A Fast, High-Precision Implementation of the Univariate One-Parameter Box-Cox Transformation Using the Golden Section Search in SAS/IML®

Charles D. Coleman, U.S. Census Bureau, Population Division

ABSTRACT

The one-parameter Box-Cox transformation is implemented using the golden section search algorithm in SAS/IML®. This code quickly produces the optimal value of the transformation to the user-specified parameter. It is presented in the form of a macro which has been tested using The SAS System for Windows®, releases 6.12 and 8.2. Changes to the SAS/IML® code are also presented for Schlesselman's data-independent version of the Box-Cox transformation.

INTRODUCTION

The one-parameter Box-Cox transformation (Box and Cox, 1964) is a popular transformation for eliminating skewness in continuous data where all values are positive.[1] It is a type of power transformation: the data are exponentiated. The goal of the transformation is to maximize the probability that the transformed data come from a symmetric normal distribution. If the data are already symmetric, such as those generated by a symmetric normal distribution, then the transformation is unnecessary. The Box-Cox transformation has many practical uses. Emerson and Stoto (1983) provide several examples of its uses and interpretations. Emerson (1983) further looks at mathematical properties of the transformation. This paper also considers Schlesselman's (1971) scale-invariant, data-independent form of the Box-Cox transformation.

The one-parameter Box-Cox transformation has the form

$$y^{(\lambda)} = \begin{cases} \dfrac{y^{\lambda}-1}{\lambda} & \lambda \neq 0 \\ \ln y & \lambda = 0 \end{cases} \quad . \qquad (1)$$

The constant term in the upper ($\lambda \neq 0$) definition of $y^{(\lambda)}$ makes the transformation continuous in λ at $\lambda = 0$. The optimal value of λ, λ^*, results from maximizing the log-likelihood function

$$\ell(\lambda) = -\frac{n}{2}\ln\left[\frac{1}{n}\sum_{i=1}^{n}\left(y_i^{(\lambda)} - \overline{y^{(\lambda)}}\right)^2\right] + (\lambda-1)\sum_{i=1}^{n}\ln y_i \qquad (2)$$

where i indexes the n observations and

$$\overline{y^{(\lambda)}} = \frac{1}{n}\sum_{i=1}^{n} y_i^{(\lambda)} \qquad (3)$$

is the average of the $y_i^{(\lambda)}$.

Schlesselman (1971) pointed out that the original Box-Cox transformation is not scale-invariant: multiplying the y_i by a constant changes λ^*. Schlesselman (1971, p. 310, eq. 12) offers a scale-invariant, data-independent version of (1):[2]

$$y_k^{(\lambda)} = \begin{cases} \dfrac{y^{\lambda}-k^{\lambda}}{\lambda} & \lambda \neq 0 \\ \ln(y/k) & \lambda = 0 \end{cases} \qquad (4)$$

where $k > 0$ is an arbitrary constant in the same measurement units as y. Alterations to the code to implement this will be presented.

The Box-Cox transformation is implemented in the form of a macro. You should feel free to alter the implementation to suit your purposes. For example, you may want to perform additional operations on the data or perform simulation studies. In the first case, you may decide to add code to the macro. In the latter case, you may dispense with macro code entirely and invoke PROC IML directly from a SAS® command file.

THE CODE

Macro `fastbc` implements the Box-Cox transformation using the golden section search algorithm. Not only does it achieve high accuracy, it does so quickly. Other implementations in SAS®, either conduct computationally expensive grid searches, such as SAS/QC's® `adxtrans` macro, SAS/ETS's® `boxcoxar` macro and Dimakos's (1995) `bctrans` macro, or require computing derivatives, such as Hyde (1999). Another alternative, implemented in some other packages requires computing derivatives to maximize $\ell(\lambda)$. In the literature, Ogwang and Rao (1997) describe a set of methods based on Taylor series expansions that require computing derivatives.

The golden section search simply starts with prespecified minimum and maximum values a and b, which bracket the maximum of $\ell(\lambda)$. That is, the maximum lies in the interval (a,b). The golden ratio $r = \left(\sqrt{5}-1\right)/2 \approx 0.61803399$ is predefined. Then, the two new points c and d are calculated as $c = a + r\,(b - a)$ and $d = b - r\,(b - a)$. If $\ell(c) > \ell(d)$, then $a \leftarrow d$ and $d \leftarrow c$. Otherwise, $b \leftarrow c$ and $c \leftarrow d$. The process is iterated until $|a - b| <$ `tol`. Each iteration successively narrows the bracket surrounding the maximum. The second assignment in each pair reduces computation complexity by carrying forward a previously calculated intermediate point. The upshot is that each iteration only requires one evaluation of $\ell(\lambda)$.

The value of `tol`, the tolerance, has to be chosen carefully. Press et al. (1992, p. 398) show that the minimum tolerance should be the square root of the machine precision. Since The SAS System® stores numbers as eight byte reals, its machine precision is about `1e-15`. Thus, the minimum usable value of `tol` is about `3e-8`. The macro uses `tol = 1e-7`.

```
%macro fastbc(data=, out=, var=, min=,
  max=);

  proc iml;
    use &data;
    read all var {&var} into y;
    n = nrow(y);
    sumlog = sum(log(y));
    tol = 1e-7;
    r = 0.61803399;
    a = &min;
    b = &max;

    start llf(lambda) global(tol, y, n,
      sumlog);
      if abs (lambda) < tol then yl = log(y);
      else yl = (y ## lambda - 1)/lambda;
      avgyl = yl[:,];
      f = - n * log(ssq(yl - avgyl)/n)/2 +
        (lambda - 1) * sumlog;
      return(f);
    finish llf;

    fa = llf(a);
    fb = llf(b);
    c = a + r*(b-a);
    fc = llf(c);

    do while ((fc < fa) | (fc < fb));
```

```
      if (fc < fa)
        then do;
          a = a-2;
          fa = llf(a);
        end;
      else do;
        b = b+2;
        fb = llf(b);
      end;
    c = a + r*(b-a);
    fc = llf(c);
  end;

  it = 0;
  cdflag = 'c';
  do while (b - a > tol);
    it = it + 1;
    diff = r * (b - a);
    if (cdflag = 'd')
      then do;
        c = a + diff;
        fc = llf(c);
      end;
    if (cdflag = 'c')
      then do;
        d = b - diff;
        fd = llf(d);
      end;
    if (fc > fd)
      then do;
        a = d;
        fa = fd;
        d = c;
        fd = fc;
        cdflag = 'd';
      end;
      else do;
        b = c;
        fb = fc;
        c = d;
        fc = fd;
        cdflag = 'c';
      end;

    if (fa > fb)
      then lambda = a;
      else lambda = b;

    print "Iterations" it;
    create &out from lambda[colname=lambda];
    append from lambda;

  quit;
%mend;
```

IMPLEMENTING SCHLESSELMAN'S VARIANT

Schlesselman's (1971) variant requires adding a line of code and modifying module `llf`. The additional line can be placed anywhere before the `do while` statement:

```
k = 2;
```

`k` can be any positive value other than 1. Module `llf` is changed to

```
start llf(lambda) global(tol, y, n, sumlog,
  k);
  if abs (lambda) < tol
    then yl = log(y / lambda);
    else yl = (y ## lambda -
      k**lambda)/lambda;
  avgyl = yl[:,];
  f = - n * log(ssq(yl - avgyl)/n)/2 +
  (lambda - 1) * sumlog;
    return(f);
  finish llf;
```

This module differs from the original one by declaring `k` to be `global` in the `start` statement and by implementing equation (4).

INVOKING THE MACRO

The macro is invoked with five arguments:

`data`: The data set which contains the variable to be transformed.

`out`: The data set to which variable `lambda`, containing λ*, is output.

`var`: The variable to be transformed.

`min`: The minimum initial value of λ. For most data sets, -2 should suffice.

`max`: The maximum initial value of λ. For most data sets, 2 should suffice.

INSIDE THE CODE

First, variable `&var` is read from data set `&data` into vector `y`. Two global variables are then calculated: `n`, the number of rows of `y` and `sumlog`, the sum of the natural logarithms of the elements of `y`. The next four statements then initialize constants: `tol`, the tolerance, `r`, the golden ratio already defined in Section "The Code," `&min` is loaded into `a`, the initial minimum value of λ, and `&max` is loaded into `b`, the initial maximum value of λ. The initializations end with the definition of module `llf`, which operationalizes $\ell(\lambda)$. Note that within module `llf`, whenever |`lambda`| < `tol`, `lambda` is assumed to be 0. Again, this is an implementation of the idea of tolerance: when `lambda` is close enough to 0, it is computationally indistinguishable from 0.

The block of code immediately after the definition of module `llf`, through the first `do while` loop, tests for the existence of the maximum in [`a`,`b`]. If this interval is not found to bracket the maximum, then an endpoint is changed to widen the interval until bracketing is achieved. In the process, the values of `fa = llf(a)`, `fb = llf(b)` and `fc = llf(c)` are calculated for use in the first iteration.

One more initialization precedes the `do while` loop that controls the iterations. Variable `cdflag` indicates whether `fc = llf(c)` or `fd = llf(d)` has been computed. A value of `'c'` or `'d'` indicates that `fc` or `fd` has been calculated, respectively. During each iteration, the other member of the pair is calculated immediately after the `do while` statement, until convergence. Convergence is defined by the condition `b - a > tol`. The absolute value is not need, as `b` is always greater than `a`. The `if-then` block tests whether `fc` > `fd` and assigns variables accordingly.

After convergence, `a` or `b` is output as variable `lambda` in dataset `&out`, according to whether fa > fb. The `quit` statement exits PROC IML and `%mend` ends the macro.

OPTIMIZING OTHER FUNCTIONS

You can modify the golden section search to optimize other functions, finding either the maximum or minimum. A requirement is that the optimum is global. This is assured if only one optimum exists. A sufficient condition is that the function is

quasi-concave or quasi-convex. A function *f* is strictly quasi-concave if and only if for any two real values *x* and *y*, $f(tx+(1-t)y) = \min\{f(x), f(y)\}$ for $0 < t < 1$ implies $x = y$. Likewise, a function *f* is quasi-convex if and only if the function *–f* is quasi-concave. Quasi-concave functions have unique maxima while quasi-convex functions have unique minima.

To optimize a function other than `llf(lambda)`, simply create a module with the function to be optimized and replace module `llf` with it. Then, change all calls to `llf()` with calls to your new function.

To find the minimum, simply reverse all of the inequalities which compare function values. For example, the first inequality

```
if fc < fa then do;
```

becomes

```
if fc > fa then do; .
```

AN APPLICATION

Swanson, Tayman and Barr (2000) published a dataset of absolute percentage errors of forecasts of the 1970 populations of counties in Washington state. They used the Box-Cox transformation to develop a new measure of forecast accuracy. PROC UNIVARIATE produced the following moment statistics and normality tests:[3]

The UNIVARIATE Procedure
Variable: APE (APE)

Moments

N	39	Sum Weights	39
Mean	5.06787	Sum Observations	197.647
Std Deviation	3.812	Variance	4.5299
Skewness	0.83522	Kurtosis	-0.2923
Uncorr SS	1553.7869	Corrected SS	552.138
Coeff Var	75.2153	Std Error Mean	0.61038

Tests for Normality

Test		-Statistic-		-----p Value-----
Shapiro-Wilk	W	0.9072	Pr < W	0.0036
Kolmogorov-Smirnov	D	0.1554	Pr > D	0.0184
Cramer-von Mises	W-Sq	0.1903	Pr > W-Sq	0.0068
Anderson-Darling	A-Sq	1.1959	Pr > A-Sq	<0.0050

The four normality tests reject normality at *p*-values all less than 0.02. This appears to be driven by the large value of skewness. Thus, the Box-Cox transformation is appropriate for reducing skewness in this dataset and obtaining normally distributed transformed variables.

Macro `fastbc` returned $\lambda^* = 0.2946924$. After transforming variable APE using equation (1), PROC UNIVARIATE reported the following:

The UNIVARIATE Procedure
Variable: APET

Moments

N	39	Sum Weights	39
Mean	1.73013	Sum Observations	67.47498
Std Deviation	1.2948	Variance	1.67644
Skewness	-0.07783	Kurtosis	-0.776696
Uncorr SS	180.44511	Corrected SS	63.7047817
Coeff Var	74.836949	Std Error Mean	0.20732989

Tests for Normality

Test		-Statistic-		-----p Value-----
Shapiro-Wilk	W	0.9767	Pr < W	0.5854
Kolmogorov-Smirnov	D	0.0699	Pr > D	>0.1500
Cramer-von Mises	W-Sq	0.0274	Pr > W-Sq	>0.2500
Anderson-Darling	A-Sq	0.2129	Pr > A-Sq	>0.2500

Transformation reduces the absolute value of skewness by an order of magnitude (.8 to .08) and makes all tests for normality insignificant. However, the transformed data are more platykurtotic, that is, they have even less kurtosis than the normal distribution.[4] This can be seen in the decrease in the already negative kurtosis from -.3 to -.7. Thus, the transformed data, in this respect, depart even more from normality. In fact, it is possible for the Box-Cox transformation to produce data whose departure from 0 kurtosis causes rejection of normality.[5] The reason for this lies in the definition of the Box-Cox transformation: it maximizes the probability that the transformed data come from normal distribution. However, since it can effectively only operate on positive data and produces positive data unless $\lambda^* = 0$, while the normal distribution is defined on the set of all real numbers, it is impossible to achieve normality.[6] Thus, the Box-Cox transformation is really a quasi-maximum likelihood estimator and does not truly achieve its goal.

COMPARISON TO OTHER ALGORITHMS

Macro `fastbc` required 37 iterations to find the solution with &min = –2 and &max = 2. Each iteration required one evaluation of $\ell(\lambda)$. Adding the three initial evaluations of $\ell(\lambda)$ makes a total of 40 evaluations of $\ell(\lambda)$. An exhaustive grid search on [–2,2] with steps of 10^{-7} requires 4×10^7 iterations. A smarter grid search searches on successively finer grids to find bracketing pairs until convergence. If the grids are of fineness 1, .1, .01, …, 10^{-7}, the best-case performance on the same interval is $3 \times 2^7 = 384$ iterations. Thus, one of the best of the most used competing search algorithms requires a minimum of about 10 times as many evaluations of $\ell(\lambda)$ as `fastbc`, and usually far more. The advantages of `fastbc` compared to these algorithms are clear.

The importance of using the correct convergence criterion can be seen in three instances. Press et al.'s (1992, p. 401-402) golden section search routine `golden` uses a different convergence criterion that stops quicker than that used in the present paper. An implementation of that algorithm for this problem took 20 iterations to converge to an incorrect value. SHAZAM (Whistler et al., 2001) appears to use the change in $\ell(\lambda)$ as the convergence criterion. If $\ell(\lambda)$ is flat around its maximum, it can converge to the wrong value. This problem is particular acute when the accuracy is set to .001 using the ACCURACY option, instead of the default .01. Ogwang and Rao (1997) use the change in the current value of λ as a convergence criterion. Their Table 1 (p. 407) shows λ^* varying as function of the degree of the Taylor series approximation used. In their equation "Consumer goods imports," λ^* varies from .528 to .553. Although they acknowledge imprecision by using .01 as their standard of comparison to SHAZAM's results (Ogwang and Rao, 1997, p.

406), these values vary by about .02, making them different by their own standard. Hyde (1999) develops SAS/IML® code for inference in multivariate Box-Cox models, but uses a far too small tolerance (`1e-10`).

CONCLUSION

This paper has presented SAS/IML® code for using the golden section search to compute fast, high-precision, one-parameter Box-Cox transformations. This code produces estimates of the transformation parameter up to machine precision. It avoids the pitfalls of other algorithms in that it does not converge improperly and it economizes on the number of function evaluations. The golden section search is a general method that is suited to other optimization problems. Adaptation of the code to these problems has been discussed.

REFERENCES

Box, G.E.P. and Cox, D.R. (1964), "An Analysis of Transformations," *Journal of the Royal Statistical Society, Series B*, 26, 211-243.

Dimakos, I.C. (1995), "Power Transformations Using SAS/IML® Software," *Proceedings of the 22nd SAS Users Group International*, http://www2.sas.com/proceedings/sugi22/CODERS/PAPER95.PDF. NC: SAS Institute. Accessed September 4, 2002.

Emerson, J.D. (1983), "Mathematical Aspects of Transformations." In Hoaglin, D.C., Mosteller, F. and Tukey, J.W. [Eds.], *Understanding Robust and Exploratory Data Analysis*, New York: Wiley, 247-282.

Emerson, J.D. and Stoto, M.A. (1983), "Transforming Data." In Hoaglin, D.C., Mosteller, F. and Tukey, J.W. [Eds.], *Understanding Robust and Exploratory Data Analysis*, New York: Wiley, 97-128.

Hyde, S. (1999), "Likelihood Based Inference on the Box-Cox Family of Transformations: SAS and MATLAB Programs," M.S. Thesis, Department of Mathematical Sciences, Montana State University, Bozeman, MT, http://www.math.montana.edu/~hyde/msthesis.pdf. Accessed September 9, 2002.

Ogwang, T. and Rao, U.L.G. (1997), "A Simple Algorithm for Estimating Box-Cox Models," *The Statistician*, 46, 399-409.

Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. (1992), *Numerical Recipes in C: The Art of Scientific Computing*, Second Edition. New York: Cambridge University Press.

SAS Institute, Inc. (1989). *SAS/IML® Software and Reference, Version 6, First Edition*, Cary, NC: SAS Institute.

Schlesselman, J. (1971), "Power Families: A Note on the Box and Cox Transformation," *Journal of the Royal Statistical Society, Series B*, 33, 307-311.

Swanson, D.A., Tayman, J. and Barr, C.F. (2000), "A Note on the Measurement of Accuracy for Subnational Demographic Estimates," *Demography*, 37, 233-249.

Whistler, D., White, K.J., Wong S.D. and Bates, D. (2001), *SHAZAM Version 9 User's Reference Manual*, Vancouver, BC, Canada: Northwest Econometrics.

Zarembka, P. (1974), "Transformation of Variables in Econometrics," in Zarembka, P. [ed.], *Frontiers of Econometrics*, New York: Academic Press.

ACKNOWLEDGMENTS

I would like to thank Rob Agnelli of SAS Institute, Inc. for help with debugging an earlier version of this macro and Aref N. Dajani for peer review and improvements to the macro. This report is released to inform interested parties of research. The views expressed on methodological and technical issues are those of the author and not necessarily those of the U.S. Census Bureau.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Charles D. Coleman
U.S. Census Bureau
Washington, DC 20233-8800
Work Phone: +1 (301) 763-6068
Fax: +1 (301) 457-2481
Email: charles.d.coleman@census.gov

---

[1] Box and Cox (1964) also define a two-parameter transformation that includes a shift parameter. This paper does not consider that transformation. The presence of the second parameter creates a multivariate optimization problem, which requires different techniques than those used and discussed in the present paper.

[2] Schlesselman (1971) uses *c* for *k*. *k* is used in the present paper for consistency with the SAS/IML® code herein.

[3] The listings from PROC UNIVARIATE have been edited to fit into one column.

[4] Graphically, a platykurtotic distribution has a larger modal value and thinner tails than a normal distribution.

[5] Thomas Bryan (personal correspondence) has observed this in several data sets.

[6] J.B. Ramsey (quoted by Zarembka (1974, p. 87)) appears to have been the first to observe this. However, Zarembka (1974, p. 87) notes that "if the probability of large negative values [of the transformed data] is quite low, the error term may still be approximately normal."