R input and output for authoring statistics problems in LON CAPA

Marianne Huebner and Stuart Raeburn, Michigan State University

17 February 2015

LON CAPA is a web based software for class management, exam or homework questions that can be taken online or printed for a paper test [1]. Questions for these tests can be created in the authoring space. Questions are written in PERL but can be interfaced with R. This comes in very handy for statistics questions.

The tutorial *Authoring Statistics Problems in LON-CAPA using R* by M. Huebner and S. Raeburn [2] provides examples of statistics and probability problems in LON CAPA including data description, probability, inference, and regression. Some of the questions require importing a data vector into R, reading a simulated random sequence from R, or identifying quantities from output of R functions. The purpose of this document is to explain steps involved for importing data into R and extracting information from R output.

A LON CAPA problem is structured with a script for the coding of the problem, a text portion where questions can be asked and a response field. Data transfer from R to Perl and vice versa is coded in the `script` portion, and we consider how to write the output in the text portion of the problem.

```
<problem>
<script type="loncapa/perl">
# In the script portion variables are defined and answers are
calculated
</script>

<startouttext/>
# In the text portion the problem is described.
<endouttext/>
```

The LON CAPA script for problems is written in Perl. However it interfaces with R using `&cas`, if the output is a scalar, and `&cas_hashref`, if the output is a vector, matrix, or a list.

Reading scalars from R output

Since the R output is a scalar, `&cas` is used here.

1. After importing values of variables

A variable that has been assigned a value in Perl can be read into R, and operations can be performed within R.

```
$a=1;
$b=2;
$result = &cas("R", "$a+$b");
```

Only the last output of the R operations is saved. In the example below, only the value of y is saved as the `$result`.

```
$a=1;
$b=2;
$result = &cas("R", "x<-$a*$b;  y<-$a+$b");
```

2. After importing data lists

A data list defined in Perl can be read into R for further calculations. In the code below values of a random variable X and the corresponding probabilities $P(X = x)$ are read into R and the expectation (scalar product) is calculated.

```
$x = "1,2,3,4,5";
$probdist = "0.1,0.2,0.3,0.05,0.35";
$expectation = &cas("R", "p<-c($probdist); x<-c($x), x%*%p");
```

Alternatively, a list of data can be read into R from an array.

```
@x = (1,2,3,4,5);
$x_str = join(',', @x);
$mu = &cas("R", "x<-c($x); mean(x)");
```

In summary

```
$result = &cas("R", "R code goes here. Last number is saved as
output.");
```

Reading arrays from R output

1. Simulate a random sequence in R to use in statistics problems

When sampling from probability distributions in R, it is important to set the seed. This will make sure that students have the same problem with each try. If the random seed is not fixed, a new randomization is generated each time the student submits an answer.

```
# set seed
$seed = &random(1,200000000,1);
```

```
# Generate a value for a mean and standard deviation, sampled from a
uniform distribution in Perl
$mu = &random (2,7, 1);
$sd = &random (0.1,0.5, 0.05);
```

The Perl function `&random` has the lower bound as the first argument, the upper bound as a second argument, and the step size as the third argument. For example, `$sd = &random (0.1,0.5, 0.01);` will be uniformly distributed in the interval `[0.1,0.5]` and values `0.1, 0.15, 0.2, 0.25, ...,0.45, 0.5` can be chosen as the value for `$sd`.

Next we simulate a random sequence from the normal distribution with the specified mean and standard deviation. Note that `$dump` is a version of `$data`, which can be used for debugging.

```
# length of sequence
$n = 10;
# generate uniformly distributed random numbers from Uniform(25,100).
($data,$dump)= &cas_hashref("R", "set.seed($seed);rnorm($n,mean=$mean,
sd=$sd);");
```

2. Indexing the array to use in the text of the statistics problem

Since To be able to write this sequence into the text of a problem, it needs to be converted to and array in Perl.

```
@sample = &cas_hashref_array($data);
```

In the problem test or in html tables this list of 10 numbers can be accessed by writing

```
$sample[0], $sample[1], $sample[2], $sample[3], $sample[4], $sample[5],
$sample[6], $sample[7], $sample[8], $sample[9]
```

where the index 0 extracts the first element of the list.

3. Use the random sequence as R input

In order to use this dataset in R for further operations or to write this list of data without referencing each one, the Perl array needs to be converted into a string of comma separated values.

```
$samplevalues = join(',',@sample);
```

This dataset can be written in the text as a list of numbers without further indexing, namely `$samplevalues`.

```
37.43,56.62,19.41,34.57,-
1.05,18.96,6.8,3.71,29.06,30.94,43.42,16.9,43.81,23.69,7.89,30.02
```

This array can now be used further in R, for example, to calculate the sample mean.

```
$xbar = &cas("R","x<-c($samplevalues); mean(x)");
```

4. Vector calculations in R when the output is also a vector

For results of calculations in R with two data lists, we also need the sequence of `@cas_hashref`, `&cas_hashref_array`, and possibly `join`. For example, taking the element-wise difference of the vector elements:

```
($datadiff,$dumpdiff)= &cas_hashref("R","a<-c($list1); b<-c($list2); a-
b");
@diff = &cas_hashref_array($datadiff);
$difflist = join(',',@diff);
```

In summary three lines of code are required to work with array output from R.

```
($data,$dump)= &cas_hashref("R", "R code for vector output");
@x = &cas_hashref_array($data);
$xvalues = join(',',@x);
```

Reading output from R procedures

Output from R procedures is saved as `$data` and as `$dump`. The `$dump` list is not further used, but can be displayed to identify the structure of the output or for debugging.

For example, we calculate numerical summaries in R and read the output. The order of the data summary is minimum, first quartile, median, third quartile, and maximum, e.g for a list of numbers sampled from a normal distribution.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.750   4.920   5.085   5.053   5.158   5.450
```

Using a previously generated random sequence in LON CAPA, `$samplevalues`, we calculate the summary statistics below.

```
($summary,$summarydump)= &cas_hashref("R","a<-c($samplevalues);
summary(a)");
```

Writing `$summarydump` in the text results in the following output:

```
$VAR1 = { '1' => '3.83', '2' => '4.752', '3' => '5.735', '4' =>
'5.423', '5' => '6.012', '6' => '6.71' };
```

The minimum and first quartile are extracted as follows:

```
@numbers = &cas_hashref_array($summary);
$min = $numbers[0];  # this is 3.83 from above
$q1 = $numbers[1];   # this is 4.752 from above
```

However some output from R procedures are lists of named variables, and some of these have length longer than one. One example is confidence intervals and hypothesis test that has been described in the tutorial by M. Huebner and S. Raeburn *Authoring Statistics Problems in LON-CAPA using R*. Below we describe a linear regression example using a built-in R dataset.

In R this looks as follows.

```
library(car)
fit<-lm(prestige~education, data=Prestige)
summary(fit)$coefficients
```

```
##               Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) -10.731982  3.6770883 -2.918609 4.343443e-03
## education     5.360878  0.3319882 16.147796 1.286264e-29
```

In LON CAPA we need to load the library `car` when calculating the regression fit.

```
($regdata, $regdump) = &cas_hashref("R","fit<-lm(prestige~education,
data=Prestige); summary(fit);", "car");
```

The structure of the output is as follows, but due to the length of the output residuals were omitted.

```
$VAR1 = { 'call' => \{ '1' => 'lm', '2' => 'prestige ~ education', '3'
=> 'Prestige' }, 'terms' => \{ '1' => '~', '2' => 'prestige', '3' =>
'education' },  'coefficients' => \{ '1' => \{ '1' => '-10.73198', '2'
=> '3.677088', '3' => '-2.918609', '4' => '0.004343443' }, '2' => \{
'1' => '5.360878', '2' => '0.3319882', '3' => '16.1478', '4' =>
'1.286264e-29' } }, 'aliased' => \{ '1' => 'Unrecognized output',
'Unrecognized output' => 'Unrecognized output' }, 'sigma' =>
'9.103291', 'df' => \{ '1' => '2', '2' => '100', '3' => '2' },
'r.squared' => '0.7228007', 'adj.r.squared' => '0.7200287',
'fstatistic' => \{ 'value' => '260.7513', 'numdf' => '1', 'dendf' =>
'100' }, 'cov.unscaled' => \{ '1' => \{ '1' => '0.1631591', '2' => '-
0.01428149' }, '2' => \{ '1' => '-0.01428149', '2' => '0.00132999' } }
};
```

The `$regdump` output is organized as follows:

- formula, response (`prestige`), and predictor (`eduation`)
- residuals for each data point (in this dataset these are names of professions) which is the longest part of the output depending on the sample size. Omitted above.
- coefficients, standard errors, and p values for both intercept and slope
- residual standard error
- degrees of freedom

- coefficient of variation
- F test

In particular, the output for the table of coefficients is as follows:

```
'coefficients' => \{
'1' => \{ '1' => '-10.73198', '2' => '3.677088', '3' => '-2.918609',
'4' => '0.004343443' },
'2' => \{ '1' => '5.360878', '2' => '0.3319882', '3' => '16.1478', '4'
=> '1.286264e-29' }
}
```

The intercept is in the first row ('1' =>\{...}) and the predictor in the second row ('2' =>\{...}) . We can extract the coefficient, standard error, and p value for the predictor `education` by accessing '`coefficients`', then the second row, and lastly choosing estimate ('1'), standard error ('2'), t statistic ('3'), and the p value ('4') for testing whether the slope is 0 or not.

```
$slope=&cas_hashref_entry($regdata, "coefficients", "2", "1");
$slopeSE=&cas_hashref_entry($regdata, "coefficients", "2", "2");
$slopeTstat=&cas_hashref_entry($regdata, "coefficients", "2", "3");
$slopePval=&cas_hashref_entry($regdata, "coefficients", "2", "4");
```

Instead we could read the array for the coefficients for `education` and then extract the slope from this quantity:

```
@education=&cas_hashref_array($regdata, "coefficients", "2");
$slope = $education[0];
```

In summary use the `$dump` portion of the output to identify the structure of the array and identify variable names. This can then be used to extract specific elements of the R output. The structure for generating output from R procedures is

```
($data, $dump) = &cas_hashref("R","code for R procedure;", "package
name");
```

The last part can be omitted if no R package needs to be loaded. Scalar output is read with `&cas_hashref_entry` and arrays with `&cas_hashref_array`.

Conclusion

In this document we describe steps involved for importing data into R and extracting information from R output that can be scalars, arrays, or lists from R procedures.

References

[1] LON CAPA Documentations: http://www.lon-capa.org/documentation.html

[2] M. Huebner and S Raeburn. Authoring Statistics Problems in LON-CAPA using R (2014)
https://loncapa.msu.edu/res/msu/statlib2msu/Documentation/loncapa_statsR.pdf