# TITLE FIXME

Mitch Richling

YYYY-MM-DD FIXME

**Author:** *Mitch Richling*
**Updated:** *2022-05-08 11:48:20*
**Generated:** *2022-05-08 11:48:24*



## Contents

# 1 Math

Here is some math $5 + 3^4$.

Reminder: toggle inline preview of an expression with C-c C-x C-l

Here is some display math

$$\sum 4$$

# 2 Markup

## 2.1 Inline stuff

Some **bold** text.

Some *italics* text.

Some underlined text.

Some `verbatim` text.

Some `code` text.

Some ~~strike-through~~ text.

## 2.2 Structural stuff

### 2.2.1 Special Paragraphs

Here we have a quote:

> A human being is a part of a whole, called by us universe, a part limited in time and space. He experiences himself, his thoughts and feelings as something separated from the rest... a kind of optical delusion of his consciousness. This delusion is a kind of prison for us, restricting us to our personal desires and to affection for a few persons nearest to us. Our task must be to free ourselves from this prison by widening our circle of compassion

to embrace all living creatures and the whole of nature in its beauty. – Albert Einstein

We can also keep newlines intact in an indented paragraph:

Whales Weep Not!

They say the sea is cold, but the sea contains
the hottest blood of all, and the wildest, the most urgent.
...

– D.H. Lawrence

We can have a "verbatim" section with an "EXAMPLE" block.

```
    Here is some text.
Note that
 everything is    just as typed.
```

### 2.2.2 Tables

Table 1: A formatted table

| col 1 | col 2 | col 3 | Col 4 |
|---|---|---|---|
| another | bit | 1 | 1.00 |
| a | b | 2 | 4.00 |

**With formatting & a formula**

**Tables used to hold data** We can use tables to hold data for other blocks to read in the document. The section 9.2 shows how to access the table below from R. Note some specifics for R:

- We have no "top line" on the table – otherwise the row of titles is not recognized!!
- Spaces in column titles are transformed into periods for `data.frame` column names.
- Empty data cells will be `NA` in the `data.frame`
- Non-numeric columns will be "characters" not "factors"

| factor 1 | factor 2 | value 1 | value 2 |
|---|---|---|---|
| a | z | 1 | 5 |
| a | x | 2 | 6 |
| a | y | 3 | 7 |
| b | x | 4 | |

#### 2.2.3 Lists

Here is itemized list:

- first
- second
- third

Here is enumerated list:

1. First
2. Second
3. Third

A bit of both:

1. First
2. Second
    - first
    - second
    - third
3. Third

### 2.3 Todo/action items

#### 2.3.1 TODO:NEW This is a todo

#### 2.3.2 ACTION:DONE This is an action item – work speak. ;)

**CLOSED:** *[2015-11-01 Sun 22:36]* **DEADLINE:** *<2015-11-01 Sun>*

### 2.3.3 ACTION:NEW This is an item with sub-items [1/2]

**ACTION:DONE A subitem** **CLOSED:** *[2015-11-01 Sun 22:39]*

**ACTION:NEW Another subitem**

### 2.3.4 ACTION:NEW Here is an action item with list compoents [2/3]

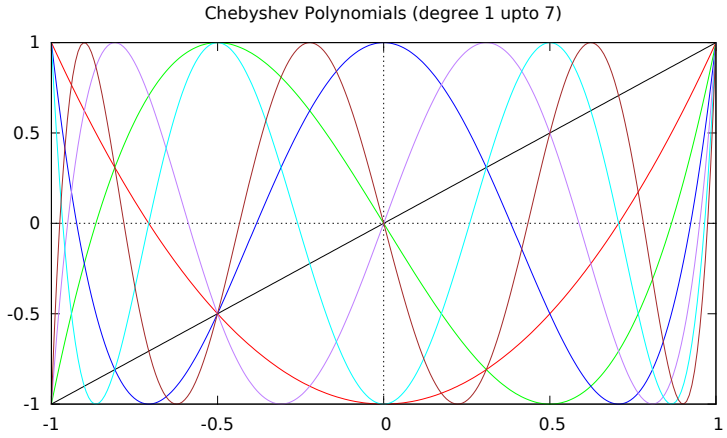
**DEADLINE:** *<2015-11-03 Tue>* **SCHEDULED:** *<2015-11-01 Sun>*

- ☐ Step 1
- ☒ Step 2
- ☒ Step 3

# 3 Images

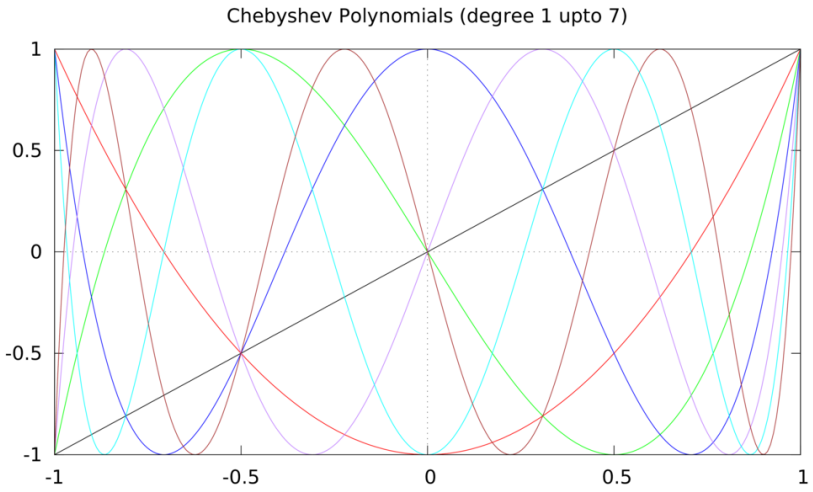
## 3.1 PDFs in LaTeX and Raster Image in HTML

In this Section you will see one image. A PNG for HTML, and a PDF for LaTeX!



### 3.2 Links to images and converting PDFs to high quality raster images

Here we have a pretty graph (in a PNG file):




The above file was generated from a high quality PDF file: example.pdf. Note that the link in the previous sentence is a link in both HTML and LaTeX because the link has a 'display text' component.

The conversion was done like so:

```
convert -density 600 -resize 1024 -background white -flatten example.pdf example.png
```

## 4 Including external code

Some Ruby code is the file `example.rb`. It's contents are listed below:

```
#!/usr/local/bin/ruby

##
# @file      hello.rb
# @author    Mitch Richling <http://www.mitchr.me/>
# @Copyright Copyright 2006 by Mitch Richling.  All rights reserved.
# @brief     The classic hello world program the Ruby way.@EOL
# @Keywords  ruby example hello world
# @Std       Ruby 1.8
```

```
#
#            The methods puts, print, printf & putc are all in the IO
#            class as well so that they can be used to write to
#            different IO streams.  As used here, they write to
#            STDOUT.

puts("Hello, World!")

print("Hello, World!\n")

printf("Hello, World!\n")

STDOUT << "Hello, World!\n"

STDOUT.write("Hello, World!\n")

"Hello, World!\n".each_byte {|b| putc(b) }
```

# 5 Inline Code

Here is a number, `(* 2 3) 6`, that comes from a bit of elisp code.

# 6 Code Blocks

## 6.1 Text code blocks

Text code blocks can be used as a kind of verbatum environment instead of BEGIN_EXAMPLE. This gives more control over formatting. See the Email example next.

```
> Some Mail
>> Some More
>>> Even More
>>>> Even more
```

## 6.2 Email code blocks

This is nice because we get some highlighting for quoted e-mails and threads.

```
> Some Mail
```

```
>> Some More
>>> Even More
>>>> Even more
```

### 6.3 Emacs Lisp

While you can use "`value`" insead of "`output`" for code blocks, it really is **very** usefull for Emacs Lisp.

Note that if you leave off the `:wrap` header argument, the result will be `emacs-lisp`. In this case the result will be an executable and eligible for tangle. When evaluating an entire document this can be used to advantage to sequentially evaluate code that geneates new code and then evaluatg eth enew code – you can even create an infinite loop with self printing code. ;)

```
(+ 1 2 3 5)
```

```
11
```

### 6.4 Emacs Calc

For more complex mathematical computations done with just Emacs (no outside tools) we can use calc.

```
deriv(3*x^2+log(x), x)
```

```
6 x + 1 / x
```

### 6.5 Maxima

For super complex math, we can use maxima.

Here we see a pretty printed result

```
programmode:false;
d:diff(3*x^2+log(x), x);
print(d);
```

```
      1
6 x + -
      x
```

Same answer, but 2D printed

```
programmode:false;
display2d:false;
d:diff(3*x^2+log(x), x);
print(d);
```

```
6*x+1/x
```

We can also output things in LaTeX so that the result is typeset on export!

```
programmode:false;
d:diff(3*x^2+log(x), x);
tex(d);
```

$$6\,x+\frac{1}{x}$$

Lastly we can use maxima to write code in other langauges for us. How some LaTeX exported as a code block?

```
programmode:false;
d:diff(3*x^2+log(x), x);
tex(d);
```

```
$$6\,x+{{1}\over{x}}$$
```

Or FORTRAN:

```
programmode:false;
d:diff(3*x^2+log(x), x);
fortran(d);
```

```
6*x+1/x
```

### 6.6 Shells

```
date "+%Y-%m-%d %H:%M:%S"
```

```
2020-07-21 16:27:29
```

### 6.7 Ruby

```
puts("HI MOM")
```

```
HI MOM
```

# 7 Fancy Code Block Stuff

## 7.1 Generateing code

```
(cl-loop for f in '("foo" "bar")
         do (princ (message "mv \"%s\" \"%s.bak\" # Rename %s\n" f f f)))
```

```
mv "foo" "foo.bak" # Rename foo
mv "bar" "bar.bak" # Rename bar
```

## 7.2 Code links

You can link to a target inside a code block: Visible Link Text

```
(cl-loop for f in '("foo" "bar")                        ;;      (foo)
         do (princ (message "mv %s %s.bak\n" f f)))
```

You can remove the visable refrences from the source listings with the -r option in the babel header. Otherwise they will appear in the listing – fontlocked white.

## 7.3 Line Numbers

This will include line numbers in the block when exported

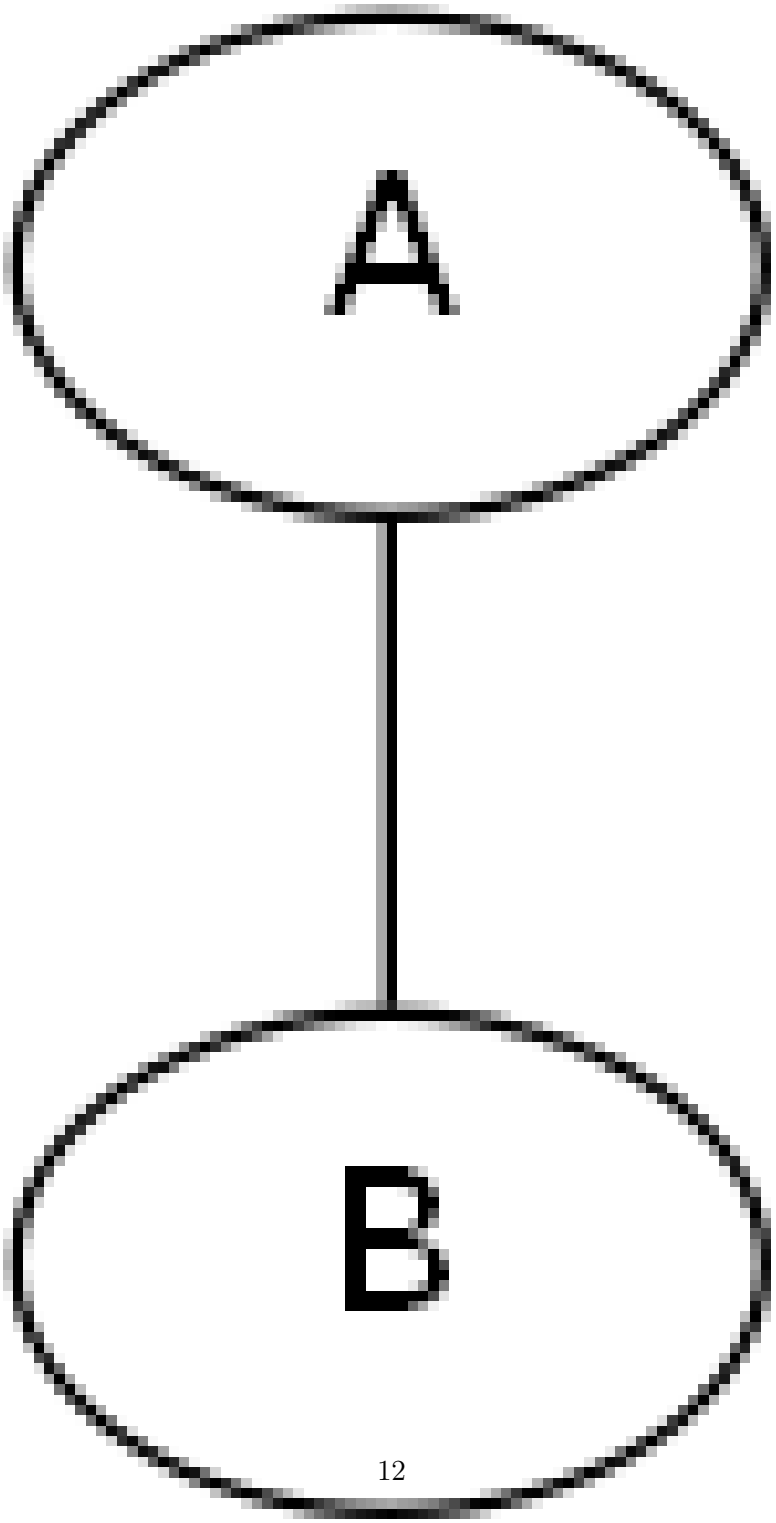
```
(cl-loop for f in '("a" "b" "c")
         do (princ (message "mv %s %s.bak\n" f f)))
```

```
1  mv a a.bak
2  mv b b.bak
3  mv c c.bak
```

# 8 dot

Here we do not export the code, just the results – as an image. This results in a nice rendering.

```
graph {
 A -- B;
}
```


A
B

# 9 R

## 9.1 Just run some R code in a new session

```
print("HI MOM")
```

```
[1] "HI MOM"
```

## 9.2 Access a table in this document as a data.frame

```
someData
```

```
  factor.1 factor.2 value.1 value.2
1        a        z       1       5
2        a        x       2       6
3        a        y       3       7
4        b        x       4      NA
```

## 9.3 Output from R as a org-mode table

```
someData
```

| | | | |
|---|---|---|---|
| a | z | 1 | 5 |
| a | x | 2 | 6 |
| a | y | 3 | 7 |
| b | x | 4 | nil |

## 9.4 Run some code in a R persistent session (the someData variable is available for later blocks)

```
someData <- data.frame(a=1:10, b=rnorm(10))
print(someData)
```

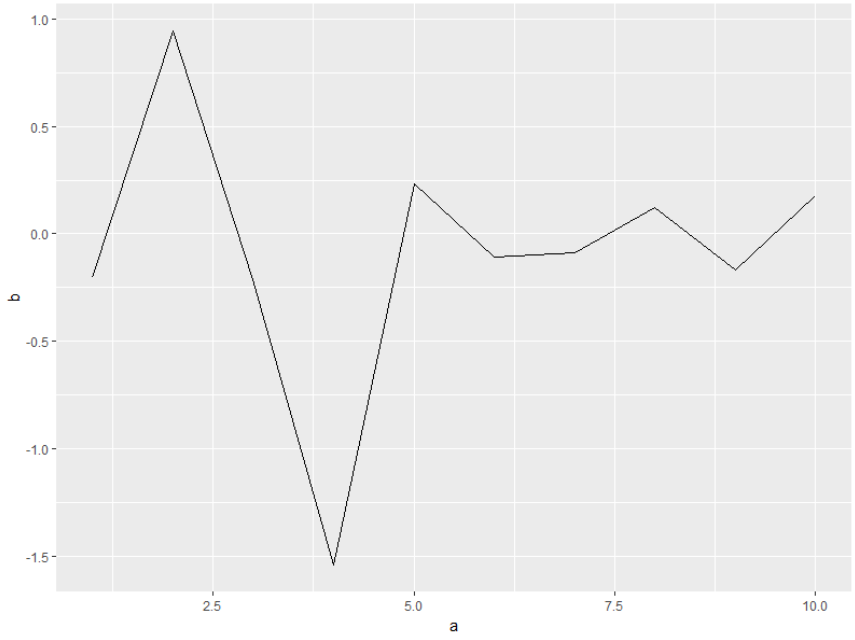
```
    a           b
1   1  0.379789874
2   2  1.998294645
3   3  0.466352572
4   4  0.001298143
5   5 -1.656637383
6   6 -1.037567358
7   7 -1.107296046
8   8 -1.653000457
9   9 -0.889483086
10 10 -0.387415308
```

### 9.5 Use the someData variable in the session, and draw a graph.

No speical org-mode stuff for graphics. Just saved the output in files via R. Add link text later.

```
g <- ggplot(someData, aes(x=a, y=b)) + geom_line()
ggsave("rOut1.png", width=8, height=6, dpi=100, units='in', plot=g);
ggsave("rOut1.pdf", width=8, height=6, dpi=600, units='in', plot=g);
```

The graph:

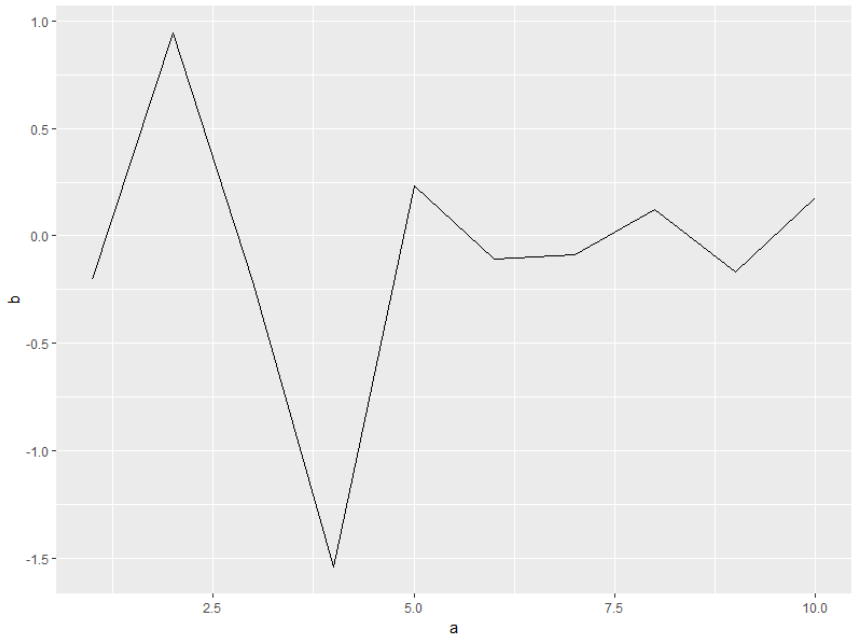



A high quality PDF version is here – note the "here" is a link for both LaTeX and HTML.

### 9.6 We can use org-mode to make the file too.

Note: :session is required for this to work – otherwise we must "print" the graphic.

```
ggplot(someData, aes(x=a, y=b)) + geom_line()
```




### 9.7 And plotly works too

Note the silent output – we don't need the result, so we just don't print it.

```
library(plotly)
library(htmlwidgets)
data(diamonds, package = "ggplot2")
ggp <- ggplot(diamonds, aes(x = log(carat), y = log(price))) + geom_hex(bins = 100)
plp <- ggplotly(ggp)
saveWidget(plp, "examplePlotly.html", selfcontained=FALSE, libdir="examplePlotlyLib")
```

## 10 Reproduciblity

This section is here to help anyone wishing to reproduce the results above, or to understand the mechanics of how the results were obtained..

Reminder: All blocks in the entire tree can be evaluated with C-C C-V C-S

### 10.1 FILES

Documented in this section are (for each file in this archive):

- SHA1
- Output from an 'ls -l' command
- Output from the 'wc' command – byte, word, and line counts

The use cases are two fold:

- Insure that the input data files being used are the same
- Check if reproduced results match

Replace the `‘find ./ -type f‘` with a list of files and/or wildcards to explicitly select the desired files.

```
date
for c in wc 'openssl sha1' 'ls -l' ; do
    echo $c; $c ‘find ./ -type f‘
done
```

## 10.2 ENVIRONMENT

The input files are only part of the reproduciblity equation. It is also important to understand the tools and computational environment used for the original analysis. This section contains various bits of meta-data about the tools and system I used for this analysis.

### 10.2.1 Embedded Ruby Version

```
puts(RUBY_VERSION)
```

### 10.2.2 Embedded Perl Version

```
print $]
```

### 10.2.3 Embedded R Information

#### R version

```
R.version
```

#### Session Information

```
sessionInfo()
```

#### Loaded Package Versions

```
installed.packages()[(loadedNamespaces()),c('Version', 'LibPath')]
```

### 10.2.4 Emacs Information

#### Emacs Version

```
(emacs-version)
```

#### org-mode Version

```
org-version
```

#### ESS Version

```
(ess-version)
```

#### Process Environment

```
process-environment
```

#### System Type

```
system-type
```

#### System Configuration

```
system-configuration
```

### 10.2.5 System Information

```
for e in date whoami groups id hostname domainname dnsdomainname 'ifconfig -a' 'uname -
  c=`echo $e | awk '{print $1}'`;
  if hash $c 1>/dev/null 2>/dev/null; then
    ruby -e 'puts("="*90)'
    echo $e
    sh -c "$e"
  fi
done
```

### 10.2.6 Command Line Tool Information

```
for e in gcc g++ gfortran                                   \
         wc ls grep sed awk cut sort uniq                   \
         bash ksh tcsh dash csh sh zsh                      \
         vi vim emacs em                                    \
         ruby ruby1.8 ruby2 python3 python2 perl            \
         gnuplot maxima octave M2 gap julia R               \
         qtiplot ggobi                                      \
         povray                                             \
         openscad xcircuit                                  \
         convert pqiv import display                        \
         gs pdftex pdflatex tex latex dvips                 \
         sbcl clisp ecl ccl                                 \
         diff diff3 patch merge                             \
         sqlite3 mysqld                                     \
         paraview visit                                     \
         grass                                              \
         tar gzip bzip2 ; do
  ruby -e 'puts("="*90)'
  echo "Tool: $e"
  if hash $e 1>/dev/null 2>/dev/null; then
    CPH=`which $e`
    if [ -n "$CPH" -a -e "$CPH" ] ; then
      echo $CPH    | sed 's/^/  Path: /'
      ls -ld $CPH  | sed 's/^/  ls-l: /'
      $e --version | sed 's/^/  Ver:  /'
    else
      echo "  Unable to locate (which): $e"
    fi
  else
    echo "  Unable to locate (hash): $e"
  fi
done
ruby -e 'puts("="*90)'
```

# 11 Publishing

By "publishing" I mean simply copying stuff from the current directory tree to a new location – usually one shared by a web/file server or to a staging

area to be later uploaded to a web server.

To control very precicely what gets published, put the files in the file `files_to_publish`. One way to do that is like so:

```
EXT2PUB='.org .html .png .gif .jpeg .pdf .ps .sh .rb .R .c .cpp .h .hpp .csv .csv.gz'
if test -e files_to_publish; then cp files_to_publish files_to_publish_before; wc -l fi
for e in $EXT2PUB; do
  find ./ -name "*$e"
done | sed 's/^\.\///' | egrep -v '^(#|\.)' > files_to_publish
sort files_to_publish | uniq > files_to_publish~
mv files_to_publish~ files_to_publish
wc -l files_to_publish

PUB_DIR=/tmp/foo
HTML_NAME=
PUB_MODES=a+rX
VERBOSE=-ii
if test -z "$HTML_NAME" -o 0 -eq `find ./ -cnewer "$HTML_NAME" -a -type f 2>/dev/null
    RSYNC_OPTS='--delete -a'
    if test -n "$VERBOSE";         then RSYNC_OPTS="$RSYNC_OPTS $VERBOSE"; fi
    if test -e '.rsync-filter';    then RSYNC_OPTS="$RSYNC_OPTS -F"; fi
    if test -e 'files_to_publish'; then RSYNC_OPTS="$RSYNC_OPTS --files-from=files_to_p
    date
    rsync $RSYNC_OPTS ./ "$PUB_DIR"
    date
else
  echo "ERROR: $HTML_NAME is not the newest file here.  Please regenerate it (C-c C-e h
fi
```