8-1997

A Generalized Univariate Change-of-Variable Transformation Technique

Andrew G. Glen

Lawrence Leemis
William & Mary, lmleem@wm.edu

John H. Drew

Recommended Citation

Glen, Andrew G.; Leemis, Lawrence; and Drew, John H., A Generalized Univariate Change-of-Variable Transformation Technique (1997). *INFORMS Journal of Computing*, 9(3), 231-318. https://doi.org/10.1287/ijoc.9.3.288

A Generalized Univariate Change-of-Variable Transformation Technique

ANDREW G. GLEN / *Department of Mathematics, College of William & Mary, Williamsburg, VA 23187, Email: agglen@math.wm.edu*

LAWRENCE M. LEEMIS / *Department of Mathematics, College of William & Mary, Williamsburg, VA 23187, Email: leemis@math.wm.edu*

JOHN H. DREW / *Department of Mathematics, College of William & Mary, Williamsburg, VA 23187, Email: jhdrew@math.wm.edu*

We present a generalized version of the univariate change-of-variable technique for transforming continuous random variables. Extending a theorem from Casella and Berger [3] for many–to–1 transformations, we consider more general univariate transformations. Specifically, the transformation can range from 1–to–1 to many–to–1 on various subsets of the support of the random variable of interest. We also present an implementation of the theorem in a computer algebra system that automates the technique. Some examples demonstrate the theorem's application.

Subject classifications: probability

Other key words: change-of-variable technique, computer algebra systems, probability density functions, random variables

In its simplest application, the change-of-variable technique is used to determine the distribution of a continuous random variable Y given the distribution of a continuous random variable X and a 1–to–1 transformation from the support of X to the support of Y. As the conditions on the transformation $Y = g(X)$ become more general, the technique requires more detail in its description and is less likely to appear in introductory probability and statistics texts. Casella and Berger [3, page 51] discuss transforming random variables using the change-of-variable technique when the entire transformation is many–to–1, except for a finite number of points, that is, the cardinality of the set $g^{-1}(y)$ is the same for almost all y in the support of Y. Hogg and Craig [5, page 190] extend this many–to–1 technique to n-dimensional random variables. We are concerned with a more general univariate case in which the transformations are "piecewise many–to–1," where "many" may vary based on the subinterval of the support of Y under consideration. We state and prove a theorem for this case and present code in a computer algebra system to implement the result. Although the theorem is a straightforward generalization of Casella and Berger's, there are a number of details that have to be addressed in order to produce an algorithm for finding the probability density function (pdf) of Y. The resulting computer algebra system implementation of our theorem relieves analysts, researchers, and students from arduous computations.

Consider the following example. Let $f_X(x) = \frac{1}{3}$ for $-1 < x < 2$, be the pdf for the random variable X. Consider the transformation $Y = g(X) = X^2$. This transformation is a 2–to–1 transformation on the interval $X \in (-1, 1)$ (except at $X = 0$) and it is 1–to–1 on the interval $X \in [1, 2)$; see Figure 1. Some introductory probability texts use this as an example (e.g., Larsen and Marx [7, page 137]), but fail to state a general theorem that treats such piecewise many–to–1 transformations. The difficulty lies in identifying an appropriate partition of the support of X and a corresponding partition for the support of Y, and then determining which of these subsets of the support of X correspond to each of the subsets of the support of Y. A further complication is encountered when the transformation itself is either discontinuous or non-differentiable at certain points. For example, consider the random variable X, where $f_X(x) = (x + 1)/18$ for $-1 < x < 5$, and the transformation (see

Figure 3):

$$Y = g(X) = \begin{cases} X^2 & -1 < X < \frac{3}{2} \\ X & \frac{3}{2} < X < 5 \end{cases} .$$

In this example, the transformation is discontinuous as well as "piecewise many–to–1." Our theorem and resulting implementation in a computer algebra system will determine the pdf of Y for such transformations.

In Section 1, we will present our transformation theorem, modeled after Casella and Berger's Theorem 2.1.3 (page 51). Our theorem has been presented more generally in earlier papers. Barr and Zehna [1, page 225] consider multivariate many–to–one transformations. Rohatgi [9, pages 73–74] and Port [8, page 462] consider the piecewise many–to–one case in the univariate and multivariate settings, respectively. Our theorem is strictly univariate, but permits implementation in a computer algebra system. It determines the distribution of $Y = g(X)$ for any univariate random variable X of the continuous type with few restrictions on the transformation $g(X)$. We note that our theorem will be stated in a somewhat elaborate form for the purpose of facilitating its implementation. Section 2 discusses an algorithmic implementation of the theorem using the computer algebra system capabilities of the software package Maple V. Section 3 illustrates the usefulness of the algorithm.

1 Theorem

Before we present the theorem, we overview the rationale for the notation. We assume that the support of X, denoted by $\mathcal{X}$, consists of a finite union of open intervals. The points $x_1 < x_2 < \cdots < x_{n+1}$ generate n consecutive subintervals and are determined as follows. The first subinterval of $\mathcal{X}$ begins with x_1 and the last subinterval of $\mathcal{X}$ ends with x_{n+1}. The remaining x_i's correspond to other endpoints of the intervals of $\mathcal{X}$ or to the locations in $\mathcal{X}$ where the function g is discontinuous or non-differentiable, or g' is zero. Let $g_i(x)$ denote the restriction of $g(x)$ to (x_i, x_{i+1}); by design, g_i is monotone and therefore invertible. Let $X^\star = \{x_1, x_2, \ldots, x_{n+1}\}$; note that since X is continuous, $P(X \in X^\star) = 0$.

We introduce $Y^\star$, a set of points on the y-axis, to partition the support of Y into subintervals. The range of each g_i, denoted by (m_i, M_i), either contains or is disjoint from each

Y subinterval. The set $Y^\star$ is designed so that the final pdf of Y may be specified by designating its value on each Y subinterval. The set $Y^\star$ is defined using one-sided limits because the transformations $g_1, g_2, \ldots, g_n$ are not necessarily defined at the points in $X^\star$. The set I_j consists of integers i with the property that the range of g_i is either equal to or properly contains the jth Y subinterval. The pdf for each Y subinterval depends on the pdf of X and the associated transformations g_i.

Theorem. Let X be a random variable of the continuous type with pdf $f_X(x)$ and with support $\mathcal{X}$, where $\mathcal{X}$ consists of a finite union of open intervals. Let $g(x)$ be a real-valued function whose domain includes $\mathcal{X}$. Let $-\infty \leq x_1 < x_2 < \cdots < x_n < x_{n+1} \leq +\infty$ be a sequence of extended real numbers which satisfy the following conditions:

1. The sequence includes the endpoints of the intervals whose union is $\mathcal{X}$.

2. $f_X(x)$ is continuous on each open interval $A_i = (x_i, x_{i+1})$ for $i = 1, 2, \ldots, n$.

3. If $f_X(x)$ is not identically zero on A_i, then the function $g_i(x)$, which is the restriction of $g(x)$ to A_i, is monotone on A_i and has a nonzero derivative at each point in A_i, for $i = 1, 2, \ldots, n$.

Let $X^\star = \{x_1, x_2, \ldots, x_{n+1}\}$.
Let $\alpha = \{i | f_X(x) \text{ is not identically zero on } A_i\}$.
Let $m_i = \min\left\{\lim_{x \downarrow x_i} g(x), \lim_{x \uparrow x_{i+1}} g(x)\right\}$ for $i = 1, 2, \ldots, n$.
Let $M_i = \max\left\{\lim_{x \downarrow x_i} g(x), \lim_{x \uparrow x_{i+1}} g(x)\right\}$ for $i = 1, 2, \ldots, n$.
Let $Y^\star = \cup_{i \in \alpha} \{m_i, M_i\}$.
Let $m = \| Y^\star \| - 1$, where $\| \cdot \|$ denotes cardinality.
Order the elements of y_j of $Y^\star$ so that $y_1 < y_2 < \cdots < y_{m+1}$.
Let $I_j = \{i | m_i \leq y_j \text{ and } y_{j+1} \leq M_i\}$, for $j = 1, 2, \ldots, m$.
Then for $y \in (y_j, y_{j+1})$,

$$f_Y(y) = \sum_{i \in I_j} f_X\left(g_i^{-1}(y)\right) \left| \frac{dg_i^{-1}(y)}{dy} \right|$$

for $j = 1, 2, \ldots, m$.

Proof. Without loss of generality, consider the jth Y subinterval (y_j, y_{j+1}). Also suppose that a and b are any points that lie inside (y_j, y_{j+1}) such that $a < b$. Furthermore, let $M_i = \max\{g_i^{-1}(a), g_i^{-1}(b)\}$ and $m_i = \min\{g_i^{-1}(a), g_i^{-1}(b)\}$ for $i \in I_j$. As Hogg and Craig [5, page 190] point out,

$$P(a < Y < b) = \sum_{i \in I_j} P\Big(m_i < X < M_i\Big).$$

Since Y is a continuous random variable,

$$\begin{aligned} P(a < Y < b) &= \sum_{i \in I_j} \int_{m_i}^{M_i} f_X(x)\ dx \\ &= \sum_{i \in I_j} \int_a^b f_X\Big(g_i^{-1}(y)\Big) \left|\frac{dg_i^{-1}(y)}{dy}\right| dy \\ &= \int_a^b \sum_{i \in I_j} f_X\Big(g_i^{-1}(y)\Big) \left|\frac{dg_i^{-1}(y)}{dy}\right| dy \end{aligned}$$

where we performed the change-of-variable $y = g_i(x)$, or equivalently $x = g_i^{-1}(y)$, and have used the absolute value to treat both increasing and decreasing transformations (see [4, page 268]). Referring to Theorem 7.1 in Freund [4], we see that the integrand

$$\sum_{i \in I_j} f_X\Big(g_i^{-1}(y)\Big) \left|\frac{dg_i^{-1}(y)}{dy}\right|$$

is the pdf for Y on the subinterval (y_j, y_{j+1}). ■

2 Implementation

It could be quite tedious to implement this theorem by hand for large m and n. With the onset of computer algebra systems such as Maple V, however, one may implement algorithms that correspond to theorems such as ours. Our implementation in Maple code is included in the appendix and parallels the theorem. Two main implementation issues emerged. First, Maple may produce several candidates for g_i^{-1}, e.g., when $g_i(x) = x^2$, Maple returns $g_i^{-1}(y) = -\sqrt{y}$ and $g_i^{-1}(y) = \sqrt{y}$. The correct inverse is selected by requiring that $g_i^{-1}(g_i(c_i)) = c_i$, where c_i is any point in the ith X subinterval. When the ith X subinterval has finite endpoints, c_i can be chosen to be the midpoint of the subinterval. In the cases where $x_1 = -\infty$ or $x_{n+1} = \infty$ [e.g., $X \sim N(\mu, \sigma^2)$], c_i must be selected more carefully. The algorithm for determining c_i is:

1. If $x_1 = -\infty$ and $x_2 = +\infty$, then $c_1 = 0$.

2. If $x_1 = -\infty$ and $x_2 \neq +\infty$, then $c_1 = x_2 - 1$.

3. If $x_n \neq -\infty$ and $x_{n+1} = +\infty$, then $c_n = x_n + 1$.

4. For all other cases, $c_i = \dfrac{x_i + x_{i+1}}{2}$.

The second implementation issue involves redundancies in lists. Maple doesn't recognize that the integer 3 and the floating-point value 3.0, for example, are redundant in a list. The `ReduceList` procedure finds all adjacent points that differ by a small prescribed δ and discards redundant floating-point type elements.

The data structure used to represent the distribution of a random variable is a "list of three lists." The first sub-list has as elements the functions that comprise the piecewise continuous portions of the pdf. The second sub-list has as elements the ordered support points of the distribution that delineate the beginning and end of the domain for each of the functions in the first sub-list. The third sub-list contains the string "p" which denotes that the function in the first sub-list is a probability density function (rather than a cdf or a hazard function). The data structure for g is a "list of two lists" that is similar in nature to that of f. Here we characterize g by listing its monotone components and the endpoints of the corresponding domains. Example 3.1 gives a detailed illustration of the data structures for f and g.

The following are some additional implementation issues that arose during the coding of the algorithm:

- The user must supply $x_1, x_2, \ldots, x_{n+1}$. This ordered list consists of the endpoints of the open intervals which constitute $\mathcal{X}$ and all locations in $\mathcal{X}$ where $g(x)$ is discontinuous or non-differentiable, or $g'(x)$ is zero. A preprocessor to determine elements of $X^\star$ corresponding to $g'(x) = 0$ could be added to the algorithm if desired. Thus if the support of X is $(-1, 2)$, and $Y = g(X) = X^2$, this preprocessor would include $x = 0$ in $X^\star$ yielding $X^\star = \{-1, 0, 2\}$.

- The partition points $x_1, x_2, \ldots, x_{n+1}$ must be chosen so that $f_X(x)$ is not defined piecewise on any X subinterval (e.g., if X has a triangular distribution, the mode must be

a partition point).

- Adding extra x-values in $\mathcal{X}$ to $X^\star$ that are not maxima, minima or saddle points of $g(x)$ or discontinuities of $f_X(x)$ or $g(x)$ will not affect the correctness of the algorithm's implementation of the theorem, although the implementation will be slower.

- Many transformations such as $g(x) = x + e^x$ do not have an inverse that can be expressed in a closed-form. Numerical methods could be used in conjunction with the algorithm to find individual values of $f_Y(y)$. We have not implemented this in our code. If $g(x) = x + e^x$, for example, then

$$\left|\frac{dg^{-1}(y)}{dy}\right| = \left.\frac{1}{1+e^x}\right|_{x=g^{-1}(y)}.$$

- Periodic transformations can present a problem in the implementation of the theorem. Maple uses the usual principal inverses for $\sin x$, $\cos x$, and $\tan x$, but the restricted ranges of these principal inverses are often inappropriate. When the algorithm tries to identify which inverse is appropriate, it is only given one choice, the principal inverse, but the domain of X might not coincide with the range of the principal inverse. Example 3.4 illustrates the problem and a circumvention.

- No error-trapping has been done to insure that $f_X(x)$ is a legitimate pdf, i.e.

$$\int_{-\infty}^{\infty} f_X(x)dx = 1, f_X(x) \geq 0 \ \forall \ x.$$

  Also, there is limited error-trapping on $g(x)$ in that the procedure gives an error message when the inverse cannot be found.

- Parameters are allowed in $f_X(x)$ but not in $g(x)$.

3 Examples

This section contains some illustrative examples of using the algorithm described in the previous section to determine the distribution of $Y = g(X)$. Before using the program, we tested its performance on several common transformations. Given $X \sim N(0,1)$ and

$Y = g(X) = X^2$, the program returned the pdf for a χ^2_1 random variable. Given $X \sim N(\mu, \sigma^2)$ and $Y = g(X) = (X - \mu)/\sigma$, the program returned the pdf for a standard normal random variable. Now we will consider more complex examples that illustrate the theorem's implementation.

Example 3.1 Consider the first example from the introduction: given the random variable $X \sim U(-1, 2)$, find the distribution of $Y = g(X) = X^2$. Given $X^\star$, the algorithm determines the relevant partitions of the supports of X and Y. Then it determines which X subintervals map onto which Y subintervals. The subintervals are apparent in Figure 1. Note the set $X^\star$ is displayed on the

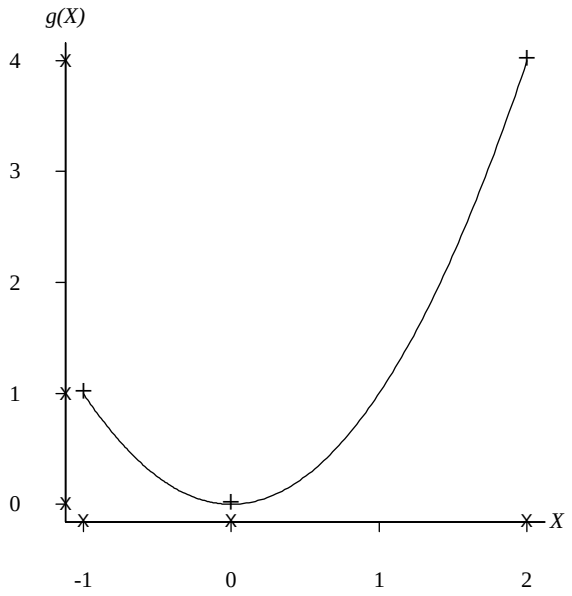

Figure 1: The transformation $Y = g(X) = X^2$ for $-1 < X < 2$.

horizontal axis, $Y^\star$ on the vertical axis, both marked with the $\times$ symbol. The transformation is partitioned into monotone segments (with identical or disjoint ranges) delineated by the $+$ symbol. The assignment of data structures for f and g and the call to `Transform` are as follows:

```
fx := [ [x -> 1 / 3], [-1, 2], [p] ];
gx := [ [x -> x ^ 2, x -> x ^ 2], [- infinity, 0, infinity] ];
```

```
fnew := Transform(fx, gx);
```

The program determines that the transformation is 2–to–1 on $-1 < x < 1$ (excluding $x = 0$) and 1–to–1 on $1 \leq x < 2$. Since $Y = X^2$ has two inverses, the program determines which inverse to apply to which X subinterval. The resulting pdf for Y is

$$f_Y(y) = \begin{cases} \frac{1}{3\sqrt{y}} & 0 < y < 1 \\ \frac{1}{6\sqrt{y}} & 1 < y < 4 \end{cases} .$$

The corresponding returned value of `fnew` is a "list of three lists" that specifies the pdf of Y:

```
[[y -> 1 / (3 * sqrt(y)), y -> 1 / (6 * sqrt(y))], [0, 1, 4], [p]].
```

Example 3.2 Consider the random variable $X \sim U(0, 7)$. Find the pdf of $Y = g(X) = ||X - 3| - 1|$. A graphical representation of this transformation is shown

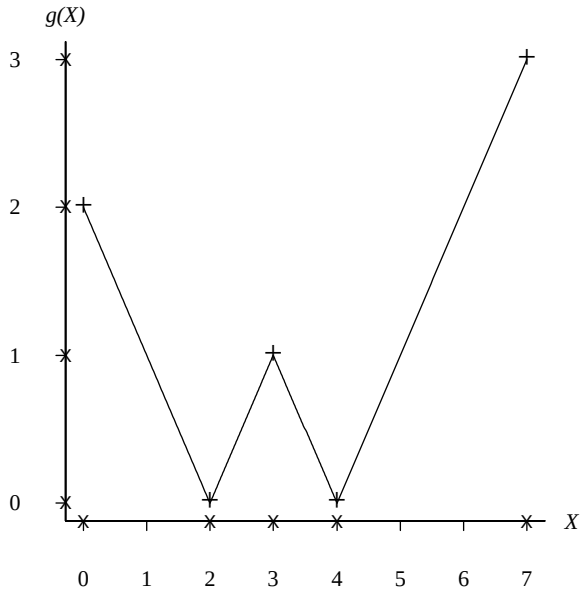

Figure 2: The transformation $Y = g(X) = ||X - 3| - 1|$ for $0 < X < 7$.

in Figure 2, with $X^\star$, $Y^\star$, and the monotone segments of g marked as before. This transformation is more complex than Example 3.1 in that it is 4–to–1 for

$0 < y < 1$, 2–to–1 for $1 < y < 2$, and 1–to–1 for $2 < y < 3$. The program yields the correct pdf for Y:

$$f_Y(y) = \begin{cases} \frac{4}{7} & 0 < y < 1 \\ \frac{2}{7} & 1 < y < 2 \\ \frac{1}{7} & 2 < y < 3 \end{cases} .$$

Example 3.3 Consider the second example from the introduction: if the random variable X has pdf $f_X(x) = (x+1)/18$ for $-1 < x < 5$, find the distribution of

$$Y = g(X) = \begin{cases} X^2 & -1 < X < \frac{3}{2} \\ X & \frac{3}{2} < X < 5 \end{cases} .$$

The appropriate partition for the transformation is depicted in Figure 3. The

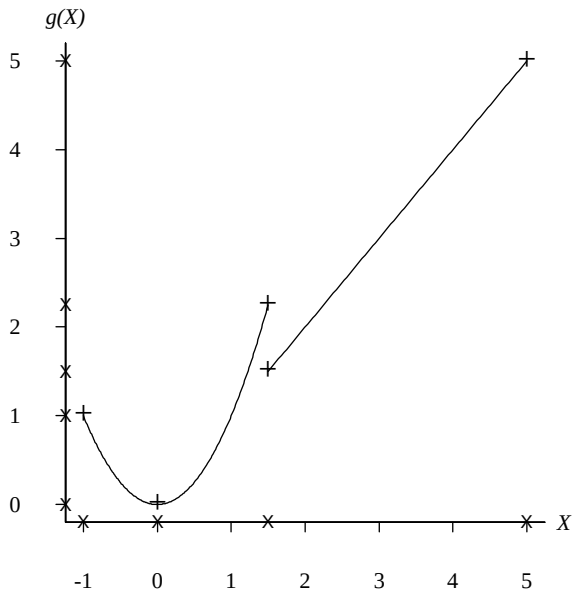

Figure 3: The transformation $Y = g(X)$ has a discontinuity and is variously 2–to–1 and 1–to–1 on different subintervals.

program determines the following pdf for Y:

$$f_Y(y) = \begin{cases} \frac{1}{18\sqrt{y}} & 0 < y < 1 \\ \frac{\sqrt{y}+1}{36\sqrt{y}} & 1 < y < 1.5 \\ \frac{(2y+3)\sqrt{y}+1}{36\sqrt{y}} & 1.5 < y < 2.25 \\ \frac{y+1}{18} & 2.25 < y < 5 \end{cases} .$$

Example 3.4 As a final example, we will consider the problem that Casella and Berger [3] discussed, without providing $f_Y(y)$, as a prelude to their theorem. Letting X be a uniform random variable on $(0, 2\pi)$, find the distribution of $Y = g(X) = \sin^2(X)$. As Figure 4 shows, this transformation is 4–to–1 for the single Y subinterval $(0, 1)$. Furthermore, since $n = 4$, $X^\star = \{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi\}$. The pdf of

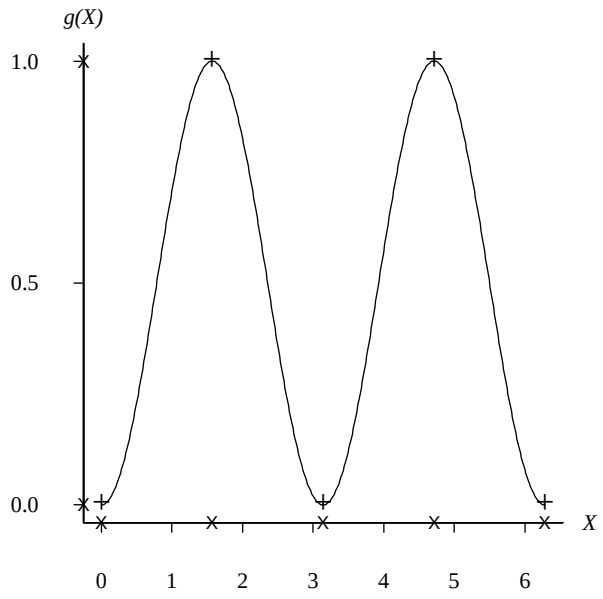

Figure 4: The transformation $Y = g(X) = \sin^2(X)$ for $0 < X < 2\pi$.

Y is

$$f_Y(y) = \frac{1}{\pi\sqrt{y - y^2}} \qquad 0 < y < 1$$

which is commonly known as the arcsin distribution (see Johnson, Kotz, and Balakrishnan [6, page 212]). To overcome the principal inverse difficulty alluded to in the previous section, we devised the following equivalent situation: consider the random variable $X \sim U(-\frac{\pi}{2}, \frac{\pi}{2})$ and let $Y = g(X) = \sin^2(X)$. In this case the domain of X will be the same as the range of g^{-1} which results from using the standard arcsin function.

This solution yielded a random variable with a notable feature. The distribution's hazard function, $h_Y(y) = \dfrac{f_Y(y)}{1 - F_Y(y)}$, might be of interest to a reliability engineer.

For this distribution,

$$h_Y(y) = \frac{2}{\sqrt{y - y^2}\,[\pi - 2\,\arcsin(2y-1)]} \qquad 0 < y < 1.$$

By plotting this hazard function, we see it has the rare property of a closed-form, "bathtub"-shaped hazard function. Furthermore, we can now apply the transformation $W = g(Y) = \lambda Y$ in order to derive a random variable W that is a one-parameter random variable with a closed-form, bathtub-shaped hazard function. To our knowledge, although the arcsin distribution has been discussed in the literature, we have yet to find mention of the fact that it has a bathtub-shaped hazard function. Most distributions with bathtub-shaped hazard functions must be analyzed numerically because their hazard functions are not closed form. One of the useful features of the `Transform` procedure is that it can help the practitioner gain insight into what transformations could result in a useful model. In this example, it becomes apparent that the transformation "crowds" the uniformly distributed X random variable into a distribution for Y that is "heavy" on either end of its support. The result is a random variable with a bathtub-shaped hazard function. The program enables the model designer to confirm, or gain insight into the univariate transformations that could result in a useful model.

4 Conclusion

We envision more for this Maple implementation of a probability theorem than what is illustrated in these examples. The tool provided by the `Transform` procedure will be useful to the practitioner as well as the academic. The academic might use this tool to find new distributions. Let X be a beta random variable with specified parameters, for example, and let $g(X) = X^2$. Applying g several times in succession may yield a distribution that is useful in modeling. Alternatively, students might be asked to combine a set of ten transformations and ten distributions in order to create 100 new distributions, picking out the interesting properties of the more notable ones.

The practitioner, on the other hand, may use the `Transform` procedure for the specific purpose of iterative probabilistic model design, illustrated in Example 3.4. For example, the

`Transform` procedure could be part of an optimization algorithm that finds the constants a and b, where $Y = g(X) = aX + b$, yielding the optimal linear transformation of X in terms of some measure of performance in analysis problems where a linear transformation of data is appropriate.

Our current intent for the procedure `Transform` is to include it in a package of probabilistic procedures that would handle many types of analysis for the practitioner. We now have, for example, the procedures `Convolute` [2], `ProductOf`, `Truncate`, `OrderStat`, `PdfOf`, `CdfOf`, `MinOf`, and `MaxOf`. The combined effect of these procedures allows one to model with previously untenable distributions that were disregarded as too complex to analyze. We are currently using these functions in the following areas of probabilistic modeling: optimizing life tests (both censored and uncensored non-exponential populations); determining exact distributions in renewal theory and reliability block diagrams; and, minimizing the use of the CLT by using the actual distributions of convolutions of random variables.

Acknowledgements

The authors gratefully acknowledge helpful comments from Jacques Carette, Gianfranco Ciardo, the Area Editor Carl Harris, two referees, the students in a Mathematical Statistics course at William & Mary, and support from a William & Mary summer research grant.

References

[1] D. Barr and P.W. Zehna, 1971. *Probability,* Brooks/Cole, Inc., Belmont, California.

[2] R. Berger, 1995. personal communication.

[3] G. Casella and R. Berger, 1990. *Statistical Inference,* Wadsworth and Brooks/Cole, Inc., Pacific Grove, California.

[4] J. Freund, 1992. *Mathematical Statistics,* Fifth edition, Prentice–Hall, Englewood Cliffs, New Jersey.

[5] R.V. Hogg and A.T. Craig, 1995. *Mathematical Statistics,* Fifth edition, Prentice–Hall, Englewood Cliffs, New Jersey.

[6] N.L. Johnson, Kotz, S., and Balakrishnan, N., 1995. *Continuous Univariate Distributions, Volume 2*, Second edition, John Wiley & Sons, New York.

[7] R.J. Larsen and M.L. Marx, 1986. *An Introduction to Mathematical Statistics and Its Applications,* Second edition, Prentice–Hall, Englewood Cliffs, New Jersey.

[8] S.C. Port, 1994. *Theoretical Probability for Applications,* John Wiley & Sons, New York.

[9] V.K. Rohatgi, 1976. *An Introduction to Probability Theory Mathematical Statistics,* John Wiley & Sons, New York.

Appendix: Maple Code

```
#
# ReduceList is a procedure that eliminates floating point redundancies (e.g.,
# 3 vs. 3.0) from a sorted Maple list.
#
ReduceList := proc(LST)
local i, size, delt, deltamin, ListIn:
deltamin := 0.0000001:

ListIn := LST:
size := nops(ListIn):
for i from (size - 1) by -1 to 1 do
  if (ListIn[i] <> -infinity and ListIn[i + 1] <> infinity) then
    delt := evalf(ListIn[i + 1]) - evalf(ListIn[i]):
    if (delt < deltamin) then
      if (whattype(ListIn[i]) <> float) then
        ListIn := subsop((i + 1) = NULL, ListIn):
      else
        ListIn := subsop(i = NULL, ListIn):
      fi:
    fi:
  fi:
od:
RETURN(ListIn):
end:
#
# Procedure Transform finds the pdf of Y = g(X), where X and Y are continuous
# random variables.  The arguments are f(x), the pdf of X, and g(X), the
# transformation.  The arguments of Transform and the returned value are in
# the 'list of lists' format.
#
Transform := proc(fX, gX)
local nx, XStarSet, ng, XStarList, n, FF, i, k, c, BEGIN, END, gndx, j, mn, Mx, a,
      b, YStarSet, YStarList, m,  ffY, h, ntmp, temp, gtemp, ii, t, itp, ginv, fY:

nx := nops(fX[2]):
XStarSet := {op(fX[2])}:

ng := nops(gX[2]):
for i from 1 to ng do
  if (evalf(gX[2][i]) > evalf(fX[2][1]) and evalf(gX[2][i]) < evalf(fX[2][nx]))
      then
```

```
    XStarSet := XStarSet union {gX[2][i]}
  fi:
od:

XStarList := sort([op(XStarSet)], (x, y) -> evalb(evalf(x) < evalf(y))):
XStarList := ReduceList(XStarList):
n := nops(XStarList) - 1:

FF := array(1 .. n):

# Find the appropriate index k of fX[1][k] for each x interval.
for i from 1 to n do
  for k from 1 to nx do
    if (evalf(XStarList[i]) >= evalf(fX[2][k])) then
      FF[i] := k:
    fi:
  od:
od:

c := array(1 .. n):

if (XStarList[1] = -infinity and XStarList[2] = infinity) then
  c[1] := 0
else
  BEGIN := 1:
  END := n:
  if (XStarList[1] = -infinity) then
    c[1] := XStarList[2] - 1:
    BEGIN := 2:
  fi:
  if (XStarList[n + 1] = infinity) then
    c[n] := XStarList[n] + 1:
    END := n - 1:
  fi:
  for i from BEGIN to END do
    c[i] := (XStarList[i] + XStarList[i + 1]) / 2:
  od:
fi:

ng := nops(gX[1]):
gndx := array(1 .. n):

# Find the appropriate index j of gX[1][j] for each x interval.
for i from 1 to n do
  for j from 1 to ng do
```

```
    if (evalf(XStarList[i]) >= evalf(gX[2][j]) and evalf(XStarList[i]) <
        evalf(gX[2][j + 1])) then
      gndx[i] := j:
      break:
    fi:
  od:
od:

mn := array(1 .. n):
Mx := array(1 .. n):
for i from 1 to n do
  a := limit(gX[1][gndx[i]](x), x = XStarList[i], right):
  b := limit(gX[1][gndx[i]](x), x = XStarList[i + 1], left):
  mn[i] := min(a, b):
  Mx[i] := max(a, b):
od:
YStarSet := {seq(mn[i], i = 1 .. n), seq(Mx[i], i = 1 .. n)}:
YStarList := sort([op(YStarSet)], (x, y) -> evalb(evalf(x) < evalf(y))):
YStarList := ReduceList(YStarList):
m := nops(YStarList) - 1:

ffY := []:
for j from 1 to m do
  h := 0:
  for i from 1 to n do
    # If we are working with a relevant transformation segment, adjust the pdf.
    if (evalf(mn[i]) <= evalf(YStarList[j]) and evalf(YStarList[j + 1]) <=
        evalf(Mx[i])) then
      temp := [solve(gX[1][gndx[i]](t2) = y, t2)]:
      ntmp := nops(temp):
      gtemp := array(1 .. ntmp):
      for ii from 1 to ntmp do
        gtemp[ii] := unapply(temp[ii], y):
      od:
      if (ntmp = 1) then
        # If there is only one inverse, assign it to ginv.
        ginv := gtemp[1](y):
      else
        # If there is more than one inverse, find the appropriate one.
        for t from 1 to ntmp do
          # Evaluate each inverse of the transformation at the midpoint of the
          # subinterval.  Set ginv to that inverse for which gtemp(g(x)) = x.
          itp := evalf(gtemp[t](gX[1][gndx[i]](evalf(c[i])))):
          if (itp > evalf(XStarList[i]) and itp < evalf(XStarList[i + 1])) then
            ginv := gtemp[t](y):
```

```
            break:
          else
            ginv := notfound:
          fi:
        od:
        if (ginv = notfound) then
           print(`ERROR, GINVERSE not found`):
        fi:
      fi:

      # For the case of an increasing transformation segment, add the change.
      if (evalf(limit(gX[1][gndx[i]](xx), xx = XStarList[i], right))
            < evalf(limit(gX[1][gndx[i]](xx), xx = XStarList[i + 1], left))) then
        h := h + fX[1][FF[i]](ginv) * diff(ginv, y):
      # For the case of a decreasing transformation segment, subtract the change.
      else
        h := h - fX[1][FF[i]](ginv) * diff(ginv, y):
      fi:
    fi:
  od:
  ffY := [op(ffY), op([unapply(simplify(h), y)])]:
od:

fY[1] := [op(ffY)]:
fY[2] := [op(YStarList)]:
fY[3] := ['p']:
RETURN(fY):
end:
```