*Journal of Statistical Software*

*November 2007, Volume 23, Code Snippet 1.* *http://www.jstatsoft.org/*

# Accurate Computation of the $F$-to-$z$ and $t$-to-$z$ Transforms for Large Arguments

**Paul Hughett**
University of Pennsylvania

**Abstract**

The obvious method for transforming an $F$ or $t$ variate into a $z$ variate is via the corresponding $p$ value, but this method is inaccurate when $F$ or $t$ is so large that $p$ differs from 1 by only the last few bits of its floating-point representation. More accurate results may be obtained by using $q = 1 - p$ as the intermediate variable so that the full precision of the floating-point representation is available.

*Keywords*: statistical parametric mapping, statistical computation, $F$-to-$z$ transform, $t$-to-$z$ transform, floating-point precision.

Statistical parametric mapping (SPM, Frackowiak, Friston, Frith, Dolan, and Mazziotta 1997) evaluates some statistical hypothesis at each voxel of a set of images which have been registered to a common coordinate system. Depending on the form of the hypothesis being tested, the statistic may be described by an $F$ or $t$ distribution with some known number of degrees of freedom. It is often convenient to transform these $F$ and $t$ variates into normal, or $z$, variates for further analysis.

The obvious approach to transforming an $F$ variate $x$ into a normal variate $z$ is to compute

$$\begin{aligned} p &= P_{F(\nu,\omega)}(x) \\ z &= P_N^{-1}(p) \end{aligned} \tag{1}$$

where $P_{F(\nu,\omega)}(x)$ denotes the cumulative probability function for the $F$ distribution with $(\nu, \omega)$ degrees of freedom and $P_N(x)$ is the cumulative distribution function for the standard normal distribution. Both $P_{F(\nu,\omega)}(x)$ and $P_N^{-1}(p)$ are available as double-precision library functions in many computer languages, including MATLAB (The MathWorks, Inc. 2003).

This approach works well for small to moderate values of $x$, but fails when $x$ becomes too large. The solid line in Figure 1 shows the $z$ values corresponding to $x$ values from 120 to 180 for $(10, 20)$ degrees of freedom, as computed by the formula above using the (double-

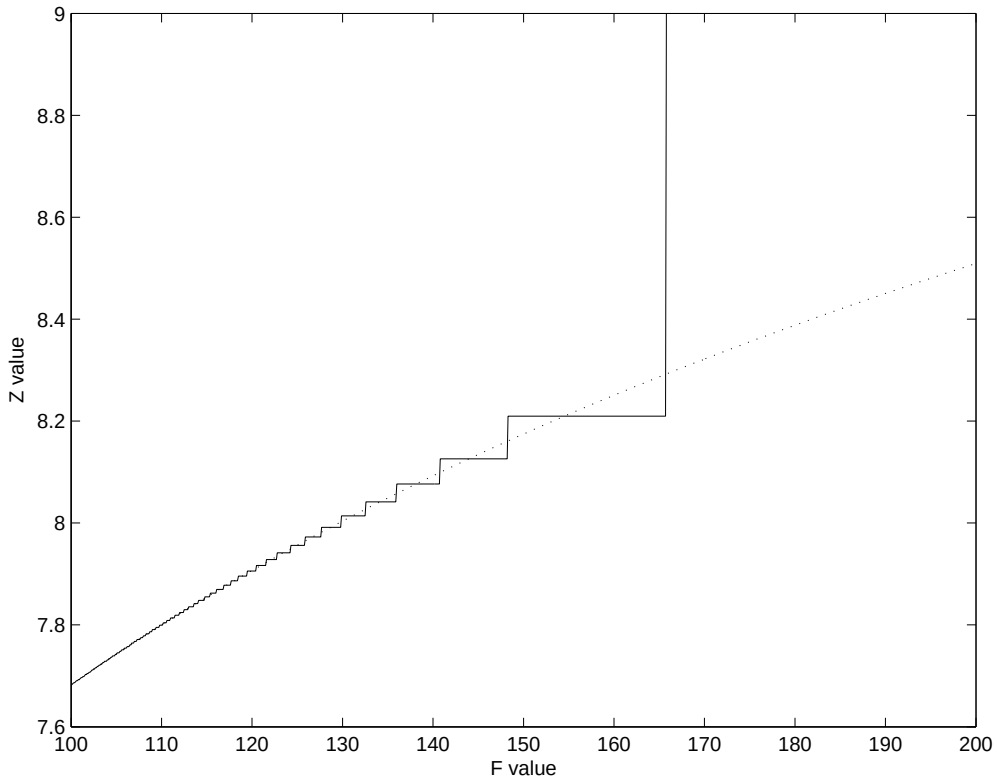


Figure 1: The solid line shows the $F$-to-$z$ transform (for 10,20 degrees of freedom) computed using the naive algorithm; the dashed line shows the same transform computed using the algorithm recommended in this paper.

precision) `fcdf` and `norminv` functions in MATLAB. The mapping shows visible stairstepping for $x > 110$ and goes to infinity at around $x = 166$.

This problem has been observed in $z$ maps generated by SPM for experiments with high statistical power. It appears visually as quantized bands of uniform $z$ value with some maximum $z$ value determined by the number of degrees of freedom.

The source of the problem is found in the limitations of the floating-point number system used in computers. In IEEE 754 floating-point arithmetic (IEEE Standards Board 1985), which is used by nearly all modern computers, double-precision floating point provides about 15 decimal digits of precision. But $P_{F(10,20)}(100) = 1 - 7.8 \times 10^{-15}$ and nearby $p$ values have a useful precision of only 1 or 2 digits; the other 13 digits are all 9's. (IEEE 754 arithmetic is actually binary rather than decimal, but the difference is not essential here.) Furthermore, the only distinct floating-point numbers just below 1 are approximately $1 - 1 \times 10^{-16}, 1 - 2 \times 10^{-16}, 1 - 3 \times 10^{-16}$, et cetera; these $p$ values correspond to $z$ values of $8.2095, 8.1259, 8.0766, \ldots$, which are the ordinates of the last few stairsteps.

The solution is to reformulate the calculation when $x$ is large so as to compute $q = 1 - p$ rather than $p$, since $q$ will be near zero and all significant digits will be available. To do this,

we use the symmetry relationships

$$P_{F(\nu,\omega)}(x) = 1 - P_{F(\omega,\nu)}(1/x) \tag{2}$$

for the $F$ distribution (Johnson, Kotz, and Balakrishnan 1995, p. 322) and

$$P_N(z) = 1 - P_N(-z) \tag{3}$$

for the normal distribution (Johnson, Kotz, and Balakrishnan 1994, p. 81). Using these relationships, the $F$-to-$z$ transformation becomes

$$\begin{aligned} q &= P_{F(\omega,\nu)}(1/x) \\ z &= -P_N^{-1}(q) \quad . \end{aligned} \tag{4}$$

Note that the degrees of freedom $\nu$ and $\omega$ must be interchanged. The dashed line in Figure 1 shows the $F$-to-$z$ transformation computed using this alternate formula.

It should be noted that the alternate formula just derived will show defects similar to that of the original formula when $x$ is very small. If it is necessary to compute the transform accurately for both large and small $x$ values, the original formula (1) should be used for small $x$ and the alternate formula (4) for large $x$. The ideal crossover point is $p = 0.5$, which corresponds to

$$x = P_{F(\nu,\omega)}^{-1}(0.5) \quad . \tag{5}$$

Transforming a $t$ variate into a $z$ variate encounters similar problems for large arguments, as seen in Figure 2 for $\nu = 10$ degrees of freedom, and can be solved in the same manner. In this case, the appropriate symmetry relationship is

$$P_{t(\nu)}(t) = 1 - P_{t(\nu)}(-t) \tag{6}$$

which follows from the fact that the $t$ density function is symmetric around zero (Johnson *et al.* 1995, p. 363). Then the appropriate computation for $t > 0$ is

$$\begin{aligned} q &= P_{t(\nu)}(-t) \\ z &= -P_N^{-1}(q) \quad . \end{aligned} \tag{7}$$

The naive approach

$$\begin{aligned} p &= P_{t(\nu)}(t) \\ z &= P_N^{-1}(p) \end{aligned} \tag{8}$$

remains appropriate for $t < 0$. The ideal crossover point $p = 0.5$ for the $t$-to$-z$ transform corresponds exactly to $t = 0$, and is independent of $\nu$. Figure 2 shows the $t$-to-$z$ transform computed by the naive and alternate approaches, using the `tcdf` and `norminv` functions in MATLAB.

A MATLAB implementation of these algorithms which is efficient for large arrays involves some subtleties. It is very inefficient, by a factor of 10 or more, to loop over the elements of the array and use an `if` statement to select the appropriate formula for each element because of the interpreter overhead for each statement; it is better to create a logical (boolean) array which indicates for each element whether it is large or small, and to use this as an array index

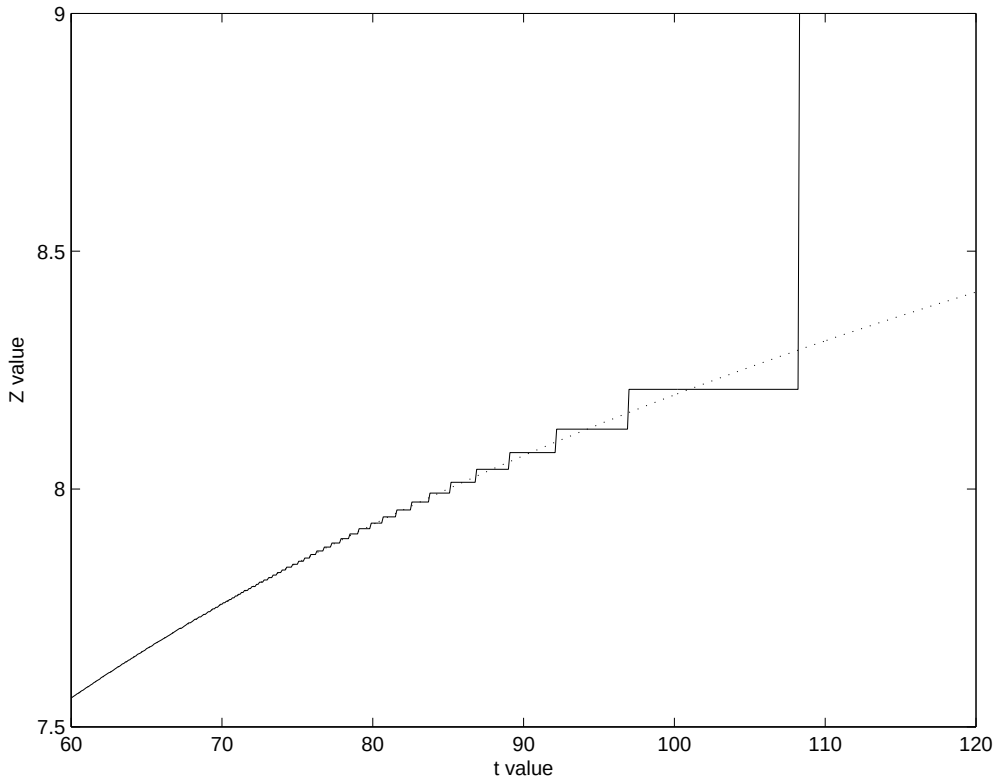


Figure 2: The solid line shows the $t$-to-$z$ transform (for 10 degrees of freedom) computed using the naive algorithm; the dashed line shows the same transform computed using the algorithm recommended in this paper.

to select the appropriate elements for each formula. Blocking the computation to minimize cache traffic can provide a moderate additional improvement in performance (about 30%), but only if the blocking factor is set appropriately for the actual cache size of the platform. Always setting the crossover point (5) in the $F$-to-$z$ calculation to $x = 1.0$ provides about a 4:1 speedup if the number of degrees of freedom is a vector, but will reduce the accuracy, especially if $\nu$ and $\omega$ are very different. The specific efficiency improvements quoted are for an Intel 3.0 GHz Xeon processor running Linux, and can be expected to vary for other platforms.

MATLAB functions that efficiently implement both these transforms are available in the files that accompany this paper. The file `f2z.m` implements an unblocked algorithm for the $F$-to-$z$ transform; `f2z_bloc.m` a blocked algorithm that is more efficient but only if tuned to the particular computer. The file `t2z.m` implements an unblocked algorithm for the $t$-to-$z$ transform, and `t2z_bloc.m` a blocked algorithm.

## Acknowledgments

This work was supported by the National Institutes of Mental Health under grants R01-MH-060722 and P50-MH-064045. The author wishes to thank Ruben Gur and Warren Bilker for their comments on the manuscript.

## References

Frackowiak RSJ, Friston KJ, Frith CD, Dolan RJ, Mazziotta JC (eds.) (1997). *Human Brain Function*. Academic Press, Inc.

IEEE Standards Board (1985). "IEEE Standard for Binary Floating-Point Arithmetic." IEEE Standard 754-1985.

Johnson NL, Kotz S, Balakrishnan N (1994). *Continuous Univariate Distributions - 1*. John Wiley & Sons, second edition.

Johnson NL, Kotz S, Balakrishnan N (1995). *Continuous Univariate Distributions - 2*. John Wiley & Sons, second edition.

The MathWorks, Inc (2003). *MATLAB – The Language of Technical Computing, Version 6.5*. The MathWorks, Inc., Natick, Massachusetts. URL http://www.mathworks.com/products/matlab/.

### Affiliation:

Paul Hughett
University of Pennsylvania
3400 Spruce Street
Philadelphia PA 19104, United States of America
E-mail: hughett@mail.med.upenn.edu

*Journal of Statistical Software* http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 23, Code Snippet 1 *Submitted:* 2007-06-28
November 2007 *Accepted:* 2007-10-27