Introduction to Knitr Round Trip: Phase 1 Summary

Eric Lim

July 30, 2014

1 Introduction

knitr is a wonderful package that enables dynamic report generation with R. It allows us to implement R code into LaTeX, LyX, HTML, Markdown, AsciiDoc, and reStructuredText documents through the concepts of literate programming, which involves interaction between code and documentation for report generation. The main purpose of **knitr** follows an important idea in academic research that the ideal report should include the computational environment used for the research such as code and data which can reproduce results (reproducible research).

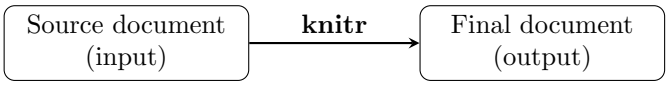


Figure 1: Unidirectional process of generating final documents

While there are tremendously important ideas to consider and many advantages, the process of generating R code embedded documents using **knitr** is almost always a one-way trip (Figure 1), meaning source documents (as input) can only generate final documents (as output), not the other way around. This is simply due to the fact that **knitr** is designed with intention to dynamically generate reports, not to extract displayed R code in final documents to generate source documents. There may be a few limitations, caused by this property, that we will explore further.

Consider a situation where a person who provides documents (using **knitr**) and another who reads or reviews the documents are involved. In this situation, it is often difficult for the document provider to receive feedback from the reviewer efficiently and make appropriate changes. Since it is impossible to go back from final documents to source documents, the reviewer would have to provide his or her feedback by either writing physically on the printed version of the final report or by electronic means such as through e-mails. The document provider, then, has to rectify the source documents accordingly, and repeat the process of generating and presenting the report to the reviewer. This process often has to be repeated until final correction can be achieved. As we can see, it can quickly become tedious.

We believe this is relevant to the field of statistics as similar situations mentioned above can often arise. Interaction between clients and consultants is

fundamentally important for statisticians and any possible factor to deteriorate the relationship with clients is best avoided. Therefore, we want to avoid this issue by a more efficient way to generate final documents.

A possible solution is to allow reviewers to interact directly with the final documents to edit them. Then the document provider can see the changes made by the reviewer and discuss those changes more efficiently. Furthermore, to take reproducible research into account it would be best to find a way to generate a new source document based on the edited final document (so the newly edited final reports can be reproducible). To achieve this solution, there must be a way to enable final documents to be converted to source documents. In other words, the one-way trip from source documents to final documents must incoporate another trip from the final to source documents.

The project we are working on has its main focus on making this ideal round trip possible, and if not what possible issues prevent this from happening. For a start, we deal with simple HTML documents in the hopes of finding a more generalised approach for our goal.

Before proceeding further into more detailed look at the round trip, here is a brief illustration of how an HTML source document in the `.Rhtml` format can produce the final HTML document.

Below is the code structure of the source document, `example1.Rhtml`. We can notice from the lines 9–11 that a line of R code (line 10) is enclosed in a special comment tag.

Listing 1: `example1.Rhtml`

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Example</title>
5    </head>
6    <body>
7      <h1>Example</h1>
8      <p>20 random variates from the standard normal distribution</p>
9      <!--begin.rcode
10     rnorm(20)
11     end.rcode-->
12     </body>
13 </html>
```

The following code calls the function, `knit()`, on the above `.Rhtml` document, and the document is "knitted" to generate the final document, `example1.html`.

```
R> library(knitr)
R> knit("example1.Rhtml")
```

Listing 2 shows the structure of the final document, `example1.html`. The R code chunk has been modified (lines 14–20) to match the HTML syntax and be processed as a part of the HTML document. The original line of R code (from line 10 of Listing 1) is still noticeable (highlighted text in Listing 2). The output from the R code is in lines 16–18 in Listing 2.

Listing 2: `example1.html`

```
<!DOCTYPE html>
<html>
  <head>
<style type="text/css">
.knitr.inline {
    ...about 58 lines of styles defined by knitr...
}
</style>
    <title>Example</title>
  </head>
  <body>
    <h1>Example</h1>
    <p>20 random variates from the standard normal distribution</p>
<div class="chunk" id="unnamed-chunk-1"><div class="rcode"><div class="
    source"><pre class="knitr r"><span class="hl kwd">rnorm</span><span
    class="hl std">(</span><span class="hl num">20</span><span class="hl
     std">)</span>
</pre></div>
<div class="output"><pre class="knitr r">## [1] -0.326450 0.868096
    0.109990 0.555811 0.961545 0.200446 0.352004
##  [8] -0.009584 -0.132941 1.233210 0.656319 1.166483 0.238306 2.137970
## [15] 1.268752 -0.808173 0.207059 -0.734526 0.820859 1.248322
</pre></div>
</div></div>

  </body>
</html>
```

Figure 2 shows what `example1.html` will look like in a browser.

Example

20 random variates from the standard normal distribution

```
rnorm(20)
```

```
##  [1]  1.7067  1.5782  0.3496 -0.9307  2.0008  1.6086 -0.1480  0.4313
##  [9]  0.4988  0.4699 -0.1154  0.1436  1.0866 -0.8908 -2.1270  1.0619
## [17]  2.1381 -0.7105 -0.7915  0.5133
```

Figure 2: `example1.html` in a browser

2 Overview

There are six main steps in a complete cycle of the round trip. Each step generally involves modification of the document to prepare it for a specific goal. Ultimately, our ideology is that the reviewer edits directly on the `.edit.html` document prepared by the document provider and saves the changes. Then the document provider can re-generate the source document with appropriate changes which can be knitted again to produce another `.edit.html` document for extra editing. The round trip can be a repeated process to allow for additional editing if required.

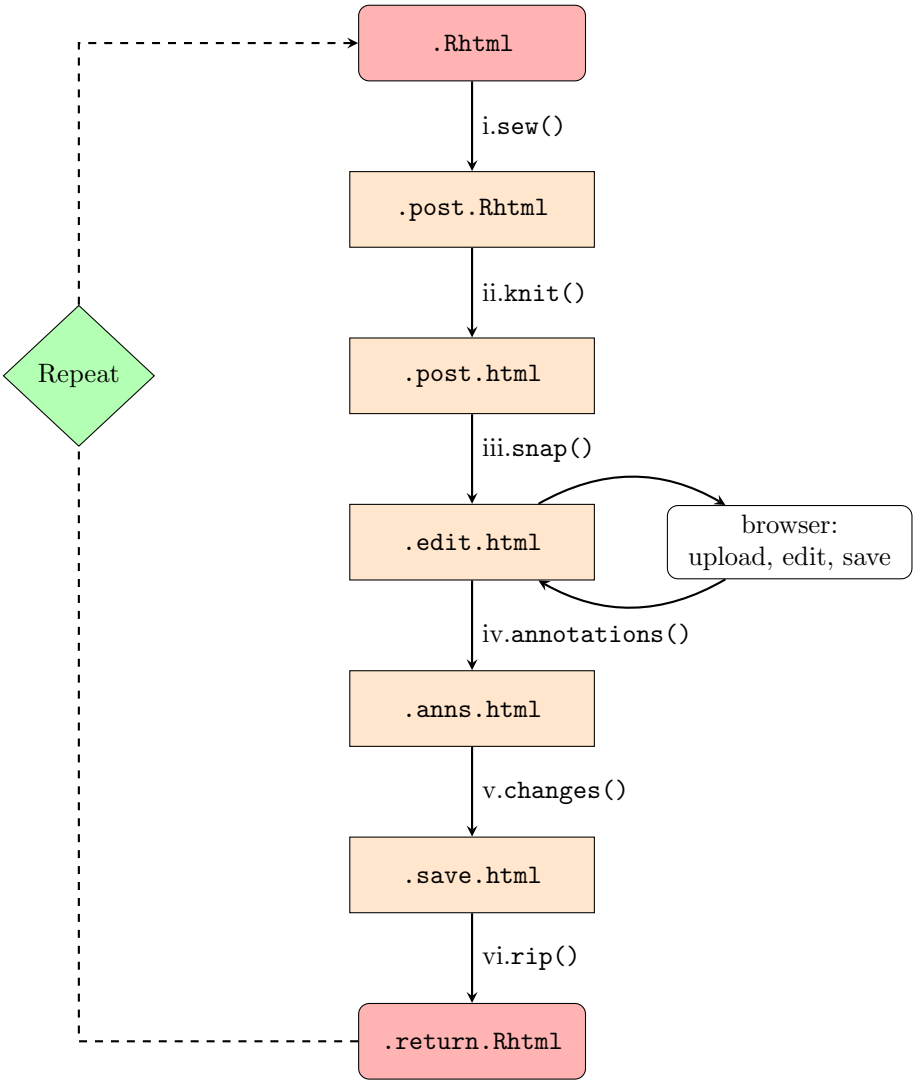


Figure 3: Steps involved in the round trip. The source document (`.Rhtml`) goes through each step until it reaches the final destination (`.return.Rhtml`). The final `.return.Rhtml` file can be fed through the process again if additional changes are required.

The flow chart in Figure 3 outlines a series of steps involved in our hypothesised round trip. A brief description for each step is as follows:

i The function `sew()` is called on the source document (`.Rhtml`) to generate a `.post.Rhtml` file.

Each R code chunk in the source document is copied with a slight change to preserve each of the original R code chunks. The goal is to compensate for the fact that the original R code is heavily modified by **knitr** as the final document is generated (refer back to Listing 2 as a reminder). We must preserve the original R code so that it can be "reproduced" or reappear in the new source document generated from the round trip.

ii The `.post.Rhtml` document is knitted to produce a `.post.html` document which retains the copies of the original R code chunks.

Note that these copies are not displayed as they are hidden inside HTML comment tags while the information of the original R code is retained in the subsequent steps. The copies will not be interfered until the very last step of the round trip.

iii The function `snap()` is called on the `.post.html` document to produce a `.edit.html` document.

The main task carried out by `snap()` is to append a few supplementary lines of code in the `.post.html` document to enable the features of text editing and annotating. The sections of the document relating to R such as R code chunks and any output from R are restricted for annotations only while the rest of the document is entirely editable with a few possible exceptions (the reason behind this decision will be discussed in more detail later).

Once the output document (`.edit.html`) is generated (by `snap()`), it needs to be uploaded onto a server through which editing, annotating and saving the changes made are fully functional. The changes are saved as text files.

iv The saved text file (from the previous step) contains information on any annotations made. The function `annotations()` can access this information to make appropriate changes on the `.edit.html` document to produce a `.anns.html` document.

In the output document (`.anns.html`), a message is inserted at the top of each annotated R code chunk to display *which* text is annotated with *what* annotations by *who*.

v Similarly to the previous annotations step, any changes made on the `.edit.html` document are accessed through the saved text file.

The old, unwanted lines of text from the input document `.anns.html` are replaced with the corresponding new lines of text to generate a `.save.html` document.

vi The function `rip()` is called on the `.save.html` document to produce a `.return.Rhtml` document.

The main goal of `rip()` is to "reproduce" a new source document with the changes and annotations made the previous steps. It essentially aims to convert the `.save.html` document to match the format of the source

document (`.Rhtml`). Any additional lines of code inserted during the round trip and the lines added by **knitr** (such as the lines of code for the style defined by **knitr**) are removed as the main algorithm for the reproducibility.

Important assumptions and more detailed discussion of these functions involved in the round trip will be introduced in Section 3.

3 Functions

There have been various ideas to support the logic behind the algorithms involved in the steps of the round trip. It is absolutely vital to explore deeper into each function for us to gain better understanding. The main purpose of this section is to simply discuss the role of each function in more detail in order to obtain adequate understanding of the overall project and hopefully answer any questions and uncertainties raised in the previous section.

3.1 `sew()`

One of the most important ideas to constantly remind ourselves was the preservation of the original R code. As briefly mentioned in Section 2, retaining the exactly identical R code (that we started with) is a fundamentally important aspect of the round trip in order to reproduce the identical source document. We have figured that the task of preserving the original R code would be much more difficult once the code is converted by the function `knit()`, as it would then require an extra step of text-processing on the converted R code. Gathering the original information from the converted format is unnecessary complication that we can avoid, which led us to a conclusion that the preservation should happen prior to the conversion step of `knit()`.

A possible solution is to formulate a similar strategy that the package **knitr** uses. As a reminder we can refer back to Listing 1 and note the way R code is presented. The original line of R code is wrapped inside slightly modified HTML comment tags (lines 9 and 11) which act as a marker to signal that the contents in these comments are intended to be displayed in the final document. In other words, the function `knit()` recognises these specific types of HTML comments and convert their contents (lines of R code) appropriately for display. The package requires the user to enclose all R code chunks intended for display in final documents in this special manner.

We have decided to implement a similar algorithm involving HTML comment tags through which the retention of the original R code is possible. The only conceptual difference between the two algorithms is in their objectives: detection of R code chunks for conversion in **knitr** and for preservation in ours. The algorithm we have formulated is to copy the original chunks of R code and insert the copies below the originals. In addition, the HTML comment tags to contain the copies are further modified so that they are explicitly controllable in later steps while avoiding detection from `knit()`. These copies remain technically as comments in the HTML syntax which can only be examined through the internal code structure of the document (hidden from display through a browser). As a result, the copies of R code chunks (generated by `sew()`) are preserved in all steps unless we decide to control them to undergo modification.

In summary, our function `sew()`:

- detects any R code chunks that obey the required syntax of **knitr**, that is the chunks intended to be displayed in final documents,
- creates identical copies of those (original) R code chunks with altered the comment tags and
- inserts the copies below the corresponding R code chunks.

The following command in R calls `sew()` on `example1.Rhml` to generate `example1.post.Rhtml`.

```
R> source("sew.R")
R> sew("example1.Rhtml")
```

Listing 3 shows the structure of the output document `example1.post.Rhtml` from `sew()`. Note that the code structures of the input document `example1.Rhtml` (Listing 1) and the output document `example1.post.Rhtml` (Listing 3) are identical except the highlighted text which represents the copy of the original R code (lines 9–11).

Listing 3: `example1.post.Rhml`

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Example</title>
5    </head>
6    <body>
7      <h1>Example</h1>
8      <p>20 random variates from the standard normal distribution</p>
9      <!--begin.rcode
10     rnorm(20)
11     end.rcode-->
12     <!--begin.keepcode
13     rnorm(20)
14     end.keepcode-->
15   </body>
16 </html>
```

3.1.1 `knit()`

The output document `example1.post.Rhtml` from the function `sew()` is ready to undergo the knitting procedure in a sense that the resultant document `example1.post.html` (from `knit()`) will always retain the original R code.

The following code is executed in R to knit the input document `example1.post.Rhtml`...

```
R> knit("example1.post.Rhtml")
```

...to generate the output document `example1.post.html` whose internal code structure can be examined in Listing 4. Note that the highlighted lines represent that the copy of the original R code chunk which remains untouched by the knitting procedure.

Listing 4: `example1.post.Rhml`

```
<!DOCTYPE html>
<html>
  <head>
    ...lines of style from knitr
    <title>Example</title>
  </head>
  <body>
    <h1>Example</h1>
    <p>20 random variates from the standard normal distribution</p>
<div class="chunk" id="unnamed-chunk-1"><div class="rcode"><div class="
    source"><pre class="knitr r"><span class="hl kwd">rnorm</span><span
    class="hl std">(</span><span class="hl num">20</span><span class="hl
     std">)</span>
</pre></div>
<div class="output"><pre class="knitr r">## [1] 0.21997 -1.02766 1.22804
    0.78368 -0.09822 -1.41733 0.25276
##  [8] -0.89354 0.01876 2.55360 -0.46999 1.14496 0.57621 -0.12584
## [15] -0.40660 -1.74065 0.27885 0.90726 -0.97394 0.62497
</pre></div>
</div></div>

    <!--begin.keepcode
    rnorm(20)
    end.rcode-->
  </body>
</html>
```

Figure 4 shows how the document `example1.post.html` looks like in a browser. We should notice that the copy of the origiinal R code is hidden from the display and the only differences in the browser-display are the random variates.

Example

20 random variates from the standard normal distribution

```
rnorm(20)
```

```
##  [1]  0.21997 -1.02766  1.22804  0.78368 -0.09822 -1.41733  0.25276
##  [8] -0.89354  0.01876  2.55360 -0.46999  1.14496  0.57621 -0.12584
## [15] -0.40660 -1.74065  0.27885  0.90726 -0.97394  0.62497
```

Figure 4: `example1.post.html` in a browser

3.2 `snap()`

So far, we have managed to come up with a way to execute the function `knit()` on source documents (`.Rhtml`) while preserving the original information of R code chunks. The next stage of the round trip reflects on our primary objective (of the project), that is to be able to directly edit HTML documents that `knit()` produces. Possible complications involving the restriction due to **knitr**'s unidirectional document generation (Section 1) have led us to a decision to implement the use of annotations. Especially the sections of R-related contents in final documentation are considered pointless to be directly edited as it will only result in text-based modification and will not have any impact on generating new results from the modified code. To gain the conceptual benefits of editing as discussed in Section 1, these sections of R-associated contents are decided to be annotated to effectively deliver necessary messages from a person (editor) to another (viewer).

The first task of the function `snap()` is designed to identify the parts of an input document that should be editable. The editable sections are identified by searching for all top level elements inside the HTML body of the input document that are not introduced by the function `knit()`. Then two special attributes are inserted into the opening tag of each of these identified, editable elements: one to serve as a marker to let us know the corresponding element is editable and the other to "number" the editable elements so that we can deal with them in an orderly manner in later steps. The top elements of annotatable sections (the R-associated contents) are identified by their distict attributes defined by **knitr**. After correctly identifying and marking up all the editable elements, the input document is appended with two supplementary sections of code, namely `button.html` and `edit.js`. The HTML code piece `button.html` contains definition of the designs for two HTML buttons called `save` and `submit` while the JavaScript code piece `edit.js` includes a few assigned instructions. Firstly, `edit.js` loads the **jQuery** library and two JavaScript modules named **CKEditor** and **AnnotateIt** whose respective functions are to enable browser-based editing and annotating on HTML documentation. Secondly, `edit.js` is instructed with the actions for the `save` button, which is to save any changes and annotations made via **CKEditor** and **AnnotateIt** as separate text files on a server, and for the `submit` button, which is to re-direct the browser to an uploading web page on the server (which will be discussed shortly) as well as saving the changes and annotations.

Once the input document is merged with the contents of `button.html` and `edit.js`, an `.edit.html` document is generated as the output of the function `snap()`. The resulting `.edit.html` document is then uploaded onto our test server through which the actual editing and annotating of the document occur. If we refer back to Listing 4, the lines 10, 11 and 12 are the three top level elements inside the body of the HTML document. By our definition, the first two elements (lines 10 and 11), which correspond respectively to a (most important) heading and a paragraph, are editable and the third element (line 12), characterised by the attribute `class="chunk"` as an element added by **knitr**, is annotatable.

The following code is used in R to call the function `snap()` on the input document `example1.post.html` to generate the output document `example1.edit.html`.

```
R> source("snap.R")
R> snap("example1.post.html")
```

Listing 5 exhibits the internal code structure of the output document. The highlighted text (lines 10 and 11) shows the attributes added by the function `snap()`. Note that the first attributes `contenteditable="true"` are used to let `snap()` recognise the corresponding elements as editable and the second `id="Editor"` attributes are used to mark up the elements in a sequential manner.

Listing 5: `example1.edit.html`

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      ...lines of edit.js...
5      ...lines of style from knitr...
6      <title>Example</title>
7    </head>
8    <body>
9      ...lines of button.html...
10     <h1 contenteditable="true" id="Editor-1">Example</h1>
11     <p contenteditable="true" id="Editor-2">20 random variates from the
           standard normal distribution</p>
12 <div class="chunk" id="unnamed-chunk-1"><div class="rcode"><div class="
       source"><pre class="knitr r"><span class="hl kwd">rnorm</span><span
       class="hl std">(</span><span class="hl num">20</span><span class="hl
        std">)</span>
13 </pre></div>
14 <div class="output"><pre class="knitr r">## [1] 0.21997 -1.02766 1.22804
       0.78368 -0.09822 -1.41733 0.25276
15 ##  [8] -0.89354 0.01876 2.55360 -0.46999 1.14496 0.57621 -0.12584
16 ## [15] -0.40660 -1.74065 0.27885 0.90726 -0.97394 0.62497
17 </pre></div>
18 </div></div>
19
20     <!--begin.keepcode
21     rnorm(20)
22     end.keepcode-->
23   </body>
24 </html>
```

The `.edit.html` document is then uploaded on to the test server. The browser-display of the document, on the server, can be seen in Figure 5 where the top and bottom displays represent the editable and annotatable environments of **CKEditor** and **AnnotateIt** respectively. Editing occurs by each element, that is each section (of the element) must be clicked separately to bring up the editable environment. Attaching annotations requires the user to have an account with **AnnotateIt** and be logged into their website `http://annotateit.org/`. In addition, annotations are made on selections of text of the annotatable sections. Notice the `save` and `submit` buttons at the top of the display which can be clicked to save the changes and annotations made. It is of great importance to keep in mind that the execution of editing and annotating takes place simul-

taneously on the uploaded document to create two separate save files with the information on the annotations and the text-based changes when the `save` or `submit` button is clicked.

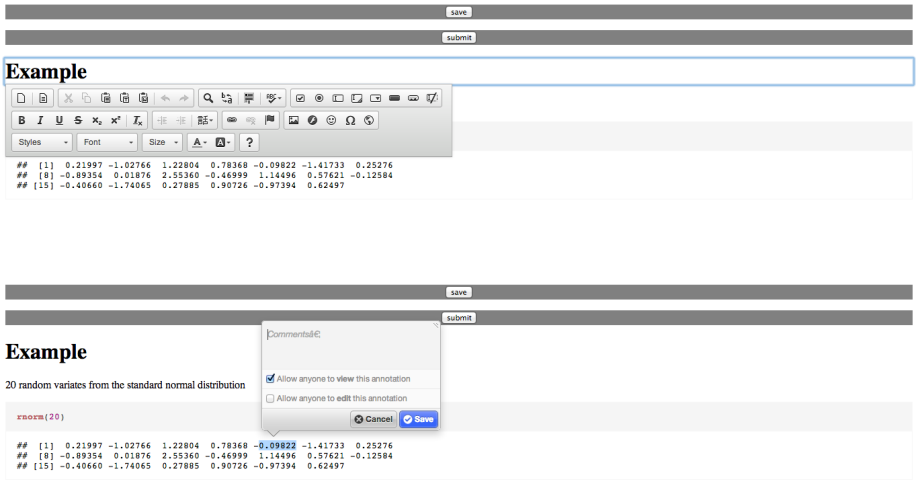


Figure 5: `example1.edit.html`, uploaded on the server, in a browser

In summary, the function `snap()`:

- identifies and marks up all editable top level elements, present in input documents, in a sequential manner,
- appends the two supplementary sections of code `buttons.html` and `edit.js` into the input documents whose respective purposes are:
  - i the provision of the designs for the two `save` and `submit` buttons and
  - ii to load the **jQuery** library, enable **CKEditor** and **AnnotateIt** on editable and annotatable sections of the input documents respectively, and save the changes and annotations made through the two JavaScript modules as text files on the test server when the buttons are clicked, with an additional feature of re-directing the browser to the uploading web page when the `submit` button is clicked, and
- finally generate `.edit.html` documents as the output which can then be uploaded onto the server for the actual editing and annotating of the documents to take place.

3.3 `annotations()`

The information on annotations made via the JavaScript module `AnnotateIt` is accessed from the test server as a text file. The module generates the save file in the JSON format for which we have decided to use an R package called **jsonlite** to effectively deal with the file format. The save file can be directly fetched from the server by using another R package **RCurl** and can then be processed, in the manner regarding the JSON format, to generate a message for each annotation. The most informative data such as the selection of text for

annotations, the author of the annotations and the annotations themselves are extracted from the save file in order to generate the message.

Figure 6 is a minimal demonstration of creating an annotation. The last random variate is selected to be marked with the annotation "Example". Upon clicking the blue `Save` button on the pop-up window, a save file is created by **AnnotateIt** on the test server.

Figure 6: annotating on `example1.edit.html`

The following code is used in R to call the function `annotations()` on the input document `example1.edit.html` to generate the output document `example1.anns.html`.

```
R> source("annotations.R")
R> annotations("example1.edit.html")
```

Listing 6 is the internal code structure of the output document `example1.anns.html` where the highlighted lines of text (lines 12–14) represent the message genereated according to the information from the save file. The message is in a paragraph tag with the `class="annotation"` and its own style attributes.

Listing 6: `example1.anns.html`

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      ...lines of edit.js...
5      ...lines of style from knitr...
6      <title>Example</title>
7    </head>
8    <body>
9      ...lines of button.html
10     <h1 contenteditable="true" id="Editor-1">Example</h1>
11     <p contenteditable="true" id="Editor-2">20 random variates from the
           standard normal distribution</p>
12 <p class="annotation" style = "background-color:coral">
13 The text "-1.88681" was annotated with the message "Example" by "e.lim0322"
14 </p>
15 <div class="chunk" id="unnamed-chunk-1"><div class="rcode"><div class="
       source"><pre class="knitr r"><span class="hl kwd">rnorm</span><span
       class="hl std">(</span><span class="hl num">20</span><span class="hl
        std">)</span>
16 </pre></div>
17 <div class="output"><pre class="knitr r">## [1] -0.21445 -0.03694 1.15209
        -0.75281 0.66412 -1.14614 -0.98223
```

```
## [8] -1.10767 -1.91184 2.00111 0.66261 -1.00241 -0.41161 -0.07720
## [15] -1.20503 -1.77474 -1.05262 -3.28418 0.76983 -1.88681
</pre></div>
</div></div>

    <!--begin.keepcode
    rnorm(20)
    end.keepcode-->
  </body>
</html>
```

Figure 7 shows the document `example1.anns.html` in a browser. The text highlighted in red is the message.

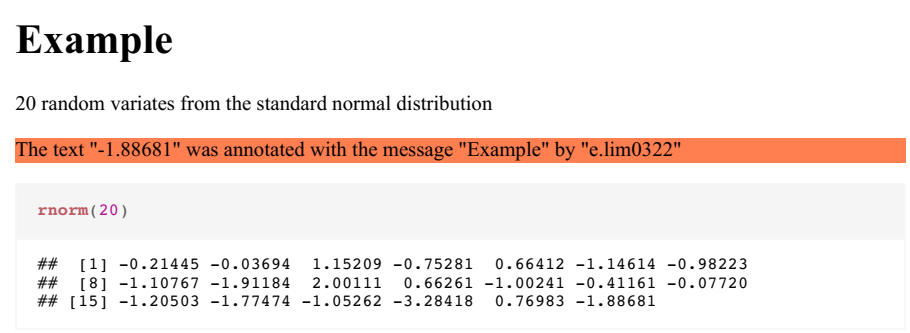


Figure 7: `example1.anns.html` in a browser

3.4 `changes()`

The primary task assigned to the function `changes()` is to access the save file for the changes made through **CKEditor** from the server and replace the old, modified text (in the input document) with the new text (in the save file). This replacement takes place only for the sections that are actually edited.

Figure 8 illustrates a brief demonstration of how the process of editing is carried out. The first editable section, which is the heading, is left as it is while the second editable section is chosen to be edited. The original line of text "20 random variates from..." is to be edited with the bold faced text "20 normal random variates". When the grey `save` button is clicked, the save file of the change is created on the server which is a simple text file with the code structure as shown in Listing 7.

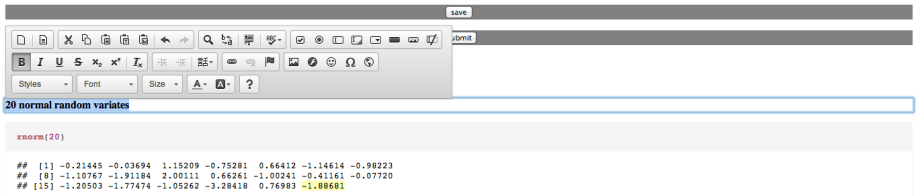


Figure 8: editing `example1.edit.html`

Note the first line in Listing 7. This line corresponds to the editable element (in `example1.edit.html`) with an attribute defined as `Editor-1`, that is the line 10 in Listing 5, which is the heading. As we can see the heading is "`NOT MODIFIED`" to allow the function `changes()` to ignore this editable section of the document. On the other hand, the line 4 displays the desired change on the second editable section.

Listing 7: `changes.txt`

```
1 EDITOR Editor-1 NOT MODIFIED
2
3 EDITOR Editor-2:
4 <strong>20 normal random variates</strong>
```

The following code is used in R to call the function `changes()` on the input document `example1.anns.html` to generate the output document `example1.save.html`.

```
R> source("changes.R")
R> changes("example1.anns.html")
```

The internal code structure of the output document `example1.save.html` can be seen in Listing 8. The highlighted text (line 12) shows that the old line (line 11 in Listing 6) is replaced with the change (line 4 in Listing 7) by `snap()` to result in a display through a browser as shown in Figure 9.

Listing 8: `example1.save.html`

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      ...lines of edit.js...
5      ...lines of style from knitr...
6      <title>Example</title>
7    </head>
8    <body>
9      ...lines of button.html
10     <h1 contenteditable="true" id="Editor-1">Example</h1>
11     <p contenteditable="true" id="Editor-2">
12 <strong>20 normal random variates</strong>
13     </p>
14     <p class="annotation" style = "background-color:coral">
15     The text "-1.88681" was annotated with the message "Example" by "e.
           lim0322"
16     </p>
17 <div class="chunk" id="unnamed-chunk-1"><div class="rcode"><div class="
       source"><pre class="knitr r"><span class="hl kwd">rnorm</span><span
       class="hl std">(</span><span class="hl num">20</span><span class="hl
        std">)</span>
18 </pre></div>
19 <div class="output"><pre class="knitr r">## [1] -0.21445 -0.03694 1.15209
        -0.75281 0.66412 -1.14614 -0.98223
20 ##  [8] -1.10767 -1.91184 2.00111 0.66261 -1.00241 -0.41161 -0.07720
21 ## [15] -1.20503 -1.77474 -1.05262 -3.28418 0.76983 -1.88681
22 </pre></div>
23 </div></div>
```

```
24
25      <!--begin.keepcode
26      rnorm(20)
27      end.keepcode-->
28    </body>
29  </html>
```

The text is successfully edited in Figure 9.

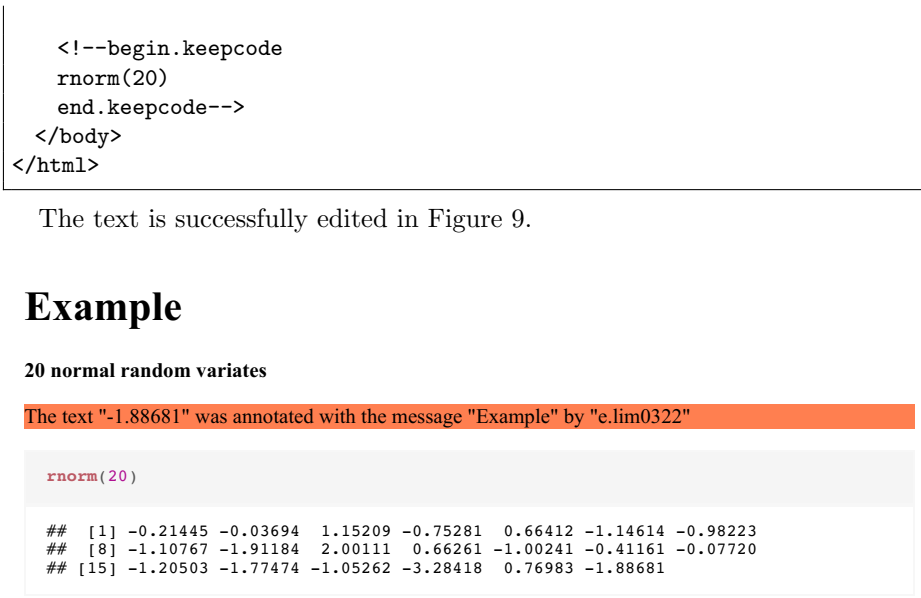


Figure 9: `example1.save.html` in a browser

3.5 `rip()`

The main purpose of the function `rip()` is to return to the original source document format while retaining changes and annotations. It searches for any foreign lines of code that are introduced by **knitr** or inserted during certain stages of the round trip, and removes those lines. Typically, the lines of the CSS style inserted by **knitr** are removed first, followed by the lines corresponding to the `div` elements generated and inserted by **knitr** (e.g. lines 17–23 in Listing 8). These `div` elements are defined with the attribute `class="chunk"` which can be identified by `snap()` to recognise them as the elements introduced by **knitr**. Then the copies of the original R code chunks (lines 12–14 in Listing 3) are modified by `snap()` to be exactly as the originals. The text "`keep`" in the comment tags of the copy (lines 12 and 14 in Listing 3) are identified and converted to the letter "`r`" so that both the opening and closing comment tags are like the originals. The lines of `button.html` and `edit.js` are also removed as well as the two attributes `contenteditable="true"` and `id="Editor"`, added to mark up the corresponding elements as editable (Section 3.2).

The following R code calls the function `rip()` on the input document `example1.save.html` to generate the output document `example1.return.Rhtml`.

```
R> source("rip.R")
R> rip("example1.save.html")
```

The internal code structure of the output document `example1.return.Rhtml` can be seen in Listing 9. If we compare the code structure of the source document `example1.Rhtml` with that of `example1.return.Rhtml`, we can notice that line 8 in Listing 1 (original) is replaced by the edited text in line 9 of Listing 9. Lines 11–13 in Listing 9 are the generated messages for the annotations

from the previous step (Section 3.3). Apart from these lines, the rest of both documents are identical.

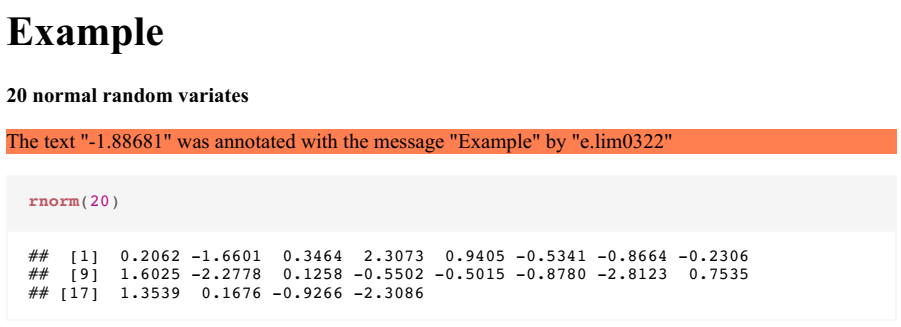
Listing 9: `example1.save.html`

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <h1>Example</h1>
    <p>
<strong>20 normal random variates</strong>
    </p>
    <p class="annotation" style = "background-color:coral">
    The text "-1.88681" was annotated with the message "Example" by "e.
        lim0322"
    </p>
    <!--begin.rcode
    rnorm(20)
    end.keepcode-->
  </body>
</html>
```

We can knit the "final" source document `example1.return.Rhtml` by the following code in R.

```
R> knit("example1.return.Rhtml")
```

The display of the knitted document `example1.return.html` through a browser can be seen in Figure 10.



Example

20 normal random variates

The text "-1.88681" was annotated with the message "Example" by "e.lim0322"

```
rnorm(20)
```

```
##  [1]  0.2062 -1.6601  0.3464  2.3073  0.9405 -0.5341 -0.8664 -0.2306
##  [9]  1.6025 -2.2778  0.1258 -0.5502 -0.5015 -0.8780 -2.8123  0.7535
## [17]  1.3539  0.1676 -0.9266 -2.3086
```

Figure 10: `example1.return.html` in a browser

4 Discussion

There are a few restrictions in the round trip that need to be discussed. Some of the steps involved in the round trip base their algorithms on the assumption that

knitr remains consistent. For example, identifying the annotatable elements (Section 3.2) is carried out under the assumption that these elements are always given the attribute `class="chunk"` by **knitr**. If there is to be a change in **knitr** that affects the structure of the annotatable elements, especially their attributes, the round trip will fail to complete the cycle and we will have to devise a new measure for the change.

The editing and annotating stage of the round trip is limited to what **CKEditor** and **AnnotateIT** offer as the necessary tools. The requirement by **AnnotateIT** for its user to maintain a logged-in status with their website is a restriction that can be avoided altogether by implementing a different annotator module. The overall process of editing and annotating can be more efficient and flexible to allow for more extensive document editing by the use of modules that have relatively less restrictions than **CKEditor** and **AnnotateIT**.

Furthermore the efficiency of the round trip may improve by the use of a different HTML editor. One of the main reasons behind the uploading procedure is to enable **CKEditor** without having to localise the module in the user's computing environment. If it is possible to integrate an HTML editor, into the round trip, that does not pose the localisation problem the uploading step can be omitted to make more efficient and tidier document generation and editing through the round trip possible.

By using a different HTML editor, we can also reduce some of the problems arising from the uploading stage of the round trip. Currently it is not possible to upload external files such as image files that are produced by R on the test server. As a result, any graphics included in the uploaded documents cannot be displayed through a browser. Again this issue can be avoided by using suitable HTML editors that do not require the server integration. A possible future direction of the project in regards to these limitations is in generalising the round trip so that the user gets to choose which HTML editors to use for the round trip.

There is a possibility of an alternative approach to the round trip. This would be to integrate a wiki or an online document system (e.g. Google Docs) as the main editing tool. These tools may improve the efficiency of the overall editing process through more dynamic and responsive process of document editing.

Finally, we need to consider how the sections of images, mathematical equations and general tables could be edited in a document. A possible approach to the problem is to allow annotations on them as we cannot directly interfere with the contents of these sections. A better approach would be to utilise a suitable JavaScript module written for the purpose of editing or annotating on these specific sections.

5 Conclusion

Our ultimate goal remains to be in further generalising the round trip to suit a broader spectrum of document types. The first phase of the project has explored the possibilities of a complete round trip between the source and final documents to enhance the flow of the document generation process. So far the phase 1 round trip is strictly limited to the HTML type of documentation. However, we hope that our suggested ideas involved with each step of the round trip can be used by others to further optimise the procedures invovled in statistical document

generation.

6 Contributions

List of contributions from the project supervisor Dr Paul Murrell:

- ideas
- `edit.js`
- `button.html`