Lucid printing of floating-point vectors

Kevin Wright

July 2, 2015

1 Abstract

The lucid package provides a function for printing vectors of floating point numbers in a human-friendly format. An application is presented for printing of variance components from mixed models.

2 Introduction

Numerical output from R is often in scientific notation, which can make it difficult to quickly glance at numbers and understand the relative sizes of the numbers. This not a new phenomenon. Before R had been created, (Finney, 1988, 351-352) had this to say about numerical output:

> Certainly, in initiating analyses by standard software or in writing one's own software, the aim should be to have output that is easy to read and easily intelligible to others. ... Especially undesirable is the so-called 'scientific notation' for numbers in which every number is shown as a value between 0.0 and 1.0 with a power of 10 by which it must be multiplied. For example:
>
> ```
> 0.1234E00 is 0.1234
> 0.1234E02 is 12.34
> 0.1234E-1 is 0.01234
> ```
>
> This is an abomination which obscures the comparison of related quantities; tables of means or of analyses of variance become very difficult to read. It is acceptable as a default when a value is unexpectedly very much larger or smaller than its companions, but its appearance as standard output denotes either lazy programming or failure to use good software properly. Like avoidance of 'E', neat arrangement of output values in columns, with decimal points on a vertical line, requires extra effort by a programmer but should be almost mandatory for any software that is to be used often.

One recommendation for improving the display of tables of numbers is to round numbers to 2 (Wainer, 1997) or 3 (Feinberg and Wainer, 2011) digits. Feinberg and Wainer (2011) give the following justification for aggresive rounding:

1. Humans cannot comprehend more than three digits very easily.
2. We almost never care about accuracy of more than three digits.
3. We can only rarely justify more than three digits of accuracy statistically.

An alternative to significant digits is the concept of "effective digits" (Ehrenberg, 1977; Kozak et al., 2011), which considers the amount of variation in the data.

In R, using the `round` and `signif` functions can be used to round to 3 digits of accuracy, but those functions can still print results in scientific notation and leave much to be desired. The `lucid` package provides functions to improve the presentation of floating point numbers in a clear (or lucid) way that makes interpretation of the numbers immediately apparent.

Consider the following vector of coefficients from a fitted model:

```
##                    effect
## A           -1.350000e+01
## B            4.500000e+00
## C            2.450000e+01
## C1           6.927792e-14
## C2          -1.750000e+00
## D            1.650000e+01
## (Intercept)  1.135000e+02
```

Which coeficient is zero? How large is the intercept?

Both questions can be answered using the output shown above, but it takes too much effort to answer the questions. Now examine the same vector of coefficients with prettier formatting:

```
require("lucid")
options(digits=7) # knitr defaults to 4, R console uses 7
lucid(df1)
```

```
##             effect
## A           -13.5
## B             4.5
## C            24.5
## C1            0
## C2           -1.75
## D            16.5
## (Intercept) 114
```

Which coeficient is zero? How large is the intercept?

Printing the numbers with the `lucid` function has made the questions much easier to answer.

The sequence of steps used by `lucid` to format and print the output is.

1. Zap small numbers to zero
2. Round using 3 significant digits (user controllable option)
3. Drop trailing zeros
4. Align numbers at the decimal point

The `lucid` package contains a generic function `lucid` with specific methods for numeric vectors, data frames, and lists. The method for data frames applies formatting to each numeric column and leaves other columns unchanged. The `lucid` function is primarily a *formatting* function, the results of which are passed to the regular `print` functions.

3 Example: Antibiotic effectiveness

Wainer and Larsen (2009) present data published by Will Burtin in 1951 on the effectiveness of antibiotics against 16 types of bacteria. The data is included in the `lucid` package as a dataframe called `antibiotic`. The default view of this data is:

```
print(antibiotic)

##                          bacteria penicillin streptomycin neomycin gramstain
## 1            Aerobacter aerogenes    870.000         1.00    1.600       neg
## 2                Brucella abortus      1.000         2.00    0.020       neg
## 3               Brucella antracis      0.001         0.01    0.007       pos
## 4           Diplococcus pneumoniae     0.005        11.00   10.000       pos
## 5                Escherichia coli    100.000         0.40    0.100       neg
## 6            Klebsiella pneumoniae   850.000         1.20    1.000       neg
## 7        Mycobacterium tuberculosis  800.000         5.00    2.000       neg
## 8                Proteus vulgaris      3.000         0.10    0.100       neg
## 9           Pseudomonas aeruginosa   850.000         2.00    0.400       neg
## 10 Salmonella (Eberthella) typhosa     1.000         0.40    0.008       neg
## 11        Salmonella schottmuelleri   10.000         0.80    0.090       neg
## 12            Staphylococcus albus     0.007         0.10    0.001       pos
## 13           Staphylococcus aureus     0.030         0.03    0.001       pos
## 14           Streptococcus fecalis     1.000         1.00    0.100       pos
## 15       Streptococcus hemolyticus     0.001        14.00   10.000       pos
## 16          Streptococcus viridans     0.005        10.00   40.000       pos
```

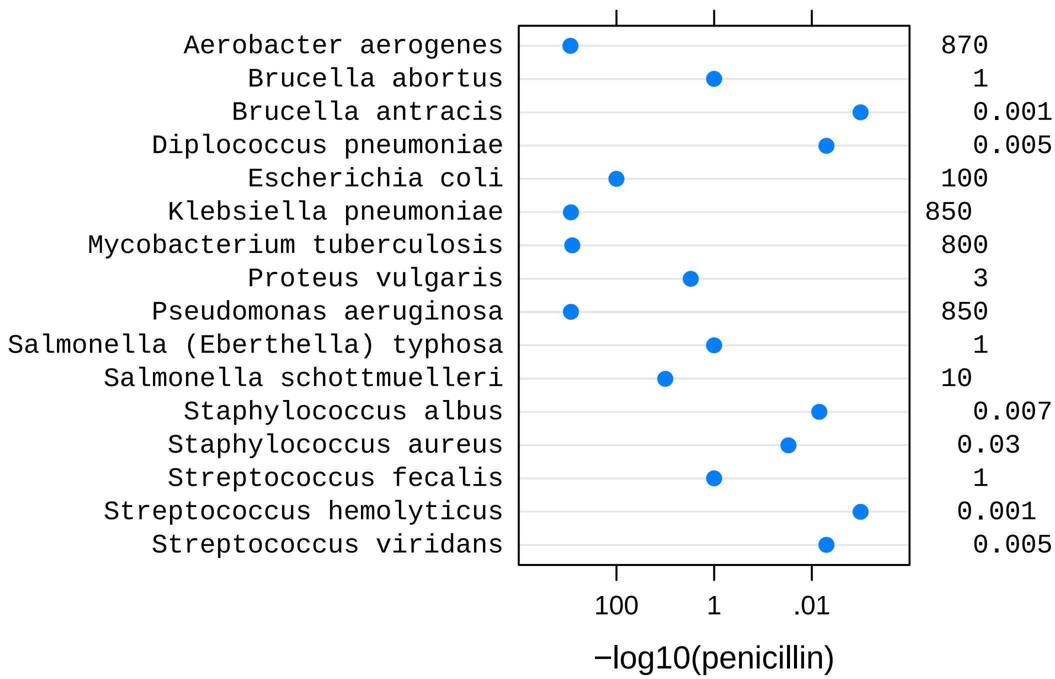
Due to the wide range in magnitude of the values, nearly half of the floating-point numbers in the default view contain trailing zeros after the decimal, which adds significant clutter and impedes interpretation. The `lucid` display of the data is:

```
lucid(antibiotic)

##                          bacteria penicillin streptomycin neomycin gramstain
## 1            Aerobacter aerogenes    870            1        1.6         neg
## 2                Brucella abortus      1            2        0.02        neg
## 3               Brucella antracis      0.001        0.01     0.007       pos
## 4           Diplococcus pneumoniae     0.005       11       10           pos
## 5                Escherichia coli    100            0.4      0.1         neg
## 6            Klebsiella pneumoniae   850            1.2      1           neg
## 7        Mycobacterium tuberculosis  800            5        2           neg
## 8                Proteus vulgaris      3            0.1      0.1         neg
## 9           Pseudomonas aeruginosa   850            2        0.4         neg
## 10 Salmonella (Eberthella) typhosa     1            0.4      0.008       neg
## 11        Salmonella schottmuelleri   10            0.8      0.09        neg
## 12            Staphylococcus albus     0.007        0.1      0.001       pos
## 13           Staphylococcus aureus     0.03         0.03     0.001       pos
## 14           Streptococcus fecalis     1            1        0.1         pos
## 15       Streptococcus hemolyticus     0.001       14       10           pos
## 16          Streptococcus viridans     0.005       10       40           pos
```

The lucid display is dramatically simplified, providing a clear picture of the effectiveness of the antibiotics against bacteria. This view of the data matches exactly the appearance of Table 1 in Wainer and Larsen (2009).

A stem-and-leaf plot is a semi-graphical display of data, in that the *positions* of the numbers create a display similar to a histogram. In a similar manner, the lucid output is a semi-graphical view of the data. The figure below shows a dotplot of the penicillin values on a reverse log10 scale. The values are also shown along the right axis in lucid format. Note the similarity in the overall shape of the dots and the positions of the left-most significant digit in the numerical value.



4 Application to mixed models

During the process of iterative fitting of mixed models, it is often useful to compare fits of different models to data, for example using loglikelihood or AIC values, or with the help of resiudal plots. It can also be very informative to inspect the estimated values of variance components.

To that end, the generic VarCorr function found in the nlme (Pinheiro et al., 2014) and lme4 (Bates et al., 2014) packages can be used to print variance estimates from fitted models. The VarCorr function is not available for models obtained using the asreml (Butler, 2009) package.

The lucid package provides a generic function called vc that provides a unified interface for extracting the variance components from fitted models obtained from the asreml, lme4, nlme, and rjags packages. The vc function has methods specific to each package that make it easy to extract the estimated variances and correlations from fitted models and formats the results using the lucid function.

Pearce et al. (1988) suggest showing four significant digits for the error mean square and two decimal places digits for F values. The lucid function uses a similar philosophy, presenting the variances with four significant digits and asreml Z statistics with two significant digits.

4.1 Example 1 - Rail data

The following simple example illustrates use of the vc function for identical REML models in the nlme, lme4, and asreml packages. The travel times of ultrasonic waves in six steel rails was modeled as an overall mean, a random effect for each rail, and a random residual. The package rjags is used to fit a similar Bayesian model inspired by Wilkinson (2014).

```
require("nlme")
data(Rail)
mn <- lme(travel~1, random=~1|Rail, data=Rail)
vc(mn)
```

```
##       effect variance stddev
##  (Intercept)    615.3  24.81
##     Residual    16.17  4.021
```

```
require("lme4")
m4 <- lmer(travel~1 + (1|Rail), data=Rail)
vc(m4)
```

```
##       grp        var1 var2   vcov  sdcor
##      Rail (Intercept) <NA> 615.3  24.81
##  Residual        <NA> <NA>  16.17  4.021
```

```
# require("asreml")
# ma <- asreml(travel~1, random=~Rail, data=Rail)
# vc(ma)
##         effect component std.error z.ratio constr
##  Rail!Rail.var     615.3     392.6     1.6    pos
##     R!variance     16.17       6.6     2.4    pos
```

In a Bayesian model all effects can be considered as random.

```
require("nlme")
data(Rail)
require("rjags")
m5 <-
"model {
for(i in 1:nobs){
  travel[i] ~ dnorm(mu + theta[Rail[i]], tau)
}
for(j in 1:6) {
  theta[j] ~ dnorm(0, tau.theta)
}
mu ~ dnorm(50, 0.0001) # Overall mean. dgamma()
tau ~ dgamma(1, .001)
tau.theta ~ dgamma(1, .001)
```

```
residual <- 1/sqrt(tau)
sigma.rail <- 1/sqrt(tau.theta)
}"
jdat <- list(nobs=nrow(Rail), travel=Rail$travel, Rail=Rail$Rail)
jinit <- list(mu=50, tau=1, tau.theta=1)
tc5 <- textConnection(m5)
j5 <- jags.model(tc5, data=jdat, inits=jinit, n.chains=2, quiet=TRUE)
close(tc5)
c5 <- coda.samples(j5, c("mu","theta", "residual", "sigma.rail"),
                   n.iter=100000, thin=5)
```

```
vc(c5)
```

```
##                Mean      SD     2.5%   Median    97.5%
## mu            66.59    9.916   46.66    66.7     86.41
## residual       3.947   0.8258   2.716    3.824    5.913
## sigma.rail    23.39    7.519   13.67    21.87    42.01
## theta[1]     -34.49   10.08   -54.84   -34.58   -14.51
## theta[2]     -16.37   10.11   -36.68   -16.47     3.803
## theta[3]     -12.43   10.09   -32.57   -12.57     7.651
## theta[4]      15.88   10.11    -4.207   15.77    36.25
## theta[5]      17.85   10.1     -2.088   17.68    38.12
## theta[6]      29.03   10.09     9.1     28.88    49.54
```

Compare these JAGS point estimates and quantiles with the results from `lme4`.

```
m4
```

```
## Linear mixed model fit by REML [lmerMod]
## Formula: travel ~ 1 + (1 | Rail)
##    Data: Rail
## REML criterion at convergence: 122.177
## Random effects:
##  Groups   Name        Std.Dev.
##  Rail     (Intercept) 24.805
##  Residual              4.021
## Number of obs: 18, groups:  Rail, 6
## Fixed Effects:
## (Intercept)
##        66.5
```

```
ranef(m4)
```

```
## $Rail
##   (Intercept)
## 2   -34.53091
## 5   -16.35675
## 1   -12.39148
## 6    16.02631
## 3    18.00894
```

```
## 4    29.24388
```

While the lucid function is primarily a formatting function and uses the standard print functions in R, the vc function defines an additional class for the value of the function and has dedicated print methods for the class. This was done to allow additional formatting of the results.

4.2 Example 2 - federer.diagcheck data

The second, more complex example is based on a model in Federer and Wolfinger (2003) in which orthogonal polynomials are used to model trends along the rows and columns of a field experiment. The data are available in the agridat package Wright (2014) as the federer.diagcheck data frame. The help page for the data shows how to reproduce the analysis of Federer and Wolfinger (2003). When using the lme4 package to reproduce the analysis, two different optimizers are available. Do the two different optimizers lead to similar estimated variances?

In the output below, the first column identifies terms in the model, the next two columns are the variance and standard deviation from the bobyqa optimizer, while the final two columns are from the NelderMead optimizer. These results are from lme4 version 1.1-7 and are likely to be different for later versions of lme4

The default output printing is shown first.

```
print(out)

##           term     vcov-bo  sdcor-bo sep       vcov-ne     sdcor-ne
## 1  (Intercept)   2869.4469  53.56722     3.228419e+03  56.81917727
## 2        r1:c3   5531.5724  74.37454     7.688139e+03  87.68203447
## 3        r1:c2  58225.7678 241.30016     6.974755e+04 264.09761622
## 4        r1:c1 128004.1561 357.77668     1.074270e+05 327.76064925
## 5           c8   6455.7495  80.34768     6.787004e+03  82.38327224
## 6           c6   1399.7294  37.41296     1.636128e+03  40.44907560
## 7           c4   1791.6507  42.32790     1.226846e+04 110.76308194
## 8           c3   2548.8847  50.48648     2.686302e+03  51.82954364
## 9           c2   5941.7908  77.08301     7.644730e+03  87.43414634
## 10          c1      0.0000   0.00000     1.225143e-03   0.03500204
## 11         r10   1132.9501  33.65932     1.975505e+03  44.44665149
## 12          r8   1355.2291  36.81344     1.241429e+03  35.23391157
## 13          r4   2268.7296  47.63118     2.811241e+03  53.02113582
## 14          r2    241.7894  15.54958     9.282275e+02  30.46682578
## 15          r1   9199.9022  95.91612     1.036358e+04 101.80169429
## 16        <NA>   4412.1096  66.42371     4.126832e+03  64.24042100
```

How similar are the variance estimates obtained from the two optimization methods? It is difficult to compare the results due to the clutter of extra digits, and because of some quirks in the way R formats the output. The variances in column 2 are shown in non-scientific format, while the variances in column 5 are shown in scientific format. The standard deviations are shown with 5 decimal places in column 3 and 8 decimal places in column 6. (All numbers were stored with 15 digits of precision.)

The `lucid` function is now used to show the results in the manner of the `vc` function.

```
lucid(out, dig=4)

##           term  vcov-bo sdcor-bo sep  vcov-ne sdcor-ne
## 1  (Intercept)   2869      53.57       3228     56.82
## 2        r1:c3   5532      74.37       7688     87.68
## 3        r1:c2  58230     241.3       69750    264.1
## 4        r1:c1 128000     357.8      107400    327.8
## 5           c8   6456      80.35       6787     82.38
## 6           c6   1400      37.41       1636     40.45
## 7           c4   1792      42.33      12270    110.8
## 8           c3   2549      50.49       2686     51.83
## 9           c2   5942      77.08       7645     87.43
## 10          c1      0       0             0      0.035
## 11         r10   1133      33.66       1976     44.45
## 12          r8   1355      36.81       1241     35.23
## 13          r4   2269      47.63       2811     53.02
## 14          r2    241.8    15.55        928.2   30.47
## 15          r1   9200      95.92      10360    101.8
## 16        <NA>   4412      66.42       4127     64.24
```

The formatting of the variance columns is consistent as is the formatting of the standard deviation columns. Fewer digits are shown. It is easy to compare the columns and see that the two optimizers are giving quite different answers.

Note. Numeric matrices are printed with quotes. Use `noquote()` to print without quotes, for example:

```
noquote(lucid(as.matrix(head(mtcars)),2))
```

```
##                   mpg cyl disp hp  drat wt  qsec vs am gear carb
## Mazda RX4         21  6   160  110 3.9  2.6 16   0  1  4    4
## Mazda RX4 Wag     21  6   160  110 3.9  2.9 17   0  1  4    4
## Datsun 710        23  4   110   93 3.8  2.3 19   1  1  4    1
## Hornet 4 Drive    21  6   260  110 3.1  3.2 19   1  0  3    1
## Hornet Sportabout 19  8   360  180 3.2  3.4 17   0  0  3    2
## Valiant           18  6   220  100 2.8  3.5 20   1  0  3    1
```

5 Acknowledgements

Thanks to Deanne Wright for a helpful review of this paper.

6 Appendix

Session information:

- R version 3.1.3 (2015-03-09), `x86_64-w64-mingw32`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: Matrix 1.2-1, coda 0.17-1, knitr 1.10.5, lattice 0.20-31, lme4 1.1-8, lucid 1.3, nlme 3.1-120, rjags 3-15
- Loaded via a namespace (and not attached): MASS 7.3-41, Rcpp 0.11.6, evaluate 0.7, formatR 1.2, grid 3.1.3, highr 0.5, magrittr 1.5, minqa 1.2.4, nloptr 1.0.4, splines 3.1.3, stringi 0.5-2, stringr 1.0.0, tools 3.1.3

References

D. Bates, M. Maechler, B. Bolker, and S. Walker. *lme4: Linear mixed-effects models using Eigen and S4*, 2014. URL http://CRAN.R-project.org/package=lme4. R package version 1.1-7.

David Butler. *asreml: asreml() fits the linear mixed model*, 2009. URL www.vsni.co.uk. R package version 3.0.

A. S. C. Ehrenberg. Rudiments of numeracy. *Journal of the Royal Statistical Society. Series A*, pages 277–297, 1977.

Walter T Federer and Russell D Wolfinger. *Handbook of Formulas and Software for Plant Geneticists and Breeders*, chapter PROC GLM and PROC MIXED Codes for Trend Analyses for Row-Column Designed Experiments. Haworth Press, 2003.

Richard A Feinberg and Howard Wainer. Extracting sunbeams from cucumbers. *Journal of Computational and Graphical Statistics*, 20(4):793–810, 2011.

D. J. Finney. Was this in your statistics textbook? II. Data handling. *Experimental agriculture*, 24:343–353, 1988.

Marcin Kozak, Ricardo Antunes Azevedo, Justyna Jupowicz-Kozak, Wojtek Krzanowski, et al. Reporting numbers in agriculture and biology: Don't overdo the digits. *Australian Journal of Crop Science*, 5: 1876–1881, 2011.

S. C. Pearce, G. M. Clarke, G. V. Dyke, and R. E. Kempson. *A Manual of Crop Experimentation*. Charles Griffin and Company, 1988.

Jose Pinheiro, Douglas Bates, Saikat DebRoy, Deepayan Sarkar, and R Core Team. *nlme: Linear and Nonlinear Mixed Effects Models*, 2014. URL http://CRAN.R-project.org/package=nlme. R package version 3.1-118.

Howard Wainer. Improving tabular displays, with NAEP tables as examples and inspirations. *Journal of Educational and Behavioral Statistics*, 22(1):1–30, 1997. doi: 10.3102/10769986022001001.

Howard Wainer and Mike Larsen. Pictures at an exhibition. *Chance*, 22(2):46–54, 2009. doi: 10.1080/09332480.2009.10722958. URL http://chance.amstat.org/2009/04/visrev222/.

Darren Wilkinson. One-way anova with fixed and random effects from a bayesian perspective, 2014. URL https://darrenjw.wordpress.com/2014/12/22/one-way-anova-with-fixed-and-random-effects-from-a-bayesian-perspective/.

Kevin Wright. *agridat: Agricultural datasets*, 2014. URL http://CRAN.R-project.org/package=agridat. R package version 1.10.