STAT:5400
Computing in Statistics

`xtable` package in R
Simulation studies

Lecture 8
Sep. 18, 2017

Kate Cowles
374 SH, 335-0727
kate-cowles@uiowa.edu

```
xtable                  package:xtable                 R Documentation

Create Export Tables

Description:

     Function converting an R object to an ’xtable’ object, which can
     then be printed as a LaTeX or HTML table.

Usage:

     xtable(x, caption=NULL, label=NULL, align=NULL, digits=NULL,
            display=NULL, ...)

Arguments:

       x: An R object of class found among ’methods(xtable)’.  See
          below on how to write additional method functions for
          ’xtable’.

 caption: Character vector of length 1 containing the table’s caption
          or title. Set to ’NULL’ to suppress the caption.  Default
          value is ’NULL’.

   label: Character vector of length 1 containing the LaTeX label or
          HTML anchor. Set to ’NULL’ to suppress the label.  Default
          value is ’NULL’.

   align: Character vector of length equal to the number of columns of
          the resulting table indicating the alignment of the
          corresponding columns.  Also, ’"|"’ may be used to produce
          vertical lines between columns in LaTeX tables, but these are
          effectively ignored when considering the required length of
          the supplied vector.  If a character vector of length one is
          supplied, it is split as ’strsplit(align,"")[[1]]’ before
          processing. Since the row names are printed in the first
          column, the length of ’align’ is one greater than ’ncol(x)’
          if ’x’ is a ’data.frame’. Use ’"l"’, ’"r"’, and ’"c"’ to
          denote left, right, and center alignment, respectively.  Use
          ’"p{3cm}"’ etc for a LaTeX column of the specified width. For
          HTML output the ’"p"’ alignment is interpreted as ’"l"’,
          ignoring the width request. Default depends on the class of
          ’x’.
```

```
  digits: Numeric vector of length equal to one (in which case it will
          be replicated as necessary) or to the number of columns of
          the resulting table *or* matrix of the same size as the
          resulting table  indicating the number of digits to display
          in the corresponding columns. Since the row names are printed
          in the first column, the length of the  vector ’digits’ or
          the number of columns of the matrix ’digits’ is one greater
          than ’ncol(x)’ if ’x’ is a ’data.frame’. Default depends of
          class of ’x’. If values of ’digits’ are negative, the
          corresponding values  of ’x’ are displayed in scientific
          format with ’abs(digits)’ digits.

 display: Character vector of length equal to the number of columns of
          the resulting table indicating the format for the
          corresponding columns. Since the row names are printed in the
          first column, the length of ’display’ is one greater than
          ’ncol(x)’ if ’x’ is a ’data.frame’. These values are passed
          to the ’formatC’ function.  Use ’"d"’ (for integers), ’"f"’,
          ’"e"’, ’"E"’, ’"g"’, ’"G"’, ’"fg"’ (for reals), or ’"s"’ (for
          strings). ’"f"’ gives numbers in the usual ’xxx.xxx’ format;
          ’"e"’ and ’"E"’ give ’n.ddde+nn’ or ’n.dddE+nn’ (scientific
          format); ’"g"’ and ’"G"’ put ’x[i]’ into scientific format
          only if it saves space to do so.  ’"fg"’ uses fixed format as
          ’"f"’, but ’digits’ as number of _significant_ digits.  Note
          that this can lead to quite long result strings.  Default
          depends on the class of ’x’.

     ...: Additional arguments.  (Currently ignored.)
```

```
Details:

     This function extracts tabular information from ’x’ and returns an
     object of class ’"xtable"’. The nature of the table generated
     depends on the class of ’x’. For example, ’aov’ objects produce
     ANOVA tables while ’data.frame’ objects produce a table of the
     entire data.frame.  One can optionally provide a caption (called a
     title in HTML) or label (called an anchor in HTML), as well as
     formatting specifications.  Default values for ’align’, ’digits’,
     and ’display’ are class dependent.

     The available method functions for ’xtable’ are given by
     ’methods(xtable)’. Users can extend the list of available classes
     by writing methods for the generic function ’xtable’. These
     methods functions should have ’x’ as their first argument with
     additional arguments to specify ’caption’, ’label’, ’align’,
     ’digits’, and ’display’.  Optionally, other arguments may be
     present to specify how the object ’x’ should be manipulated. All
     method functions should return an object whose class if given by
     ’c("xtable","data.frame")’. The resulting object can have
     attributes ’caption’ and ’label’, but must have attributes
     ’align’, ’digits’, and ’display’.  It is strongly recommened that
     you set these attributes through the  provided replacement
     functions as they perform validity checks.

Value:

     An object of class ’"xtable"’ which inherits the ’data.frame’
     class and contains several additional attributes specifying the
     table formatting options.

Author(s):

     David Dahl dahl@stat.tamu.edu with contributions and suggestions
     from many others (see source code).

See Also:

     ’print.xtable’, ’caption’, ’label’, ’align’, ’digits’, ’display’,
     ’formatC’, ’methods’
```

- very handy if you are producing results in R and writing them up in LaTeX
- can be used with different kinds of objects produced by R, including
  - matrices
  - regression output
- examples with data frame created from `Davis` dataset on course web page
  - when input object is data frame, `xtable` always puts row names in the first column of output table

```
> Davis[1:5,]

  sex weight height repwt repht
1   M     77    182    77   180
2   F     58    161    51   159
3   F     53    161    54   158
4   M     68    177    70   175
5   F     59    157    59   155

> xtDavis <- xtable(Davis[1:5,], align="rcrrrr",
caption='Example of xtable for matrix or data frame')

> xtDavis
% latex table generated in R 2.9.1 by xtable 1.5-4 package
% Fri Nov 20 13:31:00 2009
\begin{table}[ht]
\begin{center}
\begin{tabular}{rcrrrr}
  \hline
 & sex & weight & height & repwt & repht \\
  \hline
1 & M &  77 & 182 &  77 & 180 \\
  2 & F &  58 & 161 &  51 & 159 \\
  3 & F &  53 & 161 &  54 & 158 \\
  4 & M &  68 & 177 &  70 & 175 \\
  5 & F &  59 & 157 &  59 & 155 \\
   \hline
\end{tabular}
\caption{Example of xtable for matrix or data frame}
\end{center}
\end{table}
```

	sex	weight	height	repwt	repht
1	M	77	182	77	180
2	F	58	161	51	159
3	F	53	161	54	158
4	M	68	177	70	175
5	F	59	157	59	155

Table 1: Example of xtable for matrix or data frame

```
> lmout <- lm( weight ~ repwt + sex, data = Davis)
> summary(lmout)

Call:
lm(formula = weight ~ repwt + sex, data = Davis)

Residuals:
     Min       1Q   Median       3Q      Max
 -7.5920  -1.7913  -0.7577   0.6400 108.4106

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.47433    3.78632   0.918    0.360
repwt        0.96634    0.06507  14.850   <2e-16 ***
sexM        -1.48263    1.79775  -0.825    0.411
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 8.426 on 180 degrees of freedom
  (17 observations deleted due to missingness)
Multiple R-squared: 0.6998,     Adjusted R-squared: 0.6964
F-statistic: 209.8 on 2 and 180 DF,  p-value: < 2.2e-16
```

```
> lmout.table <- xtable(lmout, caption='xtable for regression output')
> lmout.table
% latex table generated in R 2.9.1 by xtable 1.5-4 package
% Fri Nov 20 13:34:00 2009
\begin{table}[ht]
\begin{center}
\begin{tabular}{rrrrr}
  \hline
 & Estimate & Std. Error & t value & Pr($>$$|$t$|$) \\
  \hline
(Intercept) & 3.4743 & 3.7863 & 0.92 & 0.3601 \\
  repwt & 0.9663 & 0.0651 & 14.85 & 0.0000 \\
  sexM & -1.4826 & 1.7977 & -0.82 & 0.4106 \\
   \hline
\end{tabular}
\caption{xtable for regression output}
\end{center}
\end{table}
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	3.4743	3.7863	0.92	0.3601
repwt	0.9663	0.0651	14.85	0.0000
sexM	-1.4826	1.7977	-0.82	0.4106

Table 2: xtable for regression output

Statistical inference (review)

- statistical inference: drawing conclusions about an entire *population* based on information contained in a *sample* drawn from the population
  - *parameter*: a numeric quantity that describes an entire population
    - we can never know the exact value of a parameter unless we can measure every member of the population
  - *statistic*: a numeric quantity that describes a sample
  - inference about parameter usually uses the corresponding statistic calculated for a sample drawn from the population of interest

How do we know whether statistical procedures work as advertised??

- e.g. "95% confidence interval" means that applying this procedure will produce an interval that traps the true parameter value in 95% of all possible simple random samples from a population
- e.g. a hypothesis testing procedure claims to give us a significance level of .05 (only .05 risk of rejecting the null hypothesis when it's true)
- since we never know the true values of population parameters, how can we verify that these claims are true?
- can't do it by testing methods on real data
- sometimes we can mathematically prove properties of statistical procedures by using probability
  - often this is possible to do if we can assume that sample sizes are arbitrarily large (asymptotic theory)
  - but we usually need to know how the procedures work for small or moderate sample sizes

Simulation studies

- enable us to test statistical procedures because we create the populations and therefore we know the true parameter values
- simulation studies are experiments
- we decide under what conditions we want to test a procedure
  - what kind of population(s)
    - often characterized by different probability distributions
  - what parameter values
  - what size(s) of samples drawn from those populations
  - perhaps other characteristics that are important to the performance of the particular procedure we are interested in
- then we use a computer to draw many, many simple random samples from the population with the distribution and parameter values we have chosen (this is "simulation")
- we apply the statistical procedure to each of the samples we have simulated

- we can use the results to assess how well the procedure works

Example: procedure to calculate 95% confidence intervals for a population variance

Perhaps we are interested in estimating the variance σ^2 of heights in the population of U.S. undergraduate males.

- we can't possibly measure every single undergraduate male in the U.S., so we plan to take a simple random sample of 35 guys and measure their heights in inches
- we know how to calculate the sample variance s^2 given a set of values
  - s^2 will be our point estimate of σ^2
- we find online a formula for calculating confidence intervals for variances, but it looks wierd and we'd like to know whether it works before applying it to our data `http://sixsigmatutorial.com/Six-Sigma/Six-Sigma-t-Cc` `aspx`

Our plan

- we are interested in testing this confidence intervals procedure in two kinds of populations
  - normal distribution (note: it's actually possible to prove mathetmatically that it works in this case)
  - a skewed population that isn't normal (we'll use the gamma distribution)
- We'll pick values of the population mean μ and the population variance σ^2.
- We'll choose the confidence level that we want to test the procedure for (let's use 95%)
- We'll use the computer (R!) to simulate 1000 samples of size 35 from a normal distribution with μ and σ^2 we chose
- We'll apply the the confidence interval calculating procedure to each of the 1000 samples, producing 1000 confidence intervals

- We'll count how many of these 1000 intervals contain the true value of σ^2, and use this to calculate the proportion

```
number of intervals containing sigma^2
--------------------------------------
total number of datasets
```

  This is our simulation-based estimate of the true "coverage probability" of the procedure. Due to random variability, it probably won't be exactly equal to the confidence level. But if the procedure works well, it should be very close.

- We will repeat all the steps from the preceding page for the non-normal population.

R for the confidence interval procedure for variances

```
# Get the Chisquare values needed to compute confidence intervals

> chisqs <- qchisq(c(0.025, 0.975), 34 )
> chisqs
[1] 19.80625 51.96600

# Simulate 1000 datasets of size 35 from a normal distribution with
#      mean 0 and variance 2
#      Put them in a table; each row is one dataset

> normals <- matrix( rnorm( 35000, mean=0, sd = sqrt(2) ), nrow=1000 )

# Here's what the first row (dataset) looks like:

> normals[1,]
 [1]  1.9676564  3.4152704  1.2508041 -1.1614062  3.2911983  2.4183662
 [7] -1.4385106  0.7644241 -1.5601799  1.5786094  0.5396955 -1.8221252
[13]  2.3019667 -1.6344600  0.5452520  0.7792756  2.6084550  0.1275602
[19] -0.2160686 -2.4250036  0.9353656  0.7725724  2.1570195  0.7922126
[25] -0.5546242  0.5788185 -0.1156963  0.3573210  1.9293221  1.1952435
[31] -0.3926299  0.4034752  1.4604477  0.2986590 -2.4133204

# And here are the sample mean and sample variance of the first dataset:

>mean(normals[1,])
[1] 0.5352847
> var(normals[1,])
[1] 2.322497

# Calculate the sample variance s^2 of each of the 1000 rows;
```

```
# R lets you do this with 1 line of code!

> ssq <- apply( normals, 1, var )

# Now ssq is a vector of 1000 numbers, each the sample variance of one of
#     the datasets

> length(ssq)
[1] 1000

> ssq[1:20]
 [1] 2.3224967 2.6569998 2.7430414 1.3453956 2.7515834 2.7217364 1.4419119
 [8] 2.1723866 1.2222191 0.9203172 1.7110501 1.7675063 1.8160497 1.6341400
[15] 2.6805865 1.9869705 2.1378744 1.9456557 2.0919452 1.0685756
```

```
# Now use the formula to calculate the left endpoint and the right endpoint
#      for a confidence interval based on each dataset

> leftend <- 34 * ssq / chisqs[2]
> rightend <- 34 * ssq / chisqs[1]

# Here are the first 20 confidence intervals:

> cbind( leftend, rightend)[1:10,]
        leftend rightend
 [1,] 1.5195493 3.986867
 [2,] 1.7384059 4.561084
 [3,] 1.7947007 4.708786
 [4,] 0.8802573 2.309546
 [5,] 1.8002895 4.723450
 [6,] 1.7807614 4.672213
 [7,] 0.9434055 2.475229
 [8,] 1.4213361 3.729183
 [9,] 0.7996662 2.098098
[10,] 0.6021396 1.579844


# Now we can use R to determine whether each of the confidence intervals
#     we calculated contains 2
#     that is, is 2 greater than the left endpoint and smaller than right

> contains2 <- 2 > leftend & 2 < rightend
> contains2
   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
  [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
  [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
  [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
  [49]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
.
```

```
.
.
 [949]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [961] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
 [973]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [985]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [997]  TRUE  TRUE  TRUE  TRUE

# convert the logical values TRUE and FALSE to numeric
> contains2 <- as.numeric(contains2)
>  as.numeric(contains2)
   [1] 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1
  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
.
.
.
>  sum( contains2 ) / length(contains2)
[1] 0.951
```

Now let’s do it for a non-normal population

```
# Simulate 1000 datasets of size 35 from a gamma distribution with
#      mean  sqrt(2) and variance 2
#      Put them in a table; each row is one dataset

> gammas <- matrix(  rgamma( 35000, 1, 1/sqrt(2) ), nrow=1000)

# The gamma density with parameters 1 and 1/sqrt(2) is NOT symmetric!
```

```
> ssqgammas <- apply( gammas, 1, var)
> ssqgammas[1:20]
 [1] 1.3470058 1.8985875 2.3153270 5.1593305 1.3587234 1.4856546 1.9841449
 [8] 1.9489576 1.3338598 1.0878279 0.5044098 2.3987975 1.8152692 1.5331041
[15] 1.5600314 2.0573240 1.8465979 0.9101784 3.1174721 1.9953191

> leftendgammas <- 34 * ssqgammas / chisqs[2]
> rightendgammas <- 34 * ssqgammas / chisqs[1]
> cbind( leftendgammas,rightendgammas)[1:10,]
      leftendgammas rightendgammas
 [1,]     0.8813109       2.312310
 [2,]     1.2421964       3.259171
 [3,]     1.5148583       3.974559
 [4,]     3.3756159       8.856660
 [5,]     0.8889774       2.332425
 [6,]     0.9720252       2.550319
 [7,]     1.2981744       3.406042
 [8,]     1.2751523       3.345638
 [9,]     0.8727098       2.289743
[10,]     0.7117375       1.867398

>  contains2gammas <- 2 > leftendgammas & 2 < rightendgammas
> contains2gammas <- as.numeric(contains2gammas)
> sum(contains2gammas) / length(contains2gammas)
[1] 0.705

# Uh-oh.  This procedure doesn’t seem to work very well for this non-normal
# population!


# When we analyze real data, we’d better check whether the sample is likely
to have come from a normal population before using this confidence interval
method.
```

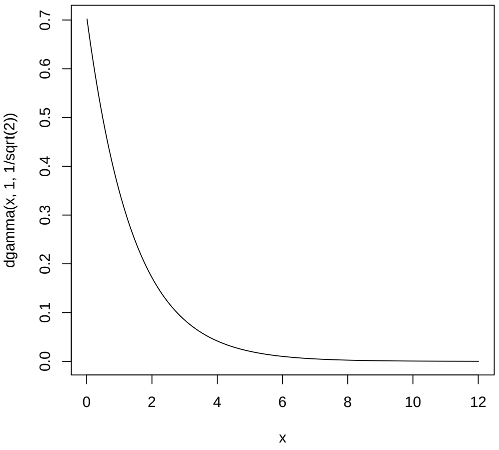

Figure 1: Gamma density plot

Properties of point estimation procedures that we can evaluate with simulation studies

- bias
  - the average value of the estimator minus the true parameter value
  - tells you whether the estimation procedure gives results that are systematically too high or too low
- mean squared error
  - the average squared difference between the estimator and the true parameter value
  - assesses how variable the estimator is and how far off on average (regardless of direction)

Assessing bias and MSE in our variance problem

```
# bias in normal case

> ssq <- apply( normals, 1, var)
> mean(ssq) - 2
[1] -0.007934868

# mean squared error in normal case

> mean( (ssq - 2)^2 )
[1] 0.2300223


# bias in skewed case

> mean(ssqgammas) - 2
[1] -0.02777675

# mean squared error in skewed case

> mean( (ssqgammas - 2)^2 )
[1] 0.8638287
```

Using simulation studies to evaluate properties of hypothesis tests

- significance level α
- power

Types of error in hypothesis testing

$H_0: \ \mu = \mu_0$
$H_a: \ \mu \neq \mu_0$

	True state of the world	
	H_0 is false	H_0 is true
Reject H_0	Correct!	Type I error
Do not reject H_0	Type II error	Correct!

$\alpha =$ P(reject H_0 | H_0 is true)
$\beta =$ P(fail to reject H_0 | H_0 is false)

or, put another way

$\alpha =$ probability of making Type I error
$\beta =$ probability of making Type II error
power $= 1 - \beta =$ probability of correctly rejecting H_0 when it is fals

Reproducible research and simulation studies

- research should be conducted in such an organized way, and documented well enough, that a different researcher could figure out how to repeat it and get the same (or similar) results
- pseudorandom number generation
- setting random number seeds and keeping track of which ones were used in any pseudorandom operations is necessary for simulation studies that will be formally presented or submitted for publication