ACTUARIAL RESEARCH CLEARING HOUSE
1995 VOL. 1

Manipulating Lagrangian distributions and associated compound distributions with Maple

Rohana S. Ambagaspitiya
Department of Mathematics and Statistics
University of Calgary
Calgary, Alberta
T2N 1N4, Canada

Abstract

Applications of Lagrangian distributions to modelling claim frequency data in an insurance portfolio is a relatively new concept. The major difficulty is that the generating functions of these distributions cannot be expressed in terms of elementary functions. Also deriving moments and/or cumulants is somewhat tedious. This article illustrates how to use Maple effectively to overcome these difficulties.

Keywords:

Lagrangian distributions, Binomial distribution, Lambert's W function, Negative Binomial distribution, Poisson distribution.

1 Introduction

Lagrangian distributions of the first kind have probability functions of the form:

$$\begin{aligned} \Pr[X=0] &= L(g;f;0) = f(0) \\ \Pr[X=x] &= L(g;f;x) = \frac{1}{x!}\frac{d^{x-1}}{dt^{x-1}}\{(g(t))^x f'(t)\}|_{t=0}, \qquad x=1,2,\ldots, \end{aligned} \tag{1.1}$$

and Lagrangian distributions of the second kind have probability functions of the form:

$$\Pr[X=x] = \frac{1-g'(1)}{x!}\frac{d^x}{dt^x}\{(g(t))^x f(t)\}|_{t=0}, \quad x=0,1,2,\ldots, \tag{1.2}$$

where $g(t)$ and $f(t)$ are two probability generating functions (pgf) defined on nonnegative integers such that $g(0) \neq 0$; different families are generated by making different choices of $g(t)$ and $f(t)$. This method of obtaining discrete distributions has been used for many years since Otter's (1949) multiplicative process. The potential of this technique has been systematically exploited by Consul and his co-workers since the formal introduction of Lagrangian distributions of the first kind in Consul and Shenton (1972). The Lagrangian distributions of the second kind were formally introduced by Janadaran and Rao (1983). Recently, a few members of these families have been used to model the claim frequency data of an insurance portfolio (see Goovaerts and Kaas (1991), Kling and Goovaerts (1993), Ambagaspitiya

and Balakrishnan (1994)). In general these distributions are difficult to handle, their pgf's can not be expressed in terms of known functions, elementary or otherwise. Consul and Shenton (1972) provide a mechanism to obtain moments and cumulants, but it is somewhat tedious. In this article first we present an attractive way to obtain moments and cumulants of Lagrangian distributions using Maple. Then we discuss the manipulation of compound Lagrangian distributions, which can be used to model the total claim amount of an insurance portfolio, with the aid of Maple. Also we provide two inverse functions, inspired by Lambert's W functions, that can be used to manipulate two important classes of distributions in the Lagrangian families.

2 Lagrangian distributions

The first step towards obtaining Lagrangian families is to solve

$$t = zg(t) \tag{2.1}$$

for the numerically smallest real root t in terms of z. Let us denote the function which yields the required value of t in (2.1) for various values of z by $h(z)$. Except in a few cases $h(z)$ cannot be expressed in terms of elementary functions. Consul and Shenton (1972) have shown that $h(z)$ is a pgf of a non-zero integer value random variable; i.e. $h(0) = 0$ and $h(1) = 1$. From the definition of Lagrange distributions it can be shown that the pgf of Lagrangian distributions of the first kind has the form:

$$P_1(z) = f(h(z)), \tag{2.2}$$

and the pgf of Lagrangian distributions of the second kind has the form:

$$P_2(z) = (1 - g'(1))\frac{f(h(z))}{1 - zg'(h(z))} \tag{2.3}$$

The moment generating function $M_i(t)$ is

$$M_i(t) = P_i(\exp(t)), \qquad i = 1, 2,$$

and the cumulant generating function $C_i(t)$ is

$$C_i(t) = \log M_i(t), \qquad i = 1, 2.$$

One method of obtaining moments and cumulants is to differentiate the respective generating functions successively with respect to t and evaluate the derivatives at $t = 0$. In the general case functional form of $h(z)$ is unknown, but if our interest is only to calculate moments and cumulants we need to evaluate $h(z)$ and it is derivatives at $z = 1$. By differentiating (2.1) with respect to z and simplifying we have

$$\frac{dh(z)}{dz} = \frac{h(z)}{z(1 - zg'(h(z)))}. \tag{2.4}$$

We have programmed this information in Maple and the source codes are given in the section 1 of Appendix C. We have calculated moments and cumulants of all known Lagrangian distributions and we are happy to provide them if anyone is interested.

2.1 Poisson Lagrangian distributions

These distributions are derived by taking the pgf of a Poisson distribution with mean b, $0 \leq b < 1$, as $g(t)$. i.e.

$$g(t) = \exp(b(t-1)).$$

In this case it can be easily shown that

$$h(z) = -\frac{W(-b\exp(b)z)}{b},$$

where W is the main branch of Lambert's W function. A detailed account of Lambert's W function and its properties can be found in Coreless *et al.* (1994). Since Maple can handle the function $W(x)$ and it is derivatives, we can use Maple to obtain moments and cumulants of Poisson Lagrangian distributions. Also since Maple library function **evalf/W** can evaluate $W(x)$ for given x we can compute generating functions of Poisson Lagrangian distributions.

2.2 Binomial Lagrangian distributions

These distributions are derived by taking pgf of a binomial distribution with parameters b and p as $g(t)$. i.e.

$$g(t) = (1-p+pt)^b, \quad 0 \leq p \leq 1.$$

The resulting function $h(z)$ is

$$h(z) = \frac{1-p}{p} IB(b, p(1-p)^{b-1}z),$$

where $IB()$ is the inverse function defined in the Appendix A. Although in the binomial distribution the parameter b is an integer, Consul and Shenton (1972) have shown that binomial Lagrangian distributions exist for any $1 \leq b < 1/p$. Using the Maple procedure given in the Appendix A we can calculate moments and cumulants of binomial Lagrangian distributions. Also we can use the procedure **evalf/IB** to compute the generating functions of binomial Lagrangian distributions at any given point.

2.3 Negative binomial Lagrangian distributions

These distributions are derived by taking the pgf of a negative binomial distribution with parameters b and P as $g(t)$ i.e.

$$g(t) = \frac{1}{(1+P-Pt)^b}, \quad b > 0, \quad bP < 1,$$

resulting function $h(z)$ is

$$h(z) = \frac{1+P}{P} IG\left(b, \frac{P}{(1+P)^{b+1}}z\right),$$

where the function $IG()$ is the inverse function as defined in Appendix B. We can use the procedure given in the Appendix B to evaluate the derivatives of $IG()$ and hence to obtain the moments and cumulants of negative binomial Lagrangian distributions. Also one could use the procedure **evalf/IG** to compute the generating functions at any given point.

3 Compound Lagrangian distributions

Let N be the number of claims produced by an insurance portfolio and let X_i be the ith claim amount. Then the total claim amount S produced by the portfolio is given by

$$S = X_1 + X_2 + \ldots + X_N. \tag{3.1}$$

In risk theory two fundamental assumptions are made and they are:

1. $X_1, X_2, \ldots$ are independently and identically distributed random variables.
2. N is a discrete random variable independent of X_i, $i = 1, 2, \ldots$.

In the actuarial literature the random variable S is said to have a compound distribution. Based on these assumptions the mgf of S can be written as:

$$M_S(t) = P_N(M_X(t)), \tag{3.2}$$

where $P_N(t)$ is the pgf of the claim number distribution and $M_X(t)$ is the mgf of the claim amount distribution. For a proof of this result see Chapter 2 of Panjer and Willmot (1992). When a member of Lagrangian family is used to model the number of claims the resulting distribution is called a compound Lagrangian distribution. The moments and cumulants of compound Lagrangian distributions can be obtained using the Maple codes given in the section 1 of Appendix C. In some cases it will be necessary to determine the density function $f_S(x)$ of S in addition to moments. One way of obtaining the density function is to invert the Laplace transform

$$\mathcal{L}\{f_S(x)\} = M_S(-t),$$

using numerical techniques. For this it is essential to calculate the Laplace transform at various values of t. With the aid of Maple library function **evalf/W** and the two inverse function **evlaf/IB** and **evalf/IG** given in this paper we can calculate the Laplace transform of compound Poisson Lagrangian , compound binomial Lagrangian and compound negative binomial Lagrangian distributions. At present we are investigating the usage of Maple for numerical inversion of Laplace transforms of compound Lagrangian distributions and our findings will be submitted as a separate article.

4 Conclusion

In this article we have presented an attractive way to obtain moments and cumulants of Lagrangian distributions and compound Lagrangian distributions. Also we have presented two inverse functions, inspired by Lambert's W function, that can be valuable additions to the Maple library. However the Maple library function **solve** cannot be modified easily to give solutions in terms of these functions in appropriate situations. Note that the procedures $IB(m, x)$ and $IG(m, x)$ work in the Maple V Release 3 but not in the Release 2, due to the fact that there are some differences in expression representations in two versions.

For numerical evaluation of generating functions of Lagrangian distributions and compound Lagrangian distributions, one need to solve $t = zg(t)$ for t numerically. Although it is possible to use the Maple library function **fsolve** to obtain t for given z, it would be more efficient to implement the best iterative technique for the particular problem. If the **fsolve** procedure did allow user written procedures to be invoked in specific cases, such as, for example, the **diff** procedure, it would greatly reduce the user's programming burden.

A The inverse function $IB(m,x)$

Let us first consider the function

$$f_1(m,x) = \frac{x}{(1+x)^m}, \qquad m \geq 0, \tag{A.1}$$

where m is a non-negative valued parameter. It can be easily established that this function

1. has no turning points if $0 \leq m \leq 1$.
2. has a maximum value of $(m-1)^{m-1}/m^m$ at $x = 1/(m-1)$ if $m > 1$.
3. if m is not an integer $x < -1$ yields complex values.
4. The following results hold with respect to limits.

$$\begin{aligned}
\lim_{x\to\infty} f_1(m,x) &= 0 \\
\lim_{x\to-\infty} f_1(m,x) &= 0 \\
\lim_{x\to-1^R} f_1(m,x) &= -\infty \\
\lim_{x\to-1^L} f_1(m,x) &= \begin{cases} -\infty & \text{if } m \text{ is even} \\ +\infty & \text{if } m \text{ is odd} \end{cases}
\end{aligned}$$

From these properties it can be easily seen that the inverse function of $f_1(m,x)$ has at most three branches.

Definition 1 *The real valued function $IB(m,x)$ is the inverse function of $f_1(m,x)$ or defined implicitly*

$$\frac{IB(m,x)}{(1+IB(m,x))^m} = x. \tag{A.2}$$

If the inverse function of $f_1(m,x)$ has more than one branch, the branch with the numerically smallest ordinate is taken as $IB(m,x)$.

From the properties of the function $f_1(m,x)$ we can conclude the following.

$$IB(0,x) = x, \tag{A.3}$$

$$IB(1,x) = \frac{x}{1-x}, \tag{A.4}$$

$$IB(2,x) = \frac{1-2x-\sqrt{1-4x}}{2x}, \quad x \leq \tfrac{1}{4} \tag{A.5}$$

$$IB(m,0) = 0, \tag{A.6}$$

$$IB\left(m, \frac{(m-1)^{m-1}}{m^m}\right) = \frac{1}{m-1}, \qquad \text{for } m > 1, \tag{A.7}$$

$$IB\left(m, \frac{pq^{m-1}}{(p+q)^m}\right) = \frac{p}{q}, \quad -1 < \frac{p}{q} \leq \frac{1}{m-1} \tag{A.8}$$

$$\frac{d}{dx} IB(m,x) = \frac{IB(m,x)(1+IB(m,x))}{x\,(1-(m-1)IB(m,x))}, \tag{A.9}$$

and it has a derivative singularity at $x = (m-1)^{m-1}/m^m$, $m > 1$. These properties have been coded in Maple and codes are given in Appendix C. In the actual implementation we considered the cases where both m and x are of type numeric in the Maple sense and all other cases separately.

A.1 Arguments of numeric type

Since Maple type/numeric includes integer, fraction and float we consider integer and fraction (rational) and float separately. If the second arguments is a floating type to obtain a floating point answer it calls the procedure **evalf/IB**. The procedure **evalf/IB** makes use of the library function **fsolve** to solve $y = (1+y)^m x$. Since the function $y/(1+y)^m = x$ has only one root in the range $-1 \leq y \leq 1/(m-1)$ procedure specifies this range when it calls **fsolve**. We found the accuracy is not very high when x takes values close to $(m-1)^{m-1}/m^m$, due to the derivative singularity.

A.1.1 First Arguments is integer and second argument is rational

In this case the procedure will attempt to give a rational number as the output. For this it first checks whether the arguments are in the acceptable range and then it checks whether the rational x can be expressed in the form $\frac{pq^{m-1}}{(p+q)^m}$ where p and q are relatively prime. The procedure factorizes the numerator of x using **ifactors** library routine, the result is in the form

$$[u, [[p_1, e_1], [p_2, e_2], \ldots, [p_i, e_i], \ldots]]$$

see pp. 294 of Char *et al.* (1990), where, $u = \pm 1$, $p_i, i = 1, 2, \ldots$ are primes and $e_i, i = 1, 2 \ldots$ are integers i.e.

$$numer(x) = u \prod_{i=1} p_i^{e_i}.$$

Then the procedure inspects each factor $[p_i, e_i], i = 1, 2, \ldots$ and if e_i is an integer multiplier of $(m-1)$ then the term $p_i^{e_i/(m-1)}$ becomes a factor of q otherwise $p_i^{e_i}$ becomes a factor of p. Finally, if the denominator of x and $(q + u * p)^m$ are the same the procedure will give $u * p/q$ as the output, otherwise it outputs the unevaluated form of $IB(m, x)$.

A.2 Arguments not of numeric type

Application of Maple procedure **simplify** to the second argument of

$$IB\left(m, \frac{pq^{m-1}}{(p+q)^m}\right) \tag{A.10}$$

yields:

$$\frac{p\left(\frac{q+p}{p}\right)^{(-m)}}{q}.$$

Therefore to check whether the second argument of $IB(m, x)$ can be expressed in the form given in (A.10) the procedure works as follows: If the second argument x contains only one operand then the procedure will return the unevaluated form of it. If the first operand of x is -1, $sgn = -1$, it will start examining each operand of x from the second operand; otherwise, $sgn = 1$, it will start examination from the first operand. If the operand, say p_2, has only one operand then p_2 becomes a factor of p; otherwise if it has the form p_2^b or $p_2^{(-b)}$, where b is not an integer multiplier of m, the operand becomes a factor of p or q, respectively. Finally it will check whether x is equivalent to the form $sgn * p/q/(1 + sgn * p/q)^m$ and then output will be $sgn * p/q$; otherwise the output will be unevaluated form $IB(m, x)$.

B The inverse function $IG(m, x)$

Consider the function

$$f_2(m, x) = x(1-x)^m, \quad m > 0. \tag{B.1}$$

This function has two turning points one at $x = 1/(m+1)$ and one at $x = 1$; if m is not an integer $x > 1$ yields complex values. Therefore the inverse function of $f_2(m, x)$ has at most three branches.

Definition 2 *The real valued function $IG(m, x)$ is the inverse function of $f_2(m, x)$; in cases where more than one branch exists the branch with the numerically smallest ordinate is taken.*

From the properties of $f_2(m, x)$ we can obtain the following:

$$IG(m, 0) = 0, \tag{B.2}$$

$$IG(0, x) = x, \tag{B.3}$$

$$IG(1, x) = \frac{1-\sqrt{1-4x}}{2}, \quad x \leq \frac{1}{4}, \tag{B.4}$$

$$IG\left(m, \frac{m^m}{(1+m)^{m+1}}\right) = \frac{1}{m+1}, \tag{B.5}$$

$$IG\left(m, \frac{P(Q-P)^m}{Q^{m+1}}\right) = \frac{P}{Q}, \quad \frac{P}{Q} \leq \frac{1}{m+1} \tag{B.6}$$

$$\frac{d}{dx} IG(m, x) = \frac{IG(m, x)(1 - IG(m, x))}{x(1-(m+1)IG(m, x))}, \tag{B.7}$$

and it has a derivative singularity at $x = m^m/(1+m)^{m+1}$.

B.1 Implementation of $IG(m, x)$

The implementation details are somewhat similar to that of $IB(m, x)$ and for brevity we have excluded them.

C Maple programs

We present three separate Maple programs that can be used to carry out the evaluations discussed in the paper. The first program can be used to compute the moments and cumulants of Lagrangian distributions and compound Lagrangian distributions in general. The next two procedures can be used to handle two important classes of Lagrangian distributions.

C.1 Computing moments and cumulants

```
# The following procedures enables deriving moments and cumulants
# of Lagrangian distributions of the first and second kind.

#########################################################################
# The procedure h() uses the fact that h(z) is a pgf of a positive      #
# valued random variable. If the argument is not 0 or 1 it returns      #
# the unevaluated form.                                                 #
#########################################################################
```

```
h := proc()
local x;
if nargs <>1 then ERROR('expecting 1 argument, got'.nargs)
else x := args[1]
fi;
x := simplify(x);
if x=0 then 0
elif x=1 then 1
else 'h'(x)
fi;
end:

#################################################################
# The following procedure defines derivatives of the function #
# h(z) as given in (2.4).                                       #
#################################################################

'diff/h' := proc(a,x)
local t, gd;
gd := subs(t=h(a),diff(g(t),t));
h(a)/a/(1 - a*gd) * diff(a,x)
end:

#########################################################################
# The Following procedure defines the probability generating function #
# of Lagrangian distributions of second kind. The procedure  assumes  #
# DG1 is a global variable as defined  below, and f(t) and g(t) are   #
# pgfs with g(0) <>0.                                                 #
#########################################################################

DG1 := 1- limit(diff(g(w),w),w=1);

P_2 := proc()
global DG1;
local t,z,gd;
if nargs<>1 then ERROR('expecting 1 argument, got'.nargs)
fi;
z := args[1];
gd := subs(t=h(z), diff(g(t),t));
simplify(DG1*f(h(z))/(1-z*gd))
end:

# The functions  mu1(n), mu2(n), cmu1(n) and cmu2(n) calculate
# moments and cumulants for given n.
#
# Function  Output
# --------  ------
# mu1(n)    nth moment of Lagrangian distributions of first kind.
# mu2(n)    nth moment of Lagrangian distributions of second kind.
```

```
# cmu1(n)   nth cumulant of Lagrangian distributions of first kind.
# cmu2(n)   nth cumulant of Lagrangian distributions of second kind.

mu1 := (n) -> limit(diff(f(h(exp(s))),s$n),s=0);
mu2 := (n) -> limit(diff(P_2(exp(v)),v$n),v=0);
cmu1 := (n) -> limit(diff(log(f(h(exp(u)))),u$n),u=0);
cmu2 := (n) -> limit(diff(log(P_2(exp(v))),v$n),v=0);

#
# To use the preceding codes one has to define f(t) and g(t) in (1.1).
# One can select a distributions given in Consul and Shenton (1972)
# by removing the first three characters of two selected lines, one
# of them being a line which begins with  #g1, $g2, #g3 for g(t) and
# the other one being a line whichs begin with  #f1, #f2, #f3, #f4, #f5
# for f(t).
#
# Possible choices of g(t) and the name of the resulting
# distributions.
# 1. Poisson Lagrangian distributions.
#g1    g := (t) -> exp(b*(t-1));
# 2. Binomial Lagrangian distributions.
#g2    g := (t) -> (1-p + p*t)^b;
# 3. Negative Binomial Lagrangian distributions.
#g3    g := (t) -> (1+P -P*t)^(-b);
#
# Possible choices of f(t) and the name of the resulting
# distributions.
# 1. Basic Lagrangian distributions.
#f1    f := (t) -> t;
# 2. Lagrangian delta distributions.
#f2    f := (t) -> t^a;
# 3. Lagrangian Poisson distributions.
#f3    f := (t) -> exp(a*(t-1));
# 4. Lagrangian binomial distributions;
#f4    f := (t) -> (1-p + p*t)^a;
# 5. Lagrangian negative binomial distributions.
#f5    f := (t) -> (1+P -P*t)^(-b);
#
# For example if one is interested in obtaining the second
# moment and the fifth cumulants of Negative binomial-Poisson
# distributions one  first have to delete the characters ''#g3''
# and ''#f3''. Then it is suffices to type mu1(3), cmu1(5) and
# mu2(3), cmu2(5) to obtain the values for Lagrangian distributions
# of first and second kind respectively.
#
# The function M_S1(t) and M_S2(t) are the moment generating
# functions of compound Lagrangian distributions of first
# kind an the compound Lagrangian distributions of second
# kind, respectively.
```

```
M_S1 := (t) -> f(h(M_X(t)));
M_S2 := (t) -> P_2(M_X(t));

# The following codes illustrate how to obtain the moments of
# compound Lagrangian distributions when the claim severity
# is exponential.
# M_X(t) is the moment generating function of exponential
# distribution with parameter beta.
# mu_S_1(n) and m_S_2(n) will contains the moments of S
# after calling these functions with integer arguments.

M_X := (t) -> beta/(beta-t);
mu_S_1 := (n) -> limit(diff(M_S1(t),t$n),t=0);
mu_S_2 := (n) -> limit(diff(M_S2(t),t$n),t=0);
```

C.2 The inverse function *IB*

```
#############################################################################
# The procedure IB is an implementation of the properties given             #
# in (A.3) -(A.4). It has been implemented following the specifications#
# given in the Appendix B.                                                  #
#############################################################################
IB := proc()
local m,x,t1,t2,t3,p1,e1,i,st,sgn1,sgn2,p,q;
 if nargs = 2 then m := args[1]; x := args[2]
 else ERROR('expecting 2 arguments, got'.nargs)
 fi;
 if  x = 0 then 0
 elif m = 0 then x
 elif m-1 = 0 and not type(x,numeric) then x/(1-x)
 elif m-2 = 0 and not type(x,numeric) then (1-2*x - (1-4*x)^(1/2))/2/x
 elif type(x,numeric) and type(m,numeric) then
  if m <1 then ERROR('First Argument has to be >= 1')
  elif  x - (1-1/m)^(m-1)/m > 0 then ERROR('Second argument out of range')
  elif m-1 = 0 then x/(1-x)
  elif m-2 = 0 then  (1-2*x - (1-4*x)^(1/2))/2/x
  elif type(x,float) then evalf('IB'(m,x))
  elif type(m,integer)  and type(x,rational)  then
              t1 := numer(x);
              t2 := denom(x);
              readlib(ifactors);
              t3 := ifactors(t1);
              p := 1; q :=1;
              for i from 1 to nops(op(2,t3)) do
               p1 := op(1,op(i,op(2,t3)));
               e1 := op(2,op(i,op(2,t3)));
               if type(e1/(m-1),integer) then q := q * (p1)^(e1/(m-1))
               else p := p * p1^e1
               fi
              od;
              if  q<>1 and (q+op(1,t3)*p)^m = t2 then op(1,t3)*p/q else 'IB'(m,x)
```

```
            fi;
  else 'IB'(m,x)
  fi
 elif nops(x) >1 then
      t3 := [simplify(op(x))];
      p := 1; q := 1;
      if op(1,t3) = -1 then sgn1 := -1; st := 2
      else sgn1 := 1; st := 1
      fi;
      for i from st to nops(x) do
        p1 := op(i,t3);
          if nops(p1) =1 then p := p*p1
          else  e1 := simplify(op(2,p1)/m);
                sgn2 := sign(op(2,p1));
                if not type(e1,integer) then
                   if sgn2 = -1 then q := q/p1
                   else p := p*p1
                   fi
                fi;
        fi
      od;
      e1 := simplify(sgn1*p/q/(1+sgn1*p/q)^m/x);
      if  e1 = 1 then sgn1*p/q  else 'IB'(m,x)
      fi
 else 'IB'(m,x)
 fi;
end:
#########################################################################
# The following procedure defines how to evaluate the derivatives  #
# of IB.                                                                #
#########################################################################
`diff/IB` := proc(m,x,y) (1+IB(m,x)) * IB(m,x)/(1 - (m-1)*IB(m,x))/x
 * diff(x,y)
 end:
#########################################################################
# The following procedure facilitate the computation of the function #
# IB(m,x) when m is type/numeric and x is type/float. It uses the    #
# Maple equation solver ''fsolve'' to obtain a solution in the       #
# interval [-1,1/(m-1)] of  y/(1+y)^m = x for y, when x and m is     #
# given.                                                             #
#########################################################################
`evalf/IB` := proc()
 local m,x,y;
 if nargs<>2 then ERROR(`expecting two arguments`)
 else m := evalf(args[1]); x := evalf(args[2]);
 fi;
 if (1-1/m)^(m-1)/m -x <= Float(1,-100000) then RETURN(1/(m-1))
 elif x <= -Float(1,100000) then RETURN(-1)
 fi;
 fsolve(y=(1+y)^m*x,y,-1..1/(m-1))
```

```
end:
```

C.3 The inverse function *IG*

```
#####################################################################
# The following procedure is an implementation of the inverse       #
# function IG(m,x) defined in Appendix B. First it tries to see     #
# whether the arguments of the function would fit into arguments    #
# given  in (B.3)-(B.6) and then it will output the  corresponding  #
# RHSs. Otherwise it will return the unevaluated  form of it.       #
#####################################################################
IG := proc()
local m,x,t1,t2,t3,p1,e1,i,p,q,r,st,sgn1,sgn2;
 if nargs = 2 then m := args[1]; x := args[2]
 else ERROR(`expecting 2 arguments, got`.nargs)
 fi;
 if  x = 0 then 0
 elif m = 0 then x
 elif m-1=0 and not type(x,numeric) then (1-(1-4*x)^(1/2))/2
 elif type(x,numeric) and type(m,numeric) then
             if m <1 then ERROR(`First Argument has to be >= 1`)
             elif  x > m^m/(1+m)^(m+1) then ERROR(`Second Argument out of range`)
             elif m-1=0 then (1-(1-4*x)^(1/2))/2
             elif type(x,float) then evalf('IG'(m,x))
             elif type(x,rational) then
               t1 := numer(x);
               t2 := denom(x);
               readlib(ifactors);
               t3 := ifactors(t1);
               p := 1; q :=1; r:=1;
               for i from 1 to nops(op(2,t3)) do
                p1 := op(1,op(i,op(2,t3)));
                e1 := op(2,op(i,op(2,t3)));
                if type(e1/m,integer) then r := r * (p1)^(e1/m)
                else p := p * p1^e1
                fi
               od;
               q := simplify(t2^(1/(m+1))) ;
               if type(q,integer) and (q-op(1,t3)*p) = r then
                  op(1,t3)*p/q else 'IG'(m,x)
               fi;
             else 'IG'(m,x)
             fi
 elif nops(x) >1 then
      t3 := [op(x)];
      p := 1; q := 1;
      if op(1,t3) = -1 then sgn1 := -1; st:=2 else sgn1 := 1; st:=1 fi;
      for i from st to nops(x) do
        p1 := op(i,t3);
        if nops(p1) =1 then p := p*p1
        else e1 := simplify(op(2,p1)/m);
```

```
            sgn2 := sign(op(2,p1));
            if not type(e1,integer)  then
              if sgn2 = -1 then q := q/p1
              else p := p*p1
              fi
            fi
        fi
      od;
      e1 := simplify(sgn1*p/q*(1-sgn1*p/q)^m/x);
      if e1=1  then sgn1*p/q
      else 'IG'(m,x)
      fi
 else 'IG'(m,x)
 fi;
end:
#######################################################################
# Following procedure defines the derivative of IG(m,x).              #
#######################################################################
'diff/IG' := proc(m,x,y) (1-IG(m,x)) * IG(m,x)/(1 - (m+1)*IG(m,x))/x
 * diff(x,y)
 end:
#######################################################################
# The following procedure facilitate the computation of the function#
# IG(m,x) when m is type/numeric and x is type/float. It uses the   #
# Maple equation solver ''fsolve'' to obtain a solution in the      #
# interval [-infinity, 1/(m+1)] of y*(1-y)^m=x for y, when x and    #
# m is given.                                                       #
#######################################################################
'evalf/IG' := proc()
 local m,x,y;
 if nargs<>2 then ERROR('expecting two arguments')
 else m := evalf(args[1]); x := evalf(args[2]);
 fi;
 RETURN(fsolve(y*(1-y)^m=x,y,-infinity..1/(m+1)))
end:
```

REFERENCES

1. Ambagaspitiya, R. S. and Balakrishnan, N. (1994). On the compound generalized Poisson distribution. To appear in *ASTIN Bulletin*.

2. Char, B. W., Geddes, K. O., Gonnet, G. H., Leong, B. L., Monagan, M. B. and Watt, S. M. (1991). *Maple V Library Reference Manual.* Springer-Verlag.

3. Consul, P. C. and Shenton (1972). Use of Lagrange expansion for generating discrete generalized probability distributions. *SIAM Journal of Applied Mathematics* **23**(2), 239-248.

4. Corless, R. M., Gonnet, G. H., Hare, D. E. G. and Jeffrey, D. J. (1993). On Lambert's *W* function. To appear in *The Advances of Computational Mathematics*.

5. Goovaerts, M. J. and Kaas, R. (1991). Evaluating compound generalized Poisson distributions recursively. *ASTIN Bulletin* **21**, 193-197.

6. Janadaran, K. G. and Rao, B. R. (1983). Lagrange distributions of the second kind and weighted distributions. *SIAM Journal of Applied Mathematics* **43(2)**, 302-313.

7. Johnson, N. L., Kotz, S. and Kemp, A. W. (1992). *Univariate discrete distributions*, Second Edition, John Wiley & Sons, Inc.

8. Kling, B. and Goovaerts, M. (1993). A note on compound generalized distributions. *Scandinavian Actuarial Journal* , 60-72.

9. Otter, R. (1949). The multiplicative process, *Annals of Mathematical Statistics* **20**, 206-224.

10. Panjer, H. H. and Willmot, G. E. (1992). *Insurance Risk Models*. Society of Actuaries.