Computer Algebra in Statistics

An Introduction to Maple

Michael J. Wichura
Department of Statistics
The University of Chicago

Email: wichura@galton.uchicago.edu

October, 1995

Preface

Course notes for Statistics 304.

Comments and corrections are welcome. Please address email correspondence to the author at `wichura@galton.uchicago.edu`.

Michael J. Wichura,
University of Chicago,
October, 1995.

Contents

1 Introduction

MAPLE is an interactive computer algebra system that can be used to analyze mathematical problems using a combination of symbolic, numerical, and graphical tools. Among other things its features include: arbitrary precision arithmetic; routines for doing differential and integral calculus, both symbolically and numerically; infinite and finite series, limits, and products; expansion and factoring of algebraic equations; linear algebra; solutions of systems of equations; pattern matching; special functions; and two- and three-dimensional graphing.

MAPLE has a simple and effective programming language. Loops, conditionals, and user-defined functions are available. Results can be saved to and retrieved from files, and can be incorporated into FORTRAN and C programs or LaTeX documents. Libraries/packages of special routines can be added to the environment to extend MAPLE's capabilities.

MAPLE will greatly enhance your ability to manipulate mathematical expressions. It can save you a lot of time and effort, by quickly and accurately performing a task that would otherwise be tedious, cumbersome, and prone to errors.

MAPLE is an on-going project of the Symbolic Computation Group at the University of Waterloo in Ontario, Canada. The system is available on a large number of machines, including most Unix workstations, the Macintosh, 386-based PCs, and the NeXT. These notes were developed using MAPLE on a Unix workstation at the University of Chicago.

1.1 References

A good way for a you to learn MAPLE in depth is to first work through the on-line tutorial and then read *First Leaves: A Tutorial Introduction to Maple V* published by Springer-Verlag. These notes are intended only as a brief introduction, to illustrate some of MAPLE's capabilities in the context of some case studies.

Additional references are the *Maple V Language Reference Guide* and the *Maple V Library Reference Manual*, also published by Springer-Verlag. Much of the information in these manuals is available through the on-line help facility.

1.2 Getting in, getting help, and getting out

To get started, enter `maple` in response to your operating system's prompt:

```
tcsh: maple
    |\^/|     Maple V Release 2 (University of Chicago)
._|\|   |/|_. Copyright (c) 1981-1993 by the University of Waterloo.
 \  MAPLE  /  All rights reserved. Maple and Maple V are registered
 <____ ____>  trademarks of Waterloo Maple Software.
      |       Type ? for help.

>
```

The "`>`" is MAPLE's prompt for you to enter your first command.

To find out about a MAPLE feature, use the extensive on-line help command `?`. Here is what part of what `?` has to say about itself:

```
> ?

FUNCTION: help - descriptions of syntax, datatypes, and functions

CALLING SEQUENCE:
   ?topic  or   ?topic,subtopic   or   ?topic[subtopic]  or
   help(topic)  or  help(topic,subtopic)  or  help(topic[subtopic])

SYNOPSIS:
         ?intro                 introduction to Maple
         ?library               Maple library functions and procedures
         ?index                 list of all help categories
         ?index,<category>      list of help files on specific topics
         ?<topic>               explanation of a specific topic
         ?<topic>,<subtopic>    explanation of a subtopic under a topic
         ?distribution          for information on how to obtain Maple
         ?copyright             for information about copyrights

- Note 1: The recommended way to invoke help is to use the question
mark.
```

In the following lectures I present many MAPLE commands with only a brief explanation as to what they do. You should make a habit of using `?` to get the full story.

Beginners are well-advised to work through the on-line tutorial:

```
> ? tutorial

HELP FOR: tutorial - on-line tutorial introduction to Maple

CALLING SEQUENCE:
   tutorial()
   tutorial(n)

PARAMETERS:
   n - an integer

SYNOPSIS:
- The function tutorial() takes the user through a beginning level, on-line
  tutorial for Maple.  This tutorial is not meant to be a replacement for the
  Maple manuals (First Leaves, etc.) or the on-line help files, but is instead
  meant to offer a quick method for getting started with Maple.

- There are 14 chapters in the on-line tutorial. tutorial(n) skips the standard
  welcoming text for the tutorial and starts the user at chapter n.

- Most chapters have question and answer sections and there are two quizzes
  included.  There is a main menu, from which the reader can begin any of the
  chapters, which can be accessed from any break in the text.
```

Important: The commands you type to MAPLE may extend over several physical lines and must be terminated by a SEMI-COLON (to display the result) or COLON (to suppress the display), not a CARRIAGE RETURN. (MAPLE won't see the line until you type a carriage return.) An important point to remember is "No (semi-)colon, no computation!" The help command `?` is one of the few exceptions to this rule.

To start afresh, without leaving MAPLE, give the command **restart**; this will erase all your current variables, functions, etc. To end your MAPLE session, you may enter either **quit**, **stop**, or **done**.

2 The Change of Variable Formula

Let X be a random variable with density f_X and let $Y = h(X)$ be obtained from X via a transformation h. If h has a continuous non-zero derivative, then Y has a density given by

$$f_Y(y) = f_X(h^{-1}(y))\Big|\frac{dh^{-1}(y)}{dy}\Big| = f_X(x)\Big|\frac{dx}{dy}\Big|, \tag{1}$$

where $x = h^{-1}(y)$. I am going to illustrate how MAPLE can be used to evaluate f_Y.

2.1 Three examples

2.1.1 The logistic distribution

Let's find the density of $Y = \log(X/(1-X))$ when X has a uniform distribution on $(0,1)$. First we need to solve the equation $y = \log(x/(1-x)) = h(x)$ for $x = h^{-1}(y)$:

```
> solve (y = log(x/(1-x)), x);
```

$$\frac{\exp(y)}{\exp(y)+1}$$

Terminating the **solve** command with a semicolon caused the result to be displayed. Next find the derivative of this with respect to y:

```
> diff (", y);
```

$$\frac{\exp(y)}{\exp(y)+1} - \frac{\exp(y)^2}{(\exp(y)+1)^2}$$

The double quote operator " returns the most recently computed expression, whatever that may be; here it's the solution $\exp(y)/(\exp(y)+1)$ to the equation. (Similarly "" returns the second most recently computed expression, and """ the third.) Now simplify the derivative:

```
> simplify(");
```

$$\frac{\exp(y)}{(\exp(y)+1)^2}$$

This is dx/dy. Since it is clearly continuous and positive, so is $h' = dy/dx$ and formula (1) applies. Since $f_X(x) = 1$, Y has density $f_Y(y) = f_X(x)|\frac{dx}{dy}| = e^y/(e^y+1)^2$, for $-\infty < y < \infty$. Y is said to be distributed according to the standard logistic distribution. We're all done with this calculation, so reinitialize MAPLE:

```
> restart;
```

2.1.2 The Pareto-Beta connection

The Pareto distribution with location parameter $k > 0$ and shape parameter $\alpha > 0$ has density

$$f(x) = \frac{\alpha k^\alpha}{x^{\alpha+1}} \quad \text{for } x \geq k. \tag{2}$$

This distribution is sometimes used to model certain long-tailed socio-economic variates, such as city size, personal income, etc. Suppose X is a random variable with density (2). What is the density of $Y = k/X$?

To find out write $y = k/x$, solve for x in terms of y, and find $\frac{dx}{dy}$:

```
> solve (y = k/x, x);
```

```
                                    k/y
```

```
> diff (", y);
```

```
                                      k
                                   - ----
                                      2
                                     y
```

Next introduce the Pareto density $f_X(x)$:

```
> f := alf * k^alf / x^(alf + 1);
```

```
                                          alf
                                   alf k
                              f := ----------
                                   (alf + 1)
                                  x
```

The "`:=`" operator assigned the expression $\alpha k^\alpha / x^{\alpha+1}$ to the variable `f`. Note that assignments are specified by `:=` in MAPLE, not by `=` as in FORTRAN or C. Now use the substitution command **subs** to replace the x in $f_X(x)$ by the solution k/y, which was computed three steps earlier:

```
> subs (x = """, f);
```

```
                                        alf
                                   alf k
                                --------------
                                     (alf + 1)
                                (k/y)
```

The command

subs(*eqn*, *expr*)

returns the result of applying the substitution specified by the equation *eqn* to the expression *expr*. Multiply $f_X(x)$ by $|\frac{dx}{dy}|$ to get $f_Y(y)$:

```
> simplify (" * abs("""));
```

```
                                    (alf - 1)
                               alf y           abs(k)
                               ----------------------
                                          k
```

Tell MAPLE that k is positive:

```
> assume (k > 0);
> ";
```

```
                                    (alf - 1)
                                alf y
```

Note that here `"` refers back to the result of the **simplify** command, since **assume** didn't compute anything. It follows that Y has density $f_Y(y) = \alpha y^{\alpha-1}$, for $0 < y \leq 1$; that is, Y has a Beta distribution with parameters α and 1.

MAPLE has a **history** command that will automatically assign labels to each output expression. This enables you to easily refer back to any previous result, but there are some disadvantages — see the help page. An alternative approach to back referencing is to selectively assign your own labels; this approach is illustrated in the next section.

2.1.3 The Chisquare distribution with one degree of freedom

Now let's find the density of $Y = h(X) := X^2$ when X is standard normal. Since the transformation h maps each of the intervals $(-\infty, 0)$ and $(0, \infty)$ monotonically onto $(0, \infty)$, the density for Y is given by

$$f_Y(y) = f_X(x_1)\left|\frac{dx_1}{dy}\right| + f_X(x_2)\left|\frac{dx_2}{dy}\right|$$

with $x_1 = -\sqrt{y}$ and $x_2 = +\sqrt{y}$, for $y > 0$. To evaluate this in MAPLE, begin by solving the equation $y = x^2$ for x:

```
> solutions := solve (y = x^2, x);
                                                1/2   1/2
                                  solutions := - y   , y
```

Note that MAPLE found two solutions, and returned them as a sequence of comma-separated expressions — a datatype MAPLE calls an expression sequence. Convert the solutions to a list, so that we can easily access its n^{th} component using the subscript operator "`[n]`":

```
> solutions := [solutions];
                                                1/2   1/2
                                solutions := [- y   , y   ]
> solutions[2];
                                                 1/2
                                                y
```

MAPLE uses the notation "`[...]`" for a list of elements; e.g., `[1,2,3]` is a list whose elements are the numbers `1`, `2`, and `3`. Introduce the density

$$f_X(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$$

of the "old" random variable X:

```
> old_f := exp(-x^2/2) / sqrt (2*Pi);
                                                      2   1/2
                                             exp(- 1/2 x ) 2
                                 old_f := 1/2 ------------------
                                                    1/2
                                               Pi
```

`Pi` is MAPLE's symbol for the number π. Get the contribution to the density of Y from the branch $x = -\sqrt{y}$ of the inverse transformation:

```
> sol := solutions[1];
                                                1/2
                                          sol := - y
> dx_dy := diff (sol, y);
                                                  1
                                       dx_dy := - ------
                                                  1/2
                                             2 y
> assume (y > 0);
> contrib1 := subs (x = sol, old_f) * abs(dx_dy);
                                                          1/2
                                              exp(- 1/2 y~) 2
                                contrib1 := 1/4 ------------------
                                                    1/2   1/2
                                               Pi    y~
```

Note that MAPLE attaches a ~ to a variable (here y) when something has been assumed about it. Now repeat the computations, using the branch $x = +\sqrt{y}$ of the inverse transformation:

```
> sol := solutions[2]:
> dx_dy := diff (sol, y):
> contrib2 := subs (x = sol, old_f) * abs(dx_dy);
                                                  1/2
                                     exp(- 1/2 y~) 2
                        contrib2 := 1/4 ------------------
                                          1/2   1/2
                                       Pi    y~
```

Here I used terminating colons instead of semicolons to suppress printing the results of the first two commands; those invisible results nonetheless could have been accessed by the " and "" operators. Finally add the two contributions to get the density of the "new" variable Y:

```
> new_f := contrib1 + contrib2;
                                                  1/2
                                     exp(- 1/2 y~) 2
                         new_f := 1/2 ------------------
                                          1/2   1/2
                                       Pi    y~
```

We have found the density of Y to be

$$f_Y(y) = \frac{1}{\sqrt{2\pi y}} \exp(-y/2), \tag{3}$$

for $y > 0$. This is the density of the so-called Chisquare distribution with one degree of freedom.

2.2 Automating the procedure: the one-dimensional case

Now I am going to automate the procedure illustrated by the preceding examples. Suppose $Y = h(X)$. Put $g = h^{-1}$ and rewrite equation (1) as

$$f_Y(y) = f_X(g(y))\,|g'(y)|. \tag{4}$$

For the time being suppose the inverse transformation g is given, i.e., it is known how to write X in terms of Y. I am going to exhibit a function `chng_vars1_` such that

$$\texttt{chng\_vars1\_}(f(x),\, y,\, x = g(y))$$

returns the density of Y when $X = g(Y)$ has density $f(x)$. (The "1" in `chng_vars1_` stands for "one-dimensional.") Before we can compute $g'(y)$ we will need to peel off $g(y)$ from the right hand side of the equation $x = g(y)$; that is easily done using the function **rhs**:

```
> rhs (x = g(y));
                                      g(y)
> `%g'(y)%` := diff (", y);
                                        d
                              %g'(y)% := ---- g(y)
                                        dy
```

MAPLE treats a string of characters enclosed in back quotes (`) as the name of a variable; this notation must be used if the name includes any non alphanumeric characters (i.e., characters other than letters, digits, and underscore). Here I used this feature to create a self-identifying label for $g'(y)$. MAPLE doesn't require the enclosing %'s; they're there to remind me that when I see `%g'(y)%` in MAPLE's output I'm looking at a name, not an expression.

Here is the definition of `chng_vars1_`:

```
>
chng_vars1_ := proc (`%f(x)%`, y, `%x=g(y)%`)
        # chng_vars1_ (some expression f(x) in x, y, x = g(y)) returns
        # the density of y when x = g(y) has density f(x)
        local `%g'(y)%`;
        `%g'(y)%` := diff (rhs(`%x=g(y)%`), y);
        simplify (subs (`%x=g(y)%`, `%f(x)%`) * abs (`%g'(y)%`))
end:
```

Like SPLUS, MAPLE treats any text after "`#`" as a comment. The procedure definition

proc ($parm_1$, ..., $parm_i$) **local** var_1, ..., var_j; $statement_1$; ...; $statement_k$ **end**

is analogous to defining a function in C or a subroutine in FORTRAN. The *parm*s are the formal parameter names. The *var*s are variables that are local to (i.e., known only in) the procedure. The *statement*s are MAPLE commands involving the *parm*s and the *var*s and perhaps some global variables as well, separated by semicolons. When the procedure is called, the actual arguments are evaluated and substituted for the *parm*s throughout the *statement*s. The *statement*s are then performed one at a time, with the last value computed being the value returned.

Let's test out `chng_vars1_`, first in general:

```
> chng_vars1_ (f(x), y, x = g(y));
                                  d
                     f(g(y)) abs(---- g(y))
                                  dy
```

and then on a specific case:

```
> f := alf * k^alf / x^(alf + 1):
> assume (k > 0);
> chng_vars1_ (f, y , x = k/y);
                                       (alf - 1)
                                  alf y
```

To see how MAPLE arrived at this result I asked it to redo the calculation while tracing the execution of `chng_vars1_`. Tracing a procedure displays the entry to and exit from it as well as the results of each *statement*:

```
> trace (chng_vars1_):
> chng_vars1_ (f, y, x = k/y);
{--> enter chng_vars1_, args = alf*k~^alf/(x^(alf+1)), y, x = k~/y
                                        k~
                                %g'(y)% := - ----
                                              2
                                             y
                                       (alf - 1)
                                  alf y
<-- exit chng_vars1_ (now at top level) = alf*y^(alf-1)}
                                       (alf - 1)
                                  alf y
> `%g'(y)%`;
                                     %g'(y)%
> untrace (chng_vars1_):
```

Note that the `-->` line shows the actual arguments passed to the procedure, and that the local variable `%g'(y)%` doesn't have a value once the procedure is finished. **untrace** turns off tracing, so that subsequent calls won't generate any diagnostic output.

The preceding discussion assumed that the transformation g from Y back to X was given. However, transformation-of-variables problems are usually stated in terms of the transformation h from X to Y. To deal with this situation I'm going to exhibit a function `chng_vars1` such that

$$\texttt{chng\_vars1}(f(x),\, x,\, y = h(x))$$

returns the density of $Y = h(X)$ when X has density $f(x)$. The basic idea is to first use **solve** to obtain $g = h^{-1}$ from h and to then call `chng_vars1_` on g. If g has several branches, `chng_vars1` calls `chng_vars1_` on each branch and accumulates the results.

```
>
chng_vars1 := proc (`%f(x)%`, x, `%y=h(x)%`)
        # preliminary version
        # chng_vars1 (some expression f(x) in x, x, y = h(x)) returns
        # the density of y = h(x) when x has density f(x)
        local solutions, y, `%f(y)%`, `%g(y)%`, `%x=g(y)%`, contrib;
        solutions := solve (`%y=h(x)%`, x);
        y := lhs (`%y=h(x)%`);
        `%f(y)%` := 0;
        for `%g(y)%` in [solutions] do
                `%x=g(y)%` := x = `%g(y)%`;
                contrib := chng_vars1_ (`%f(x)%`, y, `%x=g(y)%`);
                `%f(y)%` := `%f(y)%` + contrib
        od;
        simplify (`%f(y)%`)
end:
```

The construction

for *variable* **in** *expression* **do** *statements* **od**

performs a loop: the semicolon-separated *statements* are repeatedly executed, with *variable* running over the operands of the *expression*. (Maple's **for** statement can take a number of other forms, e.g., **for** *var* **from** *start* **by** *change* **to** *finish* **do** ... **od**.) To illustrate:

```
> solutions := [solve (y = x^2, x)];
                                              1/2   1/2
                          solutions := [- y   , y   ]
> for solution in solutions do print (solution) od;
                                        1/2
                                     - y
                                       1/2
                                      y
> for summand in a + b + c*d do print (summand) od;
                                       a
                                       b
                                      c d
```

The function **print** displays its arguments.

Let's test out `chng_vars1`, first on the Pareto example:

```
>  chng_vars1 (alf * k^alf / x^(alf + 1), x, y = k/x);
                                      (alf - 1)
                                  alf y
```

and then on the Chisquare example; in this case I asked MAPLE to trace the execution of `chng_vars1`:

```
> assume (y > 0):
> trace (chng_vars1):
> chng_vars1 (exp(-x^2/2) / sqrt (2*Pi), x, y = x^2);
{--> enter chng_vars1, args = 1/2*exp(-1/2*x^2)*2^(1/2)/Pi^(1/2), x, y~ = x^2
                                        1/2    1/2
                           solutions := - y~   , y~
                                     y := y~
                                    %f(y)% := 0
                                                1/2
                              %x=g(y)% := x = - y~
                                                       1/2
                                            exp(- 1/2 y~) 2
                           contrib := 1/4 ------------------
                                              1/2   1/2
                                             Pi    y~
                                                       1/2
                                            exp(- 1/2 y~) 2
                            %f(y)% := 1/4 ------------------
                                              1/2   1/2
                                             Pi    y~
                                                 1/2
                                 %x=g(y)% := x = y~
                                                       1/2
                                            exp(- 1/2 y~) 2
                           contrib := 1/4 ------------------
                                              1/2   1/2
                                             Pi    y~
                                                       1/2
                                            exp(- 1/2 y~) 2
                            %f(y)% := 1/2 ------------------
                                              1/2   1/2
                                             Pi    y~
                                                 1/2
                                        exp(- 1/2 y~) 2
                                    1/2 ------------------
                                              1/2   1/2
                                             Pi    y~
<-- exit chng_vars1 (now at top level) = 1/2*exp(-1/2*y~)*2^(1/2)/Pi^(1/2)/y~^(
1/2)}
                                                 1/2
                                        exp(- 1/2 y~) 2
                                    1/2 ------------------
                                              1/2   1/2
                                             Pi    y~
> untrace (chng_vars1);
```

The way `chng_vars1` handles multiple branches of g is somewhat naïve. It can happen that g has a single branch over part of its domain, two branches over another part, three branches over yet another part, and so on. Since `chng_vars1` (and perhaps **solve** itself) is not designed to handle such a complex situation, it should print out a warning message whenever multiple branches are

detected, to remind the user to think about whether the algorithm employed is appropriate for the case at hand. The following code segment shows how this may be accomplished:

```
> solutions := [solve (y = x^2, x)];
                          solutions := [- y~^(1/2), y~^(1/2)]
>
if nops(solutions) > 1 then
      print (`Warning: transformation is many to one`);
      print (map (proc (expr, var)  var = expr  end, solutions, x))
fi;
                     Warning: transformation is many to one
                                  1/2        1/2
                           [x = - y~   , x = y~   ]
```

Maple's **nops** function returns the number of operands, or components, of an expression:

```
> nops (solutions);
                                    2
> nops (a + b + c*d);
                                    3
```

The **if** statement is used for conditional execution:

if *condition*$_1$ **then** *statements*$_1$ **elif** *condition*$_2$ **then** *statements*$_2$ ... **else** *statements*$_k$ **fi**

returns the result of executing the semicolon-separated *statements*$_1$ if *condition*$_1$ is true, otherwise the semicolon-separated *statements*$_2$ if *condition*$_2$ is true, ..., and otherwise the result of the semicolon-separated *statements*$_k$. The **elif** and **else** clauses may be omitted if they are not needed. The function **map** is used to apply a procedure to each operand of an expression. The construction

map (*fun*, *expr*, *arg*$_2$, ..., *arg*$_n$)

produces a new expression of the same type as the old expression *expr*, but in which each operand *op* of *expr* is replaced by *fun*$(op, arg_2, \ldots, arg_n)$. For example

```
> map (f, [1, 2, 3], a, b);
                       [f(1, a, b), f(2, a, b), f(3, a, b)]
> map (proc (x)  x^2  end, a + b + c*d);
                                  2    2    2  2
                                 a  + b  + c  d
```

Note the use of the back quotes (`) as the string delimiters for the message "`Warning ...`"; this is a common idiom in Maple.

There is another issue that needs to be confronted — what `chng_vars1` should do when **solve** is unsuccessful in computing the inverse function g, as in the following example:

```
> solve (y = sin(x) + x, x);
>
```

When a Maple builtin function realizes that it can't complete its assigned task, it typically simply returns the command that invoked it. (**solve** is an exception to this rule.) So that's what `chng_vars1` ought to do in cases like the above. The following code segment shows how that may be accomplished:

```
>
checker := proc (var, equation)
     local solutions;
     solutions := solve (equation, var);
     if solutions = NULL then
         RETURN ('procname (args)');
     else
         RETURN (solutions);
     fi
end:
> checker (x, y = k/x);
                                          k~
                                         ----
                                          y~
> checker (x, y = sin(x) + x);
                            checker(x, y~ = sin(x) + x)
```

The symbol **NULL** stands for the empty expression sequence, which is what **solve** returns when it is unsuccessful. **RETURN**(*expr*) causes an immediate return from a procedure, with *expr* as the return value. When a procedure is executed the special names **procname** and **args** get replaced respectively by the name by which the procedure was invoked and the sequence of arguments with which it was invoked. Enclosing an expression within single quotes (') tells MAPLE that the expression stands for itself, rather than for its value. For example

```
> x := 5;
                                       x := 5
> x;
                                         5
> 'x';
                                         x
```

The single quotes around "`procname (args)`" in `checker` are needed to keep `checker` from calling itself recursively, in an infinite loop.

solve can exhibit some other anomalous behaviors, but I propose to ignore them for now. Here then are the final versions of `chng_vars1_` and `chng_vars1`. I have these functions stored in a disk file, so that they can be easily modified with a standard text editor:

```
tcsh: cat chvars1

chng_vars1_ := proc (`%f(x)%`, y, `%x=g(y)%`)
        # chng_vars1_ (some expression f(x) in x, y, x = g(y)) returns
        # the density of y when x = g(y) has density f(x)
        local `%g'(y)%`;
        `%g'(y)%` := diff (rhs(`%x=g(y)%`), y);
        simplify (subs (`%x=g(y)%`, `%f(x)%`) * abs (`%g'(y)%`))
end:

chng_vars1 := proc (`%f(x)%`, x, `%y=h(x)%`)
        # chng_vars1 (some expression f(x) in x, x, y = h(x)) returns
        # the density of y = h(x) when x has density f(x)
        local solutions, y, `%f(y)%`, `%g(y)%`, `%x=g(y)%`, contrib;
        solutions := solve (`%y=h(x)%`, x);
        if solutions = NULL then
                RETURN ('procname (args)')
        fi;
```

```
        solutions := [solutions];
        if nops (solutions) > 1 then
                print (`Warning: transformation is many to one`);
                print (map (proc(expr, var)  var = expr  end, solutions, x))
        fi;
        y := lhs (`%y=h(x)%`);
        `%f(y)%` := 0;
        for `%g(y)%` in solutions do
                `%x=g(y)%` := x = `%g(y)%`;
                contrib := chng_vars1_ (`%f(x)%`, y, `%x=g(y)%`);
                `%f(y)%` := `%f(y)%` + contrib
        od;
        simplify (`%f(y)%`)
end:
```

A file of MAPLE commands can be loaded using the **read** command; the file name must be back-quoted if contains any non alphanumeric characters, such as PERIOD or SLASH.

```
> restart;
> read (`./chvars1`);
> assume (y > 0);
> chng_vars1 (exp(-x^2/2) / sqrt (2*Pi), x, y = x^2);
                    Warning: transformation is many to one
                                    1/2          1/2
                           [x = - y~   , x = y~   ]
                                                1/2
                                    exp(- 1/2 y~) 2
                             1/2 ------------------
                                      1/2   1/2
                                    Pi    y~
> chng_vars1 (6*x*(1-x), x, y = sin(x) + x);
                  chng_vars1(6 x (1 - x), x, y~ = sin(x) + x)
```

2.3 The multi-dimensional case

Suppose $Y = (Y_1, \ldots, Y_k)$ and $X = (X_1, \ldots, X_k)$ are each k-dimensional random vectors, related by the equation $Y = h(X)$. Under appropriate regularity conditions on the transformation h, the analogue of formula (4) is

$$f_Y(y) = f_X(g(y))\,|J_g(y)| \tag{5}$$

where now $y = (y_1, \ldots, y_k)$ and J_g denotes the Jacobian of $g = h^{-1}$, i.e., the determinant of the matrix

$$\begin{pmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} & \cdots & \frac{\partial x_1}{\partial y_k} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} & \cdots & \frac{\partial x_2}{\partial y_k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_k}{\partial y_1} & \frac{\partial x_k}{\partial y_2} & \cdots & \frac{\partial x_k}{\partial y_k} \end{pmatrix}.$$

The standard regularity conditions on h are that it be a one-to-one map of an open set $\mathcal{X}$ onto an open set $\mathcal{Y}$, that the partial derivatives $\partial x_i/\partial y_j$ be continuous in y for $y \in \mathcal{Y}$, and that $J_g(y)$ be non-zero for each $y \in \mathcal{Y}$.

The new feature here is J_g. To see what's involved in calculating it in MAPLE, suppose that $k = 2$, $\mathcal{X} := \{(x_1, x_2) : x_1 > 0,\ x_2 > 0\}$ and h is the transformation

$$y_1 = x_1 + x_2$$
$$y_2 = x_1/(x_1 + x_2).$$

We first need to solve these equations for the inverse transformation g:

```
> solutions := solve ({y1 = x1 + x2, y2 = x1/(x1 + x2)}, {x1, x2});
                          {x1 = y2 y1, x2 = y1 - y2 y1}
```

This says that g is the transformation

$$x_1 = y_1 y_2$$
$$x_2 = y_1(1 - y_2)$$

and it follows that $\mathcal{Y} = \{(y_1, y_2) : y_1 > 0,\ 0 < y_2 < 1\}$. In the case of simultaneous equations the arguments to **solve** have to be expressed as a set of equations and a set of variables (this syntax can also be used for a single equation); moreover **solve** returns each of the various solutions it finds as a set of equations. In MAPLE sets are expressed by curly braces: `{a, b, c}` is the set whose elements are `a`, `b`, and `c`. Sets are different from lists in that the order of the elements is irrelevant and there are no duplicate elements.

The jacobian $J_g = \det\Big(\big(\frac{\partial x_i}{\partial y_j}\big)_{1\le i\le 2,\, 1\le j\le 2}\Big)$ may now be computed as follows. Take the right hand sides $y_2 y_1$ and $y_1 - y_2 y_1$ of the equations for x_1 and x_2 and arrange them in a list:

```
> map (rhs, solutions);
                                {y2 y1, y1 - y2 y1}
> xs := convert (", list);
                           xs := [y2 y1, y1 - y2 y1]
```

MAPLE's **convert** procedure is used to convert from one data type to another — here from a set to a list. Use the **with** command to load the jacobian and determinant routines from MAPLE's linear algebra package:

```
> with (linalg, jacobian, det);
                                 [det, jacobian]
```

Compute the matrix of partial derivatives of x_1 and x_2 with respect to y_1 and y_2:

```
> ys := [y1, y2];
                                  ys := [y1, y2]
> jacobian (xs, ys);
                                 [   y2     y1  ]
                                 [              ]
                                 [ 1 - y2  - y1 ]
```

Finally, get J_g by computing the determinant $y_2(-y_1) - y_1(1 - y_2) = -y_1$ of this matrix:

```
> det (");
                                      - y1
```

The arguments to MAPLE's **jacobian** procedure have to be lists (or vectors), not sets. Contrary to what its name would suggest, the procedure returns the matrix of partial derivatives rather than

its determinant. Note that in our example the partial derivatives $\partial x_i/\partial y_j$ are in fact continuous functions of $y = (y_1, y_2)$ and $J_g(y) \neq 0$ for all $y \in \mathcal{Y}$.

We are nearly ready to define multi-dimensional versions of `chng_vars1_` and `chng_vars1`. It would be wise for these functions to run some sanity checks on their arguments. To illustrate how this may be done, here is a procedure which expects its one and only one argument to be either an equation or a set of equations.

```
>
arg_checker := proc (a)
        if nargs <> 1 then
                ERROR (`expecting 1 argument but got`, nargs)
        fi;
        if not ( type (a, equation) or type (a, set(equation)) ) then
                ERROR (`argument`, a, `not an equation or a set of equations`)
        fi;
        `OK`
end:
> arg_checker (y = k/x);
                                        OK
> arg_checker ({y1=x1+x2, y2=x1/(x1+x2)});
                                        OK
> arg_checker (x1, y1);
Error, (in arg_checker) expecting 1 argument but got, 2
> arg_checker (x1 + y1);
Error, (in arg_checker) argument, x1+y1, not an equation or a set of equations
```

Within a procedure the special name **nargs** has as its value the number of arguments with which the procedure was called. "<>" is MAPLE's notation for "not equal". The **ERROR** function causes an immediate return to top-level MAPLE. The boolean operators **not** and **or** have their usual meanings in logic. The command

type (*expr*, *type_name*)

returns **true** if the expression *expr* is of type *type_name*, and **false** otherwise. *type_name* may be either a single type name, or a set of type names:

```
> type (x = g(y), equation);
                                      true
> type (x = g(y), list);
                                     false
> type ({y1=x1+x2, y2=x1/(x1+x2)}, {equation, list});
                                      false
> type ({y1=x1+x2, y2=x1/(x1+x2)}, {equation, set(equation)});
                                      true
```

There is actually a simpler way to do type checking on the arguments to a procedure. If you define a procedure with

proc (..., arg_i **:** $type\_name_i$, ...) ... **end**

then each time the procedure is invoked MAPLE will abort the procedure call with an appropriate error message if arg_i isn't of type $type\_name_i$. Here then is a better way to write `arg_checker`:

```
>
arg_checker := proc (a : {equation, set(equation)} )
        if nargs <> 1 then
                ERROR (‘expecting 1 argument but got‘, nargs)
        fi;
        ‘OK‘
end:
>  arg_checker (y = k/x);
                                        OK
> arg_checker (x1 + y1);
Error, arg_checker expects its 1st argument, a, to be of type {equation,
set(equation)}, but received x1+y1
```

Now we can define multi-dimensional versions of `chng_vars1_` and `chng_vars1`:

`chng_vars_` (*old_density*, $\{y_1, \ldots, y_k\}$, $\{x_1 = g_1(y_1, \ldots, y_k), \ldots, x_k = g_k(y_1, \ldots, y_k)\}$)
`chng_vars` (*old_density*, $\{x_1, \ldots, x_k\}$, $\{y_1 = h_1(x_1, \ldots, x_k), \ldots, y_k = h_k(x_1, \ldots, x_k)\}$)

Both functions should return the density of $y = (y_1, \ldots, y_k)$ when $x = (x_1, \ldots, x_k)$ has density given by *old_density*, which is to be expressed in terms of x. `chng_vars_` is to be used when x is written as $g(y)$, while `chng_vars` is to be used when y is written as $h(x)$. When $k = 1$ it is convenient to allow the simpler calling sequences `chng_vars_` (*old_density*, y, $x = g(y)$) and `chng_vars` (*old_density*, x, $y = h(x)$).

```
enset := proc (s)
        # return s if s is a set, otherwise return the set consisting of s
        if type (s, set) then s else {s} fi
end:
chng_vars_ := proc(‘%f(x)%‘, y:{name, set(name)}, ‘%x=g(y)%‘:{equation, set(equation)})
        local ‘%{y}%‘, ‘%{x=g(y)}%‘, exprs, vars, J;
        # check arguments, and express them as sets if need be
        if nargs <> 3 then
                ERROR (‘expected 3 arguments but got‘, nargs);
        fi;
        ‘%{y}%‘ := enset (y);
        ‘%{x=g(y)}%‘ := enset (‘%x=g(y)%‘);
        if nops(‘%{y}%‘) <> nops(‘%{x=g(y)}%‘) then
                ERROR (‘arguments‘, y, ‘and‘, ‘%x=g(y)%‘, ‘are of unequal length‘);
        fi;
        # compute J, the determinant of the matrix of partial derivatives
        exprs := convert (map (rhs, ‘%{x=g(y)}%‘), list);
        vars  := convert (‘%{y}%‘, list);
        J := simplify (linalg[det] ( linalg[jacobian] (exprs, vars) ) );
        if J = 0 then
                ERROR (‘singular transformation‘)
        fi;
        # express the old density in terms of the new variables and multiply by
        # the absolute value of J
        simplify ( subs (‘%{x=g(y)}%‘, ‘%f(x)%‘) * abs(J))
end:
```

Note that MAPLE's relational operator for equality is "`=`", not "`==`" as in C or SPLUS. `chng_vars_` refers to the **det** and **jacobian** procedures from the **linalg** package by their "long" names — **linalg[det]** and **linalg[jacobian]** — so that these procedures don't have to be preloaded by **with**.

```
chng_vars := proc (`%f(x)%`, x:{name, set(name)}, `%y=h(x)%`:{equation, set(equation)})
        local `%{x}%`, `%{y=h(x)}%`, `%{y}%`, solutions, `%{x=g(y)}%`,
                `%f(y)%`, contrib;
        # check arguments
        if nargs <> 3 then
                ERROR (`expected 3 arguments but got`, nargs);
        fi;
        `%{x}%` := enset (x);
        `%{y=h(x)}%` := enset (`%y=h(x)%`);
        `%{y}%` := map (lhs, `%{y=h(x)}%`);
        if nops(`%{x}%`) <> nops(`%{y=h(x)}%`) then
                ERROR (`arguments`, x, `and`, `%y=h(x)%`, `are of unequal length`);
        fi;
        # solve the equations for the old variables in terms of the new ones
        solutions := solve (`%{y=h(x)}%`, `%{x}%`);
        if solutions = NULL then
                RETURN ('procname (args)')
        fi;
        if nops([solutions]) > 1 then
                print (`Warning: transformation is many to one`);
                print (solutions);
        fi;
        # accumulate the contributions to the new density from the various
        # branches of the solution (each branch is a set of equations)
        `%f(y)%`:= 0;
        for `%{x=g(y)}%` in [solutions] do
                contrib := chng_vars_ (`%f(x)%`, `%{y}%`, `%{x=g(y)}%`);
               `%f(y)%`:= `%f(y)%` + contrib
        od;
        simplify (`%f(y)%`);
end:
```

The functions `chng_vars1_` and `chng_vars1` of the preceding section are no longer needed, since `chng_vars_` and `chng_vars` do everything they did. For example:

```
> chng_vars (exp (-x^2/2)/sqrt(2*Pi), x, y=x^2);
                    Warning: transformation is many to one
                                   1/2          1/2
                          {x = - y~   }, {x = y~   }
                                                1/2
                                     exp(- 1/2 y~) 2
                               1/2 ------------------
                                         1/2   1/2
                                        Pi    y~
```

The same disk file that holds the source for these procedures also has the corresponding help information, specified using the construction

`help/text/`*function*` := TEXT(** `*text_str*$_1$`, `*text_str*$_2$` , ..., `*text_str*$_n$`):

When the user types "`? `*function*", MAPLE prints each of the text strings *text_str*s in the **TEXT** data structure `help/text/`*function* on a separate line, stripped of the leading and trailing back quotes. For example, here is part of the help text for `chng_vars`:

```
‘help/text/chng_vars‘ := TEXT (
‘FUNCTION: chng_vars - find the density resulting from a change of variables‘,
‘  ‘,
‘CALLING SEQUENCE:‘,
‘   chng_vars (‘‘%f(x)%‘‘, x, ‘‘%y=h(x)%‘‘)‘,
‘  ‘,
‘PARAMETERS:‘,
‘   ‘‘%f(x)%‘‘    - an expression‘,
‘   x           - a name or set of names‘,
‘   ‘‘%y=h(x)%‘‘  - an equation or set of equations‘,

...

‘SEE ALSO:  chng_vars_‘
):
```

Notice that to get a back quote into a text string you have enter it as two consecutive back quotes. The help page for a MAPLE function typically has the following categories:

- HELP FOR (or FUNCTION) — the procedure under discussion;
- CALLING SEQUENCE — how to invoke the procedure;
- PARAMETERS — what each parameter must or can be;
- SYNOPSIS — a brief description what the function does;
- EXAMPLES — examples of the use of the procedure;
- SEE ALSO — related items for which help is available.

Here is how the help page for `chng_vars` looks when MAPLE prints it out:

```
> ? chng_vars
FUNCTION: chng_vars - find the density resulting from a change of variables

CALLING SEQUENCE:
   chng_vars (‘%f(x)%‘, x, ‘%y=h(x)%‘)

PARAMETERS:
   ‘%f(x)%‘    - an expression
   x           - a name or set of names
   ‘%y=h(x)%‘  - an equation or set of equations

SYNOPSIS:
- chng_vars computes the density of a random variable/vector y which is the
  transformation of some random variable/vector x having a known density.  The
  parameter ‘%y=h(x)%‘ specifies the transformation - the way y depends on x -
  as an equation of the form y=h(x), or as a set of such equations, one for
  each of the components y1,...yk of y, in terms of the components x1,...,xk
  of x.  The density ‘%f(x)%‘ of x must be expressed in terms of the components
  of x.

EXAMPLE:
> assume (k > 0):
> chng_vars (a * k^a / x^(a+1), x, y = k/x);

                                        (a - 1)
                                   a y
```

```
> assume (y1 > 0):
> dgamma := (x, r, la) -> la^r * x^(r-1) * exp(-la*x) / GAMMA(r):
> chng_vars (dgamma(x1,r1,la)*dgamma(x2,r2,la), {x1,x2}, {y1=x1+x2, y2=x1/(x1+x2)}):
> simplify (factor ("));
       (r1 + r2)   (r1 - 1)     (r1 + r2 - 1)          (r2 - 1)
     la          y2         y1~               (1 - y2)          exp(- la y1~)
     ------------------------------------------------------------------------
                               GAMMA(r1) GAMMA(r2)
SEE ALSO:  chng_vars_
```

The last example is discussed in the next subsection.

2.4 Example: the Gamma-Beta-Chisquare-F-T connection

I am going to use the tools we have developed to study some important distributions in statistics. The gamma distribution $G(r, \lambda)$ with shape parameter $r > 0$ and scale parameter $\lambda > 0$ has density

$$g_{r,\lambda}(x) = \lambda^r x^{r-1} e^{-\lambda x} / \Gamma(r) \tag{6}$$

for $x > 0$. Suppose X_1 and X_2 are independent random variables with $X_1 \sim G(r_1, \lambda)$ and $X_2 \sim G(r_2, \lambda)$. Put

$$Y_1 = X_1 + X_2,$$
$$Y_2 = X_1/(X_1 + X_2).$$

Let's use `chng_vars` to find the joint density of Y_1 and Y_2. To begin, do:

```
> restart;
> read (‘chvars‘);
> dgamma := (x, r, la) -> la^r * x^(r-1) * exp(-la*x) / GAMMA(r);
                                          r  (r - 1)
                                       la  x        exp(- la x)
                 dgamma := (x,r,la) -> -----------------------
                                             GAMMA(r)
```

GAMMA is MAPLE's notation for the Gamma function. The construction

vars `->` *result*

is short for "**proc(***vars***)** *result* **end**". Here *vars* is a single variable name or a sequence of variable names enclosed in parentheses, and *result* is a single statement specifying the result of the procedure acting on *vars*. `dgamma` is the name of my user-defined function for the gamma density. Continue with

```
> assume (y1 > 0);
> chng_vars (dgamma(x1,r1,la)*dgamma(x2,r2,la), {x1,x2}, {y1=x1+x2,y2=x1/(x1+x2)});
        (r1 + r2)   (r1 - 1)    r1              (r2 - 1)
      la          y2         y1~   (y1~ - y2 y1~)          exp(- la y1~)
     ------------------------------------------------------------------
                               GAMMA(r1) GAMMA(r2)
> simplify(factor("));
       (r1 + r2)   (r1 - 1)     (r1 + r2 - 1)          (r2 - 1)
     la          y2         y1~               (1 - y2)          exp(- la y1~)
     ------------------------------------------------------------------------
                               GAMMA(r1) GAMMA(r2)
```

This is the joint density of Y_1 and Y_2, the formula being valid for all $y_1 > 0$ and $0 < y_2 < 1$. Dividing by $g_{r_1+r_2,\lambda}(y_1)$ produces

```
> "/dgamma(y1, r1+r2, la);
                          (r1 - 1)          (r2 - 1)
                        y2         (1 - y2)          GAMMA(r1 + r2)
                        ------------------------------------------
                                  GAMMA(r1) GAMMA(r2)
```

It follows that

$$Y_1 \text{ and } Y_2 \text{ are independent,} \tag{7a}$$

$$Y_1 \sim G(r_1 + r_2, \lambda), \text{ and} \tag{7b}$$

$$Y_2 \text{ has the Beta distribution } B(r_1, r_2), \tag{7c}$$

with density

$$b_{r_1,r_2}(y) = y^{r_1-1}(1-y)^{r_2-1}/\mathrm{B}(r_1, r_2) \tag{8}$$

for $0 < y < 1$; here $\mathrm{B}(r_1, r_2) = \Gamma(r_1)\Gamma(r_2)/\Gamma(r_1 + r_2)$ is the usual Beta function.

The Chisquare distribution $\chi^2(n)$ with n degrees of freedom is the distribution of

$$SS = Z_1^2 + Z_2^2 + \cdots + Z_n^2,$$

the Z_i's being independent standard normal random variables. It follows formula (3) that $Z_i^2 \sim G(1/2, 1/2)$ and from (7b) that $SS \sim G(n/2, 1/2)$, with density

$$\chi^2_n(x) = g_{n/2,1/2}(x) = \frac{1}{\Gamma(n/2)2^{n/2}} x^{n/2-1} e^{-x/2}. \tag{9}$$

We could use `->` to define a MAPLE function to compute this, but it is easier to say

```
> dchisq := unapply (simplify (dgamma(x, n/2, 1/2)), x, n);
                                  (- 1/2 n)  (1/2 n - 1)
                                 2          x            exp(- 1/2 x)
                dchisq := (x,n) -> ------------------------------------
                                              GAMMA(1/2 n)
```

In general the result of

unapply (*expr***,** *vars***)**

is a function having the variables *vars* as its parameters and the expression *expr* as its return value.

The unnormalized F-distribution $UF(n_1, n_2)$ with n_1 and n_2 degrees of freedom is the distribution of

$$F = SS_1/SS_2$$

where SS_1 and SS_2 are independent Chisquare variables with n_1 and n_2 degrees of freedom respectively. Since

$$B := \frac{F}{1+F} = \frac{SS_1}{SS_1 + SS_2},$$

(7c) implies that $B \sim B(n_1/2, n_2/2)$. We can get the density of F from the density of B with `chng_vars_`:

```
> dbeta := (y, r1, r2) -> y^(r1-1) * (1-y)^(r2-1) / Beta(r1, r2);
                                          (r1 - 1)        (r2 - 1)
                                         y         (1 - y)
                dbeta := (y,r1,r2) -> ---------------------------
                                                 Beta(r1, r2)
> duf := unapply (chng_vars_ (dbeta (b, n1/2, n2/2), f, b = f/(1+f)), f, n1, n2);
                                       (1/2 n1 - 1)        (- 1/2 n1 - 1/2 n2)
                                      f             (1 + f)
          duf := (f,n1,n2) -> ---------------------------------------------
                                           Beta(1/2 n1, 1/2 n2)
```

Beta is MAPLE's notation for the Beta function. My function `dbeta` computes the beta density (8), and `duf` computes the density

$$\phi_{n_1,n_2}(f) = \frac{1}{\mathrm{B}(n_1/2, n_2/2)} \frac{f^{n_1/2-1}}{(1+f)^{(n_1+n_2)/2}} \tag{10}$$

of F; this formula is valid for $f > 0$.

The unnormalized t distribution $UT(n)$ with n degrees of freedom is the distribution of

$$T = Z/\sqrt{SS}$$

where Z and SS are independent, with $Z \sim N(0,1)$ and $SS \sim \chi^2(n)$. Evidently

$$F := T^2 = Z^2/SS \sim UF(1,n).$$

`chng_vars` gives the density $f_{|T|}(t)$ of

$$|T| = \sqrt{F}$$

over $(0, \infty)$:

```
> assume (abs_t > 0):
> chng_vars (duf(f, 1, n), f, abs_t = sqrt(f));
                                          2 (- 1/2 - 1/2 n)
                                 (1 + abs_t~ )
                              2 -----------------------------
                                       Beta(1/2, 1/2 n)
```

By symmetry, the density of T itself is $f_T(t) = f_{|T|}(|t|)/2$, for $-\infty < t < \infty$:

```
> subs (abs_t^2 = t^2, "/2);
                                        2 (- 1/2 - 1/2 n)
                                (1 + t )
                                ------------------------
                                   Beta(1/2, 1/2 n)
```

This says that T has density

$$\tau_n(t) = \frac{1}{\mathrm{B}(1/2, n/2)} \frac{1}{(1+t^2)^{(n+1)/2}}, \tag{11}$$

for $-\infty < t < \infty$. Make this into a MAPLE function with

```
> dut := unapply (", t, n):
```

For $n = 1$ we get the density

$$\tau_1(t) = \frac{1}{\mathrm{B}(1/2, 1/2)} \frac{1}{(1+t)^2} = \frac{1}{\pi} \frac{1}{(1+t)^2} \tag{12}$$

of the standard Cauchy distribution:

```
> dcauchy := unapply (dut(x, 1), x);
```

$$\texttt{dcauchy} := x \rightarrow \frac{1}{(1 + x^2)\,\pi}$$

To finish off this MAPLE session, do

```
> save dgamma, dchisq, dbeta, duf, dut, dcauchy, ‘densities‘;
> save dgamma, dchisq, dbeta, duf, dut, dcauchy, ‘densities.m‘;
```

to save the definitions of `dgamma`, ..., `dcauchy` into the text file `densities` and the MAPLE internal format file `densities.m`. By convention, files in internal format have names ending in `.m`; they can be loaded into MAPLE at high speed by **read** and **with**, but they are not human-readable.

Index