Abstract. A collection of MAPLE V Release 2 routines are present that aid in the computation of classical Lie point symmetries and some non-classical symmetries of systems of differential equations using the jet bundle formalism of [1]. The routines exploit the user friendly output of Release 2.
The Use of MAPLE in the Search for Symmetries

Mark Hickman
Department of Mathematics
University of Canterbury
Christchurch, New Zealand
e-mail: msh@math.canterbury.ac.nz

Abstract

A collection of MAPLE V Release 2 routines are presented that aid in the computation of classical Lie point symmetries and some non-classical symmetries of systems of differential equations using the jet bundle formalism of [1]. The routines exploit the user friendly output of Release 2.

1 Introduction

Algebraic computing packages have, almost from their very beginnings, been used in the search for (Lie) symmetries of differential equations. Many packages have now been written for MAPLE and other algebraic computing systems. In particular MAPLE has the liesym package as part of its standard library. These various packages have attempted to automate the computation of symmetries to varying degrees of success. There is an algorithm that will produce a vastly overdetermined system of defining equations (which are linear first order partial differential equations) for Lie point symmetries. Most of the standard packages (including liesym) will produce this system. A few packages, for example, [2, 3] will reduce this system to a “canonical form” by adding in the integrability conditions and “eliminating” redundant equations. Most packages allow some (in many cases, rudimentary) methods of manually simplifying the system.
In this paper, a few "tools", written in MAPLE, are presented which aid the symmetry analysis of differential equations; not only in the search for classical Lie point symmetries but also for non-classical concepts of symmetries such as non-local symmetries and Wahlquist-Estabrook type prolongations [4, 5, 6]. These sorts of analysis often lead to Backlund transformations and related concepts.

2 Design Criteria

Existing packages, such as liesymm, place much emphasis on the algorithm in finding the defining equations. This set of equations is vastly overdetermined and, once obtained, provides little insight into how they might be solved. The major design criterion was to have a methodology that would closely parallel the way symmetries would be found manually; that is, one does not polynomially decompose the defining equations to obtain all components initially but uses information gained from earlier obtained components to simplify (the yet to be determined) components. As such, the MAPLE procedures presented here do not constitute a "package" in the sense of liesymm that will automatically produce the overdetermined system obtained from polynomial decomposition but rather a collection of tools to help in the analysis. An added benefit is that the tools can be easily used to find non-classical type symmetries such as partial symmetries [6] which occur in prolongation problems.

Another major design requirement was an user friendly interface. With the advent of Release 2 of MAPLE V, it is now possible to get output in a much more easily digested form. The MAPLE routines allow the user to choose the labels for the various components for the symmetries rather than have them assigned "meaningless" labels. Thus, for example, the symmetry could be

\[
v = \zeta \frac{\partial}{\partial x} + \tau \frac{\partial}{\partial t} + \varphi \frac{\partial}{\partial u}
\]

rather than

\[
v = V_1 \frac{\partial}{\partial x_1} + V_2 \frac{\partial}{\partial x_2} + V_3 \frac{\partial}{\partial x_3}.
\]

Again this comes back to requirement that the process should closely mirror the traditional method. In other words, MAPLE should "think" like the user rather than the user being required to "think" like MAPLE!
The `liesymm` package uses the exterior calculus to formulate the symmetries of a system of differential equations. The procedures here use jet bundle methods (see for example [1]) to formulate symmetries. Whilst for evolution type equations (that is, equations that are first order on one of their variables) there is little ambiguity in the choice of a differential ideal to represent the equations, there is ambiguity for more general types of equations (for example, the sine-Gordon equation \( u_{xt} = \sin u \) [7]). This ambiguity can lead to vastly different prolongation algebras of various ideals that nominally represent the same equation\(^1\). The desire to compute non-classical symmetries as well as the classical Lie point symmetries led to the choice of the jet bundle approach.

### 3 Implementation

The first decision in the implementation was the choice of data structure to be used for the coefficients of the symmetry generator. For Lie point symmetries these are functions of the independent and dependent variables. The natural choice would be to use the MAPLE function data type (that is \( A(x) \), for example). However MAPLE sees the function argument as a parameter and thus does not regard \( A(x) \) as a global variable. This results in single level evaluation rather than the usual full evaluation. In this application this has very undesirable consequences as illustrated in the following MAPLE session.

\(^1\)The various ideals can be though of as representing the equation plus some of its integrability conditions. In the case of the sine-Gordon equation, ideals that represent \( u_{xt} = \sin u \) and the system

\[
\begin{align*}
  u_{xt} &= \sin u \\
  u_{tx} &= \sin u
\end{align*}
\]

give different prolongation structures. Traditionally, one would want to identify both these systems as the same equation.
The fact that even the explicit use of `eval` will not force full evaluation would create problems for the computation of symmetries as this process involves a (normally) long chain of substitutions in order to compute the coefficients of the generator. Of course the non-inert differentiation, `diff`, requires arguments in order to determine functional dependencies and so some type of data structure is needed that will allow the functional dependencies to be encoded. The problem with evaluation, fortunately, does not occur with indexed quantities of the type \( A(x) \). Furthermore, `diff` does recognize the indices as functional dependencies (see Appendix C for a detailed discussion). Thus the coefficients
of the symmetry generators are represented by indexed quantities.

MAPLE requires that the functional dependencies be given explicitly in each call to diff. In practice this is tedious and one would normally want to suppress the arguments (when working with “pencil and paper” one normally doesn’t include the functional dependencies). This can be implemented via the use of the alias command. The procedure domain gives a convenient way to attach functional dependencies to labels. The dependencies are stored in a global variable _FUNCNAMES so that the dependencies can be restored via explicit. If a label has already been given a functional dependency then domain will warn the user of this fact and ask if the label is to be redefined. When redefining the functional dependencies, the new variables should be a subset of the original set of variables. This is to avoid the following situation. Suppose \( f \) is declared to depend only on \( x \) and \( a := \text{diff}(f, t) \) then, even if \( f \) is redefined to depend on \( t \), \( a \) would still evaluate to 0. Note that if a label has already been assigned a value then it can not be given declared though domain. The procedure var will give the dependencies of its argument. If it is call with no arguments then it gives all the current function declarations.

```plaintext
_FUNCNAMES:=NULL;

domain:=proc(vlist:list) local i,pos,tmp,tmp1,vars;
    options 'Copyright 1993 by Mark Hickman';
    vars:=op(vlist);
    for i from 2 to nargs do
        tmp:=args[i];
        if tmp = explicit(tmp) then
            _FUNCNAMES:=_FUNCNAMES,tmp=tmp[vars];
            alias(tmp=tmp[vars])
        else
            tmpl:=op(0,tmp);
            convert(tmpl,string);
            print(cat(",' has already been declared"));
            print(cat('Do you wish to redefine ",
                ','? y or n'));
            if readline() = 'y' then
                if not {vars} union {op(tmp)} = {op(tmp)} then
                    print('WARNING: The new variables');
                    print(vars);
                    print('are not a subset of the old variables');
                    print(op(tmp))
```
In the procedure domain, the local variable tmp represents the aliased quantity whereas tmpl represents the label only. In the output both tmp and tmpl would appear the same but they are not the same to MAPLE (this can be checked by the addressof command or, in this case, by the fact that they are different data types). This distinction is particularly important when an already current alias needs to be changed. The information of the current aliases is stored in _FUNCNAMES in the form of label = alias. In the output, _FUNCNAMES appears to be an expression sequence whose elements are of the form f = f.

The independent variables are defined in a global variable named varlist whereas the dependent variables are named in dependvar. In both cases the data type is a list. In an efficient implementation of the jet bundle approach to symmetry, an ordering is required on the independent variables. This is given by the ordering in varlist and is implemented by the procedure varorder.
The labels for the jet variables are constructed from the independent variables (\text{varlist}) and the dependent variables (\text{dependvar}) in the "usual" manner (modulo the absence of subscripting); that is if \( x \) and \( t \) are independent variables and \( u \) is a dependent variable, the coordinates of the jet bundle are denoted by \( x, t, u, ux, ut, uxx, uxt, utt, \ldots \) where the ordering in the mixed derivatives is given by \text{varorder}. With this labelling convention, the procedure \text{totalder} takes two arguments \( x, \text{jet} \) and writes a procedure for the total derivative called \( Dx \) in the variable \( x \). The degree of prolongation depends upon the variable \( \text{jet} \). If \( \text{jet} \) is an integer then \( Dx \) will be the pull back of total derivative to the \( \text{jet} \)th order jet bundle; for example \text{totalder}(x, 2) gives

\[
Dx := P \rightarrow \frac{\partial P}{\partial x} + ux \frac{\partial P}{\partial u} + uxx \frac{\partial P}{\partial ux} + uxt \frac{\partial P}{\partial ut},
\]

assuming that \text{varlist} := \{x, t\} and \text{dependvar} := \{u\}. However if \( \text{jet} \) is a list then the procedure \( Dx \) will be the pull back of the total derivative to bundle of order \( \text{jet}[i] \) in the dependent variable \text{dependvar}[i]. Thus \text{totalder}(t, \{2, 1\}) would yield

\[
Dt := P \rightarrow \frac{\partial P}{\partial t} + ut \frac{\partial P}{\partial u} + uxt \frac{\partial P}{\partial ux} + utt \frac{\partial P}{\partial ut} + wt \frac{\partial P}{\partial w},
\]

where \text{dependvar} := \{u, w\} and \text{varlist} := \{x, t\}. Once the total derivatives are formed, the procedure \text{prolong} will prolong a vector field \( v \) to a specified order. It takes arguments \( \text{func}, \text{var1}, \text{var2}, \ldots \) where \text{func} is a coefficient (of a dependent variable in the vector field \( v \)) and \( \text{var1}, \text{var2}, \ldots \) are the variables in which \text{func} is to be prolonged (assuming that the total derivatives have already been constructed to that order or higher).

\begin{verbatim}
totalder:=proc(x,jet) local i,j,k,m,localjet,jetlist, tmplist,jetvar,pos,zjet,zjetl,tmp,z;
  options 'Copyright 1993 by Mark Hickman';
  tmp:=Diff(P,x);
  for m from 1 to nops(dependvar) do
    z:=dependvar[m];
    if type(jet,list) then localjet:=jet[m] else localjet:=jet fi;
    if localjet > 0 then
      tmp:=tmp+cat(z,x)*Diff(P,z);
      jetlist:=varlist;
      for i from 2 to localjet do
        tmplist:=[i];
        for j in jetlist do
          tmp:=tmp+cat(z,j,tmplist)*Diff(P,z);
        od;
      od;
    fi;
  od;
end:
\end{verbatim}
for k in varlist do
    jetvar:=sort([op(j),k],varorder);
    if not member(jetvar, tmplist) and
       member(x,jetvar,'pos') then
        tmplist:=[op(tmplist),jetvar];
        zjet:=cat(z,op(1..(pos-1),jetvar),
                  op((pos+1)..i,jetvar));
        zjetl:=cat(z,op(jetvar));
        tmp:=tmp+zjetl*Diff(P,zjet)
    fi
od
jetlist:=tmplist
fi
od;
assign(D.x=convert( unapply(tmp,P),diff));
op(D.x)
end:

prolong:=proc(func) local jet,pos,i,target;
options 'Copyright 1993 by Mark Hickman';
jet:=[seq(args[i],i=2..nargs)];
if member(func,DEPCOEFF,'pos')
   or member(op(0,func),DEPCOEFF,'pos') then
    target:=func;
    for i from 1 to nops(varlist) do
        target:=target-INDEPCOEFF[i]*cat(dependvar[pos],
                                      varlist[i])
    od
else ERROR('invalid func', func) fi;
for i in jet do
    target:=D.i(target)
od;
for i in varlist do
    sort([op(jet),i],varorder);
    target:=subs(cat(dependvar[pos],op('))=0,target)
od
end:

In the procedure prolong, global variables INDEPCOEFF and DEPCOEFF
are lists of labels for the coefficients of the vector field \( v \) with those from IN-
DEPCOEFF giving the coefficients of the independent variables and DEPCOEFF giving those for the dependent variables; that is, if INDEPCOEFF:=[xi, tau] and DEPCOEFF:=[phi] then

\[ v = \xi \frac{\partial}{\partial x} + \tau \frac{\partial}{\partial t} + \phi \frac{\partial}{\partial u} \]

(assuming that varlist and dependvar are as above). Note that the lists INDEPCOEFF and DEPCOEFF can be longer than the number of independent and dependent variables for a given problem; the procedures will only use as many members of the list as there are independent and dependent variables. Thus it is possible to have a global choice for these lists which can be used in any symmetry determining problems.

The vector field \( v \) itself is constructed by the procedure vfield. The vector field itself is labelled by the given argument of vfield or generically by \( V \) if no argument is given.

\[ \text{vfield:=proc() local i,gen,tmp; options 'Copyright 1993 by Mark Hickman'; if nargs = 0 then tmp:= 'V' else tmp:=args[1]; gen:=0; for i from 1 to nops(varlist) do gen:=gen + INDEPCOEFF[i]*Diff(P,varlist[i]) od; for i from 1 to nops(dependvar) do gen:=gen + DEPCOEFF[i]*Diff(P,dependvar[i]) od; assign(tmp=convert( unapply(gen,P),diff)); op(tmp) end; \]

The procedure jetform will change a list of equations written using the inert Diff operator into their "jet bundle" form. For example, the equation

\[ \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \left( \frac{\partial u}{\partial x} \right)^2 \]

would become \( ut = uxx + ux^2 \) under jetform. Again the list varlist is used in this procedure.
The equations satisfied by the symmetry generator $v$ are given by

$$\text{pr } v[\Delta] = 0 \pmod{\Delta = 0}$$

where $\Delta$ is the set of differential equations in question and $\text{pr } v$ is the prolongation of the vector field $v$ (to the appropriate order jet bundle). With the above choices of dependent and independent variables,

$$\text{pr } v = v + \phi^x \frac{\partial}{\partial u^x} + \phi^t \frac{\partial}{\partial u^t} + \phi^{xx} \frac{\partial}{\partial u^{xx}} + \phi^{xt} \frac{\partial}{\partial u^{xt}} + \phi^{tt} \frac{\partial}{\partial u^{tt}} + \cdots$$

where the coefficient $\phi^x$ are given by $\text{prolong}(\phi, x)$ and so on. In most cases, constructing $\text{pr } v$ in full is not necessary. Only those coefficients of jet variables that occur in the differential equation need to be computed. The procedure \text{geneqn} will compute the equations that determine the symmetry generator. It will also compute the total derivatives to the appropriate order and prolong the required coefficients.
evars:={op(eqn)};
tmp:=NULL;
while not evars=tmp do
tmp:=evars;
evars:=map(var,tmp)
o;
evars:=subs(op(map(remove,varlist)),
op(map(remove,dependvar)),evars);
eqnvar:=[];
prv:=0;
for x in varlist do
for i from 1 to nops(dependvar) do
jetvar:=cat(dependvar[i],x);
if member(jetvar,evars) then
    eqnvar:=[op(eqnvar),[i,x]];
    prv:=prv+cat(op(0,DEPCOEFF[i]),x)*Diff(P,jetvar);
evars:=subs(jetvar=NULL,evars)
fi
od
od;
jetlist:=varlist;
while member(false,map(type,evars,constant)) do
tmplist:=NULL;
for y in jetlist do
for x in varlist do
jet:=sort([op(y),x],varorder);
if not member(jet,[tmplist]) then
tmplist:=tmplist,jet;
for i from 1 to nops(dependvar) do
    z:=dependvar[i];
    jetvar:=cat(z,op(jet));
    if member(jetvar,evars) then
        eqnvar:=[op(eqnvar),[i,op(jet)]];
        prv:=prv+cat(op(0,DEPCOEFF[i]),op(jet))*Diff(P,jetvar);
evars:=subs(jetvar=NULL,evars)
fi
od
fi
od
jetlist:=[tmplist]
o;
prv:=convert(unapply(prv,P),diff);
for i from 1 to nops(dependvar) do jetorder[i]:=0 od;
for x in eqnvar do
    y:=x[1];
    jetorder[y]:=max(jetorder[y],nops(x)-1)
od;
jetorder:=convert(jetorder,list);
for x in varlist do totalder(x,jetorder) od;
for x in eqnvar do
    y:=x[1];
    z:=op(2..nops(x),x);
    assign(cat(op(0,DEPCOEFF[y]),z)=prolong(DEPCOEFF[y],z))
od;
subs(op(eqn),map(v+prv,rearrange(eqn)))
end:

This procedure first computes the variables that occur in eqn by a repeated application of var. These are stored in the local variable evars. The independent and dependent variables are removed from this set (via remove). The higher jet variables are then constructed. Those that occur in evars are added to the list eqnvar and then removed from evars. This process is continued until evars consists of only constants. Thus if the original equations include arbitrary constants, these must be appended to the constants variable (and so become system constants) or, preferably, declared to be constants using domain; that is domain([],a) to declare a as a constant.

The procedure desolve "extends" dsolve to handle differential equations involved indexed quantities rather than functions. desolve(eqn,f,x, ...) will attempt to solve the equation eqn for f in the variable x. Unless it is called with other arguments, the arbitrary "constants" will be labelled _f1, _f2 and so on. If further arguments are given then these will be used for the constants. The constants are declared to be functions of all the variables of f except x. If the result is an explicit expression for f then desolve will assign f this value otherwise the implicit form will be echoed on the terminal.

desolve:=proc(eqn,u,x) local tmp,tmp1,tmpvar,i,constname;
options 'Copyright 1993 by Mark Hickman';
tmp:=dsolve(subs(u=u(x),eqn),u(x));
if tmp=NULL
    then RETURN(`Can not solve this differential equation.`)
else
    tmp1:=NULL;
end:
Finally there is a simple routine setup that will ask the user for the independent and dependent variables, the various coefficients of the generator and the differential equations themselves. It then sets up the various global variables that are needed by the other routines.

```plaintext
setup:=proc() local z,i,vars,flag,v;
options 'Copyright 1993 by Mark Hickman';
flag:=true;
print('Welcome to Symmetry');
print();
print('Please remember to end each input with a ;');
while flag do
  varlist:=[readstat('The independent variables are ')];
  dependvar:=[readstat('The dependent variables are ')];
  INDEPCOEFF:=[readstat('The coefficients of the independent variables are ')];
  DEPCOEFF:=[readstat('The coefficients of the dependent variables are ')];
```
In the first two appendicies, the use of this collection of routines is demonstrated with the computation of the symmetry group of the potential Burgers' equation and the computation of the partial symmetries of the modified Korteweg-de Vries equation.

### A Potential Burgers' Equation

The following is a record of a MAPLE V.2 session using the above tools to find the (well-known) symmetries of the potential Burgers' equation

\[
\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \left( \frac{\partial u}{\partial x} \right)^2.
\]

14
The Potential Burgers' Equation

Welcome to Symmetry

Please remember to end each input with a ;

The independent variables are \( x, t \);
The dependent variables are \( u \);
The coefficients of the independent variables are \( \xi, \tau \);
The coefficients of the dependent variables are \( \phi \);
The DEs are (use the Diff operator) \( \text{Diff}(u,t) = \text{Diff}(u,x)^2 + \text{Diff}(u,x)^2 \);

\[
\frac{\partial}{\partial t} u = \frac{\partial^2}{\partial x^2} u + \frac{\partial}{\partial x} u^2
\]

Are these correct? y or n (do not use a ;)

y

Note that the jet form of the equations are given by the variable eqn

\[
[ ut = uxx + ux^2 ]
\]

We can now construct the equations satisfied by the symmetry generators. Note that these will be in the form of a list but since we have only one equation in this case, it is more convenient to have the equation as an expression. Hence the use of op.
> eq:=op(geneqn(eqn));
All unknowns in this equation are functions of the original variables and so we can collect the higher order jet variables.

> eq:=collect(eq,[utt,uxt,uxx,ut,ux]);
We can now split off the various components in the higher jet variables. In practice, the simplest equations result from the highest order variables.

> coeff(eq,uxt);
We see that tau is independent of x and u. Remember that tau initially depended on x, t and u only. We can remind ourselves of this by using explicit.

> explicit(");

\[
2 \left( \frac{\partial}{\partial x} \tau \right) + 2 ux \left( \frac{\partial}{\partial u} \tau \right)
\]

We see that tau is independent of x and u. Remember that tau initially depended on x, t and u only. We can remind ourselves of this by using explicit.

> explict(");

\[
2 \left( \frac{\partial}{\partial x} \tau_{x,t,u} \right) + 2 ux \left( \frac{\partial}{\partial u} \tau_{x,t,u} \right)
\]

> domain([t],tau):
tau has already been declared

Do you wish to redefine tau? y or n

y

> explicit(coeff(eq,uxx));

\[
2 \left( \frac{\partial}{\partial u} \xi_{x,t,u} \right) ux - \left( \frac{\partial}{\partial t} \tau_{x,t,u} \right) + 2 \left( \frac{\partial}{\partial x} \xi_{x,t,u} \right)
\]
Thus tau is independent of u and linear in x (since tau depends only on t).

> domain([t],alpha):xi:=diff(tau,t)*x/2+alpha;

\[
\xi := \frac{1}{2} \left( \frac{\partial}{\partial t} \tau \right) x + \alpha
\]

> eq;

\[
\left( \frac{\partial}{\partial u} \phi - \frac{\partial^2}{\partial u^2} \phi \right) ux^2 + \left( -2 \left( \frac{\partial^2}{\partial x \partial u} \phi \right) - 2 \left( \frac{\partial}{\partial x} \phi \right) - \frac{1}{2} \left( \frac{\partial^2}{\partial t^2} \tau \right) x - \left( \frac{\partial}{\partial t} \alpha \right) \right) ux
\]

> coeff(eq,ux^2);
\[-\left(\frac{\partial}{\partial u} \phi\right) + \left(\frac{\partial^2}{\partial u^2} \phi\right)\]

phi must satisfy this simple DE in \( u \). In order to solve this we can use desolve.

> desolve("\, phi, u, A, B");

\[ A + B e^u \]

Note that \( A \) and \( B \) have automatically been declared functions of \( x \) and \( t \).

> explicit("");

\[ A_{[x, t]} + B_{[x, t]} e^u \]

> eq;
\[\left(-2 \left(\frac{\partial}{\partial x} A\right) - \frac{1}{2} \left(\frac{\partial^2}{\partial t^2} \tau\right) x - \left(\frac{\partial}{\partial t} \alpha\right)\right) u x - \left(\frac{\partial^2}{\partial x^2} A\right) - \left(\frac{\partial^2}{\partial x^2} B\right) e^u + \left(\frac{\partial}{\partial t} A\right) + \left(\frac{\partial}{\partial t} B\right) e^u\]

We have now determined the \( u \) dependence explicitly and so we can collect on \( u \).

> eq:=collect(eq,ux,exp(-u));

\[ eq := \left(-2 \left(\frac{\partial}{\partial x} A\right) - \frac{1}{2} \left(\frac{\partial^2}{\partial t^2} \tau\right) x - \left(\frac{\partial}{\partial t} \alpha\right)\right) u x + \left(\frac{\partial}{\partial t} B\right) - \left(\frac{\partial^2}{\partial x^2} B\right) e^u - \left(\frac{\partial^2}{\partial x^2} A\right) \]

Thus \( A \) is quadratic in \( x \).

> domain([t], beta); A := diff(tau, t$2)*x^2/8 - diff(alpha, t)*x/2 + beta;
Thus $B$ satisfies the heat equation and

$$\text{subs}\left(\exp(-u)=0,\right);$$

$$1\left(a_{2}\right)1\left(a_{2}\right)1\left(a_{1}\right)-\frac{1}{4}\left(\frac{\partial^{2}}{\partial t^{2}}\tau\right)-\frac{1}{8}\left(\frac{\partial^{3}}{\partial t^{3}}\tau\right)x^{2}-\frac{1}{2}\left(\frac{\partial^{2}}{\partial t^{2}}\alpha\right)x+\left(\frac{\partial}{\partial t}\beta\right)$$

must vanish. All unknowns in this expression depend only on $t$ and so we can equate powers of $x$.

$$\tau:=-a_{2}t^{2}+b_{2}t+c; \quad \alpha:=d_{2}t+e;$$

$$\beta:=-a_{2}t/2+f;$$

$$P \rightarrow \left(\frac{1}{2} \left(2a_{2}t+b_{2}\right)x+d_{2}t+e\left(\frac{\partial}{\partial x}P\right)\right)+\left(\frac{1}{2} \left(2a_{2}t+b_{2}\right)x+d_{2}t+e\left(\frac{\partial}{\partial t}P\right)\right)$$

$$+\left(-\frac{1}{4}a_{1}x^{2}-\frac{1}{2}d_{1}x\right)+\left(-\frac{1}{2}a_{1}t+f\right)\left(\frac{\partial}{\partial u}P\right)$$
Thus the symmetry group for the potential Burgers' equation is generated by
\[
\begin{align*}
\frac{\partial}{\partial x}, & \quad \frac{\partial}{\partial t}, & \quad \frac{\partial}{\partial u}, \\
x \frac{\partial}{\partial x} + 2t \frac{\partial}{\partial t}, & \quad 2t \frac{\partial}{\partial x} - x \frac{\partial}{\partial u}, & \quad 4tx \frac{\partial}{\partial x} + 4t^2 \frac{\partial}{\partial t} - (x^2 + 2t) \frac{\partial}{\partial u},
\end{align*}
\]
and
\[B(x,t)e^{-u} \frac{\partial}{\partial u}\]
where \(B\) is any solution of the heat equation.

\section*{B Modified Korteweg-de Vries Equation}

This is a more interesting example. Below is an abbreviated record of the computation of partial symmetries \cite{6} of the prolongation
\[
\begin{align*}
w_x & = v \\
w_t & = -v_{xx} + 2v^3
\end{align*}
\]
of the modified Korteweg-de Vries (mKdV) equation
\[v_t + v_{xxx} - 6v^2 v_x = 0.\]

In this case, the generators are defined on the space spanned by \(x, t, v, w\) and preserve the mKdV equation but not necessarily the prolongation equations. The total derivatives are only needed to first order in the potential variable \(w\) (hence the call \texttt{totalder}(x, \{3, 1\}) but this derivative has to be pulled back onto the jet bundle given by the dependent variable \(v\). This is achieved by the use of the equations satisfied by the potential variable. The appropriate \textsc{maple} instruction is \texttt{Dx:=subs(op(poteqn), op(Dx))}; where \texttt{poteqn} is a list that contains the equations satisfied by the potential variable.
Modified KdV Equation

> setup();

Welcome to Symmetry

Please remember to end each input with a ;

The independent variables are x,t;
The dependent variables are v,w;
The coefficients of the independent variables are xi,tau;
The coefficients of the dependent variables are phi,psi;
The DEs are (use the Diff operator) Diff(v,t)=-Diff(v,x$3)+6*v^2*Diff(v,x):

The independent variables are

[x, t]

The dependent variables are

[v, w]

The symmetry generator is

\[ P \rightarrow \xi \left( \frac{\partial}{\partial x} P \right) + \tau \left( \frac{\partial}{\partial t} P \right) + \phi \left( \frac{\partial}{\partial v} P \right) + \psi \left( \frac{\partial}{\partial w} P \right) \]

The differential equations are

\[ \frac{\partial}{\partial t} v = -\left( \frac{\partial^3}{\partial x^3} v \right) + 6 v^2 \left( \frac{\partial}{\partial x} v \right) \]

Are these correct? y or n (do not use a ;)

y

Note that the jet form of the equations are given by the variable eqn

We also need the equations satisfied by the potential.

> poteqn:=[Diff(w,x)=v,Diff(w,t)=Diff(v,x$2)+2*v^3];

\[ poteqn := \left[ \frac{\partial}{\partial x} w = v, \frac{\partial}{\partial t} w = \left( \frac{\partial^2}{\partial x^2} v \right) + 2 v^3 \right] \]
\[ poteqn := \text{jetform}(poteqn); \]

\[ poteqn := [wx = v, wt = -vxx + 2v^3] \]

\[ \text{totalder}(x,[3,1]): \text{totalder}(t,1); \]

The total derivatives need to be pull backed using the equations satisfied by the potential.

\[ \text{Dx} := \text{subs}(\text{op}(poteqn), \text{op}(\text{Dx})); \]

\[ \text{Dt} := \text{subs}(\text{op}(poteqn), \text{op}(\text{Dt})); \]

\[ \text{phix} := \text{prolong}(\text{phi}, x); \text{phixxx} := \text{prolong}(\text{phi}, x^3); \]

\[ \text{phit} := \text{prolong}(\text{phi}, t); \]

\[ \text{eq} := \text{subs}(\text{op}(\text{eqn}), \text{phit} + \text{phixxx} - 6v^2 \text{phix} - 12v \text{vx} \text{phi}); \]

\[ \text{eq} := \text{collect}(\text{eq}, [vtt, vxt, vxx, vxxx, vtt, vxt, vxx, vt, vx]); \]

\[ \text{coeff}(\text{eq}, vxxt); \]

\[ -3v \left( \frac{\partial}{\partial w} \tau \right) - 3 vx \left( \frac{\partial}{\partial v} \tau \right) - 3 \left( \frac{\partial}{\partial x} \tau \right) \]

\[ \text{domain}([t], \text{tau}); \]

tau has already been declared

Do you wish to redefine tau? y or n

y

\[ \text{coeff}(\text{eq}, vxxx); \]

\[ -3 \left( \frac{\partial}{\partial v} \xi \right) vx - 3 v \left( \frac{\partial}{\partial w} \xi \right) + \frac{\partial}{\partial t} \left( \frac{\partial}{\partial x} \xi \right) - 3 \left( \frac{\partial}{\partial x} \xi \right) \]

\[ \text{domain}([t], \text{alpha}); \text{xi} := \text{diff}(\text{tau}, t)*x/3 + \text{alpha}; \]

\[ \xi := \frac{1}{3} \left( \frac{\partial}{\partial t} \right) x + C \]

\[ \text{coeff}(\text{eq}, \text{vxx}); \]

\[ 3 \left( \frac{\partial^2}{\partial v^2} \phi \right) vx + 3 v \left( \frac{\partial^2}{\partial v \partial w} \phi \right) + 3 \left( \frac{\partial^2}{\partial v \partial x} \phi \right) \]

\[ \text{domain}([t], \text{beta}); \text{domain}([x, t, w], \text{A}); \text{phi} := \text{beta} * v + A; \]

\[ \phi := \beta v + A \]

\[ \text{eq} := \text{collect}(\text{eq}, [vt, vx, v]); \]

\[ \text{coeff}(\text{eq}, \text{vx}); \]

21
\[
\left(-12\beta - 4\frac{\partial}{\partial t}\tau\right)v^2 + \left(3\frac{\partial^2}{\partial w^2}A - 12A\right)v + 3\frac{\partial^2}{\partial x\partial w}A - \frac{1}{3}\frac{\partial^2}{\partial t^2}\tau x - \frac{\partial}{\partial t}\alpha
\]

\[\beta = -\frac{1}{3}\frac{\partial}{\partial t}\tau\]

\[\text{coeff}(\text{eq},v)\]

\[3\left(\frac{\partial^2}{\partial w^2}A\right) - 12A\]

\[\text{desolve}(\text{eq},A,w,Y,Z)\]

\[Ye^2w + Ze^{-2w}\]

\[\text{explicit(coeff(eq,vx))}\]

\[6\left(\frac{\partial}{\partial x}Y_{[x,t]}\right)e^2w - 6\left(\frac{\partial}{\partial x}Z_{[x,t]}\right)e^{-2w} - \frac{1}{3}\frac{\partial^2}{\partial t^2}\tau_{[t]} x - \frac{\partial}{\partial t}\alpha_{[t]}\]

\[\text{tau:=3*a+3*b*t;alpha:=c;}\]

\[\tau := 3\ a + 3\ b\ t\]

\[\alpha := c\]

\[\text{domain([t],Y,Z);}\]

\[y\]

\[y\]

\[Z\text{ has already been declared}\]

\[\text{Do you wish to redefine } Z? \ y \text{ or } n\]

\[Y\text{ has already been declared}\]

\[\text{Do you wish to redefine } Y? \ y \text{ or } n\]

\[\text{eq;}\]

\[\left(\frac{\partial}{\partial t}Y\right)e^2w + \left(\frac{\partial}{\partial t}Z\right)e^{-2w}\]

\[\text{Y:=d;Z:=e;}\]
\[ Y := d \]
\[ Z := e \]

> vfield();

\[ P \rightarrow (b \cdot x + c) \left( \frac{\partial}{\partial x} P \right) + (3 \cdot a + 3 \cdot b \cdot t) \left( \frac{\partial}{\partial t} P \right) + (-b \cdot v + d \cdot e^2 \cdot w + e \cdot e^2 \cdot w) \left( \frac{\partial}{\partial v} P \right) \]

\[ + \psi \left( \frac{\partial}{\partial w} P \right) \]

Thus the group of partial symmetries of the mKdV equation is generated by the following vector fields

\[ \frac{\partial}{\partial x}, \frac{\partial}{\partial t}, x \cdot \frac{\partial}{\partial x} + 3t \cdot \frac{\partial}{\partial t} - v \cdot \frac{\partial}{\partial v}, e^{2w} \cdot \frac{\partial}{\partial v} \text{ and } e^{-2w} \cdot \frac{\partial}{\partial v} \]

of which the last two are non-local and are not full symmetries of the prolongation equations.

C The diff routine

While the use of indexed data type rather than function data type to describe functions is unusual, it does have many advantages; particularly with respect to evaluation. MAPLE actually defines the function data type as a special type of procedure (and thus single level evaluation applies) whereas an indexed quantity is defined through a table type structure. It was surprising to find that diff did treat the indices as functional dependencies. However it became apparent that diff did have a bug when it came to differentiating indexed quantities as the following MAPLE session shows.
Thus `diff` gives the incorrect answer for expressions when an indexed quantity \( f[x] \) is multiplied by a monomial in \( x \) and then combined into another expression. If the product is on its own then `diff` gives the correct answer or if the factor is a polynomial or any other type of expression involving \( x \), the answer is also correct! In fact if \( f[x] \) is replaced by any expression that depends on it (for example, `diff(f[x], x)` or `sin(f[x])`) then the answer is also correct. Fortunately, in the code given above the exceptional case can never occur. However, it could, perhaps, occur when the resultant equations are solved or simplified. For this reason, the `diff` routine has been modified below so that the exceptional case can never occur. The resultant routine `mydiff` should be used if there is any doubt about the expression being differentiated. Hopefully this bug will be fixed in the next release of MAPLE and so `mydiff` will not be needed.

```
mydiff:=proc(expr) local vars,x1,x2;
vars:=seq(args[i],i=2..nargs);
if type(expr, '+') then RETURN(map(mydiff, expr, vars))
end if;
end proc;
```
elif type(expr,'.') then
    x1:=op(1,expr);
    x2:=expr-x1;
    RETURN(mydiff(x1,vars)*x2+x1*mydiff(x2,vars))
else RETURN(diff(expr,vars))
fi
end:

References


