[:, 1]
# Plot
fig=plt.figure()
plt.plot(truth, color="g", label="Truth")
plt.plot(est, marker="x", label="Estimate")
plt.legend(loc='upper left')
plt.xlabel("Sample")
plt.ylabel("Value")
plt.title("Tracking")
plt.savefig('.images/est_vs_truth.png')
'.images/est_vs_truth.png' # return this to org-mode
```

Cool, the performance of the fake algorithm is not that bad. I think we can be pretty happy with it. let's see if we can gather some more information about it's performance.

Examine the performance

A table can be a good way of displaying the data that we have and analyze the values. Tables in Emacs can use `calc` syntax org `elisp` code to make the table formulas.

Truth	Estimate	Error	Absolute error
1	2	-1	1
2	1	1	1
3	4	-1	1
4	5	-1	1
5	3	2	2
6	7	-1	1
7	8	-1	1
8	9	-1	1
9	7	2	2
10	9	1	1
11	11	0	0
12	13	-1	1
13	11	2	2
14	15	-1	1
15	13	2	2
16	15	1	1
17	17	0	0
18	18	0	0
19	19	0	0
20	19	1	1
Number of values	20		
Mean error	0.2		
RMSE	1.1832159566199232		

Table 2: Evaluation numbers

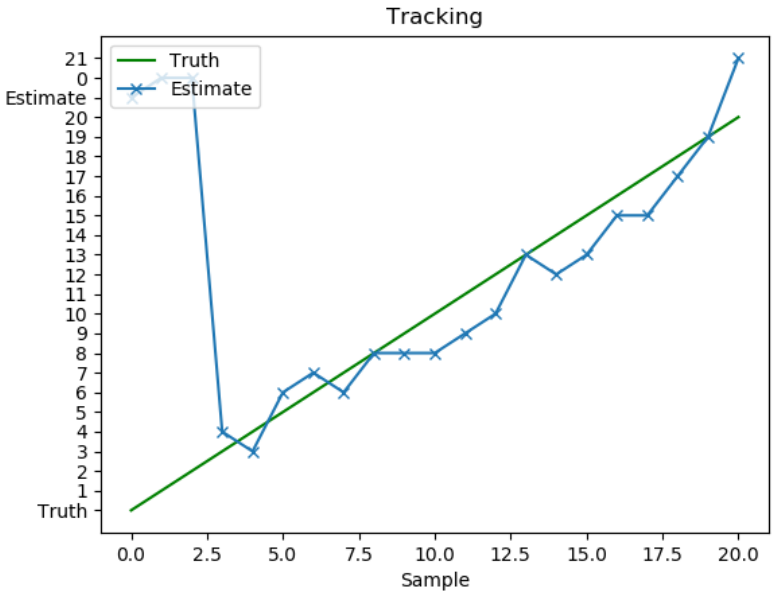

Figure 1: Tracking the true value

In order to get the values from the other table I am using remote references. To refer to the values of the other table. The formulas that the table accepts should either follow the `calc` syntax or `elisp`.

More complex formulas

It would be nice to get a KPI like rmse to have one number for the performance of the algorithm. Since that equation would be quite long in Emacs lisp maybe it's time to try something else.

We can actually pass the data from the table into other code blocks, which is a super cool. We can therefore create a block with Python code which we pass values into in order to be able to calculate the

Or perhaps a block of elisp to calculate the mean value

Describe the flow

I would like to describe the flow better. It would be great if we could visualize it, perhaps in a flow chart.

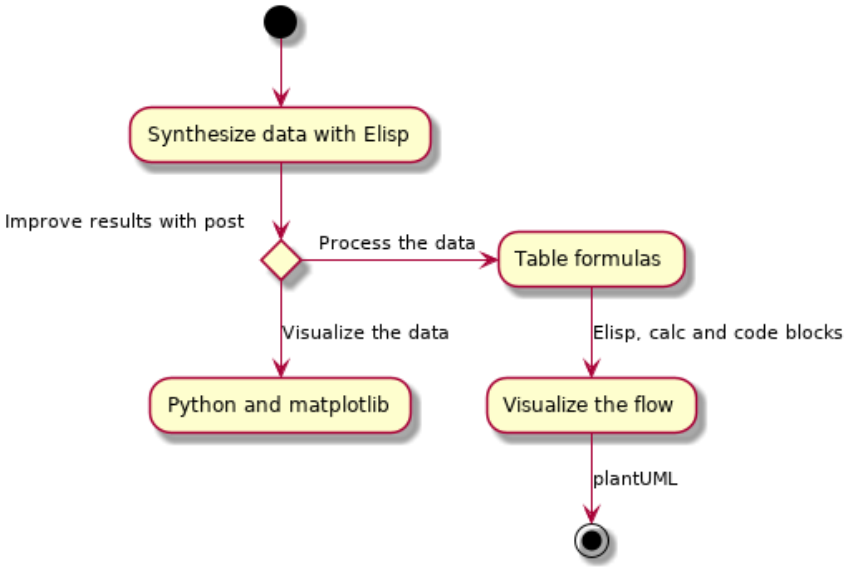

Figure 2: The flow of our investigation

Export

I think that our investigation here has been a success and it would be great if we can share the findings with our colleges. Unfortunately not all of them have access to Org-mode and can read the information in this format. Cause as we saw before this is just plain text so it won't look as nice outside this environment.

Luckily Org-mode supports a lot of different exports. I am thinking for this particular use case something common like a `pdf` would be a good choice. We also would like it to look nice and professional so let's make it a `LaTeX` styled pdf.

To export we only need to use the function `M-x org-latex-export-to-pdf`. There are two other alternatives here using `pandoc` but I didn't find the export as good so I will choose the first one.

One thing that did bother me though is that now I exported the file and the next step would naturally be to look at the results. If you noticed with pandoc there was a function called `M-x org-pandoc-export-to-latex-pdf-and-open`. I want that too cause if we open `dired` the directory editor we can see that the file is indeed here.

But I want this automated, good thing that we are using Emacs then, let's create the function we need.

Improve the export

So I already prepared for this and this is the elisp code we need to have a function which also will open the pdf after the export has finished.

```
(defun org-latex-export-to-pdf-and-open ()
"Export current buffer to LaTeX then process through to PDF and open the
resulting file"
  (interactive)
  (let* ((file-name (file-name-nondirectory buffer-file-name))
         (name (file-name-sans-extension file-name)))
  (org-latex-export-to-pdf)
  (find-file (concat name ".pdf"))))
```

We can see that it has the name `pdf-and-open`. Firstly it retrieves the name of the current file, without the extension. Then it calls the regular function and lastly opens the exported PDF.

To install it I just open it in the popup buffer and evaluate the function and now when I search through `M-x` I will find the function and it will do exactly what we want.

Appendix: Emacs Calc

Now I thought that this article deserved some bonus information about Emacs calculator. I knew about it before but writing this part got me interested into knowing more. And I think that it was a good example of one of the aspects that I love about Org-mode which is to be able to experiment. I found some useful calc-babel-examples and org-examples which I took inspiration from.

We can start of simple with some arithmetic:

```
10 + 3 * 5 /2.0
```

But the calculator knows more than that. We can also calculate `cosine`. It seems however that it doesn't use SI units which is probably some setting somewhere that needs to be set.

```
cos(180)
```

Oh but let's have a look at this equation system, could calc solve that for us too?

Yes we can!

```
fsolve(x*2+x=4,x)
```

So wait if we can do that, and it's supported in formulas that would mean that we would be able to use it like this:

Expression	Derivative
sqrt(x)	
cos(x)	
x^2	
$1/x^2$	

The conclusion is that the Emacs calc is indeed very powerful and the reputation of being the poor man's mathematica does indeed seem to be true.