1 TODO Side by side figures in org-mode for different outputs

Occasionally, someone wants side by side figures with subcaptions that have individually referenceable labels. This is not too hard in LaTeX, and there is a solution here: http://www.johndcook.com/blog/2009/01/14/how-to-display-side-by-side-figurs-in-latex/.

If we have a single figure like this:

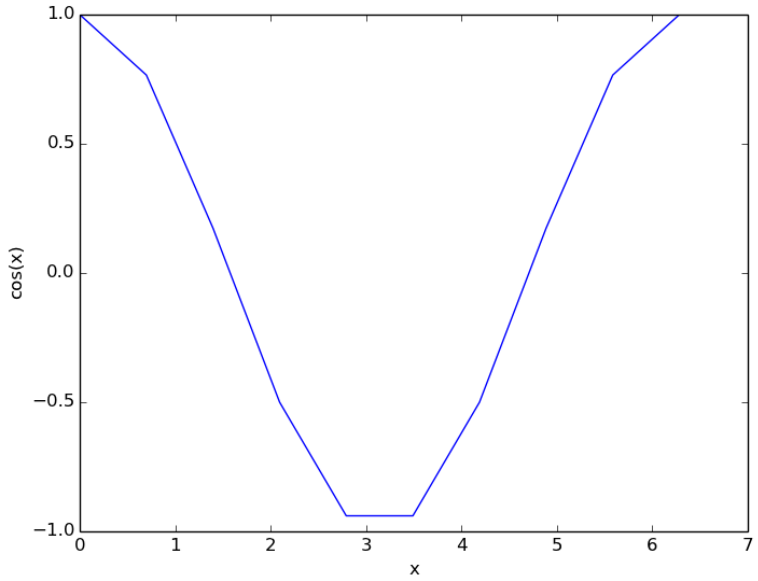


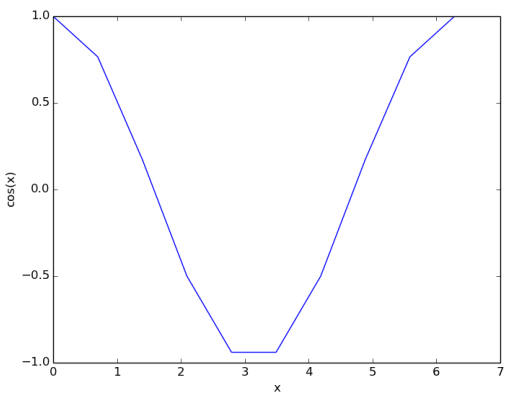
Figure 1: Single graph

We can reference it as Figure 1.

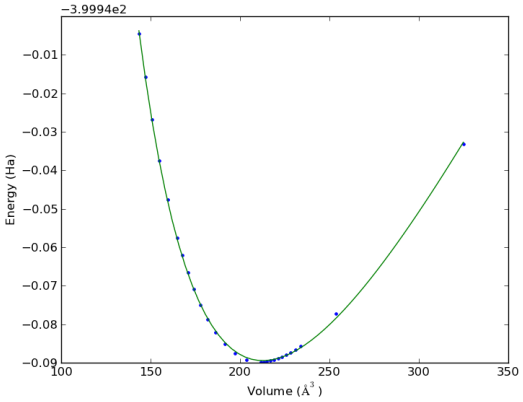
We can create side by side figures in raw LaTeX like this (note however, this will not show up in html export):

And in our text we can refer to the overall Figure 4, or the subfigures Figure 4(a) or Figure 4(b). This works fine if your end goal is LaTeX export. It does not work fine if you want to consider HTML or some other output.

So, here we consider how we could remove the LaTeX dependency by representing the figures in a sexp data structure, for example something like this:



(a) Left graph



(b) Right graph.

Figure 2: Text pertaining to both graphs,4(a)and 4(b).

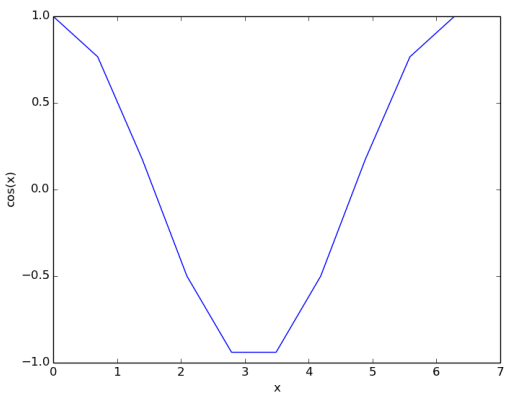
```
(figure ()
 (subfigure '(("Left graph" (label "fig:a")))
            (includegraphics '((width . "3in"))
                             "images/cos-plot.png"))
 (enskip)
 (subfigure '(("Right graph" (label "fig:b")))
            (includegraphics '((width . "3in"))
                             "images/eos-uncertainty.png"))
 (caption
  "Text pertaining to both graphs, " (ref "fig:a")
  " and " (ref "fig:b") "." (label "fig12")))
```

This doesn't look much worse than the LaTeX code itself. It might not seem useful right away, but imagine if this was really code that could evaluate to the format we want. Remember the sexp bibtex entry that could evaluate to bibtex, json or xml? Let's look at this here. What we consider is kind of like http://oremacs.com/2015/01/23/eltex/, but we could include other kinds of exports if we wanted.

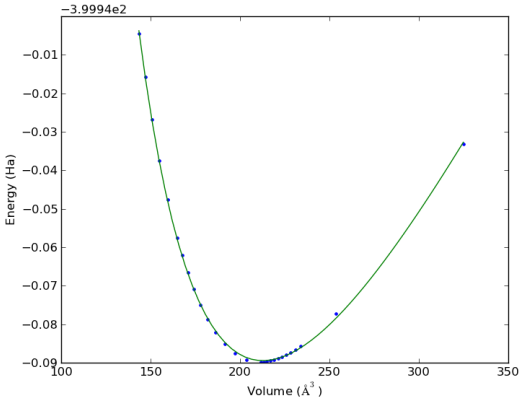
Here is our special block in org-mode. It should render roughly as side by side images in LaTeX and HTML.

Now, we need a function to format the sexp block. It is easy, we just eval the contents of the block. We do assume here there is just one sexp to evaluate.

```
(defun sb-format (sb contents info)
  (cond
```



(a) Left graph



(b) Right graph

Figure 3: Text pertaining to both graphs, 4(a) and 4(b).

```
      ((string= "SEXP" (org-element-property :type sb))
       (eval (read (buffer-substring
                    (org-element-property :contents-begin sb)
                    (org-element-property :contents-end sb)))))
      (t
       contents)))))
```

All that is left is to define the functions. We do that next.

`An Example block for testing purposes.`

1.1 Latex export

We need to define a function for each piece of the data structure that will evaluate to a string. Here are three easy ones.

```
(defun label (arg)
  (format "\\label{%s}" arg))

(defun ref (arg)
  (format "\\ref{%s}" arg))

(defun caption (&rest body)
  (format "\\caption{%s}"
          (mapconcat 'eval body "")))

(caption
  "Text pertaining to both graphs, " (ref "fig:a")
  " and " (ref "fig:b") "." (label "fig12"))
```

```
\caption{Text pertaining to both graphs, \ref{fig:a} and \ref{fig:b}.\label{fig12}}
```

Now, for includegraphics, we allow options and a path. The options we assume are in an a-list.

```
(defun includegraphics (options path)
  (format "\\includegraphics%s{%s}"
          (if options
              (format "[%s]"
                      (mapconcat (lambda (ccell)
                                   (format "%s=%s"
                                           (car ccell)
                                           (cdr ccell)))
                                 options
                                 ","))
            "")
          path))

(includegraphics '((width . "3in"))
                 "images/eos-uncertainty.png")
```

```
\includegraphics[width=3in]{images/eos-uncertainty.png}
```

Similarly for subfigure, we have options, and then a body of expressions. The options here are just expressions that should evaluate to strings. This is not consistent with the way we do options in includegraphics. This is just proof of concept work, so I don't know if this inconsistency is really problematic yet, or insufficient for all options.

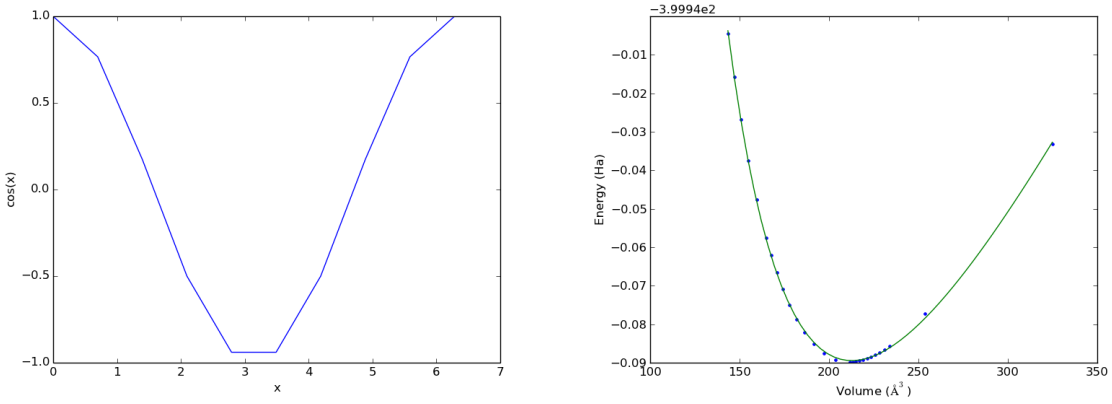
```
(defun subfigure (options &rest body)
  (format "\\subfigure%s{%s}"
          (if options
              (format "[%s]"
                      (mapconcat 'eval options ""))
            "")
          (mapconcat 'eval body "")))

(subfigure '("Right graph" (label "fig:b"))
           (includegraphics '((width . "3in"))
                            "images/eos-uncertainty.png"))
```

```
\subfigure[Right graph\label{fig:b}]{\includegraphics[width=3in]{images/eos-uncertain
```

Now, we put the whole figure together.

```
(defun figure (options &rest body)
  (format "\\begin{figure}
```



(a) Left graph

(b) Right graph

Figure 4: Text pertaining to both graphs, 4(a) and 4(b).

```
%s
\\end{figure}"
(mapconcat 'eval body "\n")))

(defun enskip () "\\enskip")
```

`enskip`

Now, we would have a block like this:

```
(figure ()
 (subfigure '("Left graph" (label "fig:a"))
            (includegraphics '((width . "3in"))
                             "images/cos-plot.png"))
 (enskip)
 (subfigure '("Right graph" (label "fig:b"))
            (includegraphics '((width . "3in"))
                             "images/eos-uncertainty.png"))
 (caption
  "Text pertaining to both graphs, " (ref "fig:a")
  " and " (ref "fig:b") "." (label "fig12")))
```

Now, we derive a new backend. An alternative approach would be to advise the special-block function to do the formatting we need.

```
(org-export-define-derived-backend 'my-latex 'latex
  :translate-alist '((special-block . sb-format)))
```

```
(org-latex-compile (org-export-to-file 'my-latex "custom-sb-export.tex"))
(org-open-file "custom-sb-export.pdf")
```

1.2 HTML functions

We need to define each element and its HTML output.

```
(defun label (arg)
  (format "<a name=\"%s\"></a>" arg))

(defun ref (arg)
  (format "<a href=\"#%s\">%s</a>" arg arg))

(defun caption (&rest body)
  (format "<caption>%s</caption>"
          (mapconcat 'eval body "")))

(caption
  "Text pertaining to both graphs, " (ref "fig:a")
  " and " (ref "fig:b") "." (label "fig12"))
```

```
<caption>Text pertaining to both graphs, <a href="#fig:a">fig:a</a> and <a href="#fig
```

We will ignore options for the includegraphics html output. We would need to specify some way to do unit conversions for html. Here we fix the width.

```
(defun includegraphics (options path)
  (format "<img src=\"%s\" width=\"300\">"
          path))

(includegraphics '((width . "3in"))
                 "images/eos-uncertainty.png")
```

```
<img src="images/eos-uncertainty.png" width="300">
```

We wrap a subfigure in a table cell.

```
(defun subfigure (options &rest body)
  (format "<td>%s%s</td>"
          (mapconcat 'eval body "")
          (when options
            (concat "<br>"
                    (mapconcat 'eval options "")))))

(subfigure '("Right graph" (label "fig:b"))
           (includegraphics '((width . "3in"))
                            "images/eos-uncertainty.png"))
```

```
<td><img src="images/eos-uncertainty.png" width="200"><br>Right graph<a name="fig:b">
```

We assume we can put the images in a single row.

```
(defun figure (options &rest body)
  (format "<span class=\"image\"><table>
<tr>%s</tr>
</table></span>"
(mapconcat 'eval body "\n")))

(defun enskip () "")
```

```
enskip
```

Now, here is our specification.

```
(figure ()
 (subfigure '("Left graph" (label "fig:a"))
            (includegraphics '((width . "3in"))
                             "images/cos-plot.png"))
 (enskip)
 (subfigure '("Right graph" (label "fig:b"))
            (includegraphics '((width . "3in"))
                             "images/eos-uncertainty.png"))
 (caption
  "Text pertaining to both graphs, " (ref "fig:a")
  " and " (ref "fig:b") "." (label "fig12")))
```

And our derived backend for HTML.

```
(org-export-define-derived-backend 'my-html 'html
  :translate-alist '((special-block . sb-format)))

(browse-url (org-export-to-file 'my-html "custom-sb-export.html"))
```

```
#<process open custom-sb-export.html>
```

1.3 Summary thoughts

I think I like the idea. Obviously there are differences between what is possible between LaTeX and HTML, notably the attributes that may or may not be supported between them, including the units of the width, labels, and references. I still have not figured out an elegant way to switch between LaTeX and HTML exports since there is basically one set of functions that need different outputs under different conditions.