The Package SPDE for Determining Symmetries of Partial Differential Equations

Fritz Schwarz
GMD, Institut F1
Postfach 1240
5205 St. Augustin
GERMANY

Telephone: +49-2241-142782
Email: fritz.schwarz@gmd.de

The package SPDE provides a set of functions which may be applied to determine the symmetry group of Lie- or point-symmetries of a given system of partial differential equations. Preferably it is used interactively on a computer terminal. In many cases the determining system is solved completely automatically. In some other cases the user has to provide some additional input information for the solution algorithm to terminate. The package should only be used in compiled form.

For all theoretical questions, a description of the algorithm and numerous examples the following articles should be consulted: "Automatically Determining Symmetries of Partial Differential Equations", Computing vol. 34, page 91-106(1985) and vol. 36, page 279-280(1986), "Symmetries of Differential Equations: From Sophus Lie to Computer Algebra", SIAM Review, to appear, and Chapter 2 of the Lecture Notes "Computer Algebra and Differential Equations of Mathematical Physics", to appear.

Function name	Operation
CRESYS(<*arguments*>)	Constructs determining system
SIMPSYS()	Solves determining system
RESULT()	Prints infinitesimal generators and commutator table

Table 1: SPDE Functions

Function name	Operation
PRSYS()	Prints determining system
PRGEN()	Prints infinitesimal generators
COMM(U,V)	Prints commutator of generators U and V

Table 2: SPDE Useful Output Functions

1 Description of the System Functions and Variables

The symmetry analysis of partial differential equations logically falls into three parts. Accordingly the most important functions provided by the package are:

Some other useful functions for obtaining various kinds of output are:

There are several global variables defined by the system which should not be used for any other purpose than that given in Table 3 and 4. The three globals of the type integer are:

In addition there are the following global variables of type operator:

The differential equations of the system at issue have to be assigned as values to the operator deq i applying the notation which is defined in Table 4. The entries in the third and the last line of that Table have obvious extensions

Variable name	Meaning
NN	Number of independent variables
MM	Number of dependent variables
PCLASS=0, 1 or 2	Controls amount of output

Table 3: SPDE Integer valued globals

Variable name	Meaning
X(I)	Independent variable x_i
U(ALFA)	Dependent variable u^{alfa}
U(ALFA,I)	Derivative of u^{alfa} w.r.t. x_i
DEQ(I)	i-th differential equation
SDER(I)	Derivative w.r.t. which DEQ(I) is resolved
GL(I)	i-th equation of determining system
GEN(I)	i-th infinitesimal generator
XI(I), ETA(ALFA) ZETA(ALFA,I)	See definition given in the references quoted in the introduction.
C(I)	i-th function used for substitution

Table 4: SPDE Operator type global variables

to higher derivatives.

The derivative w.r.t. which the i-th differential equation deq i is resolved has to be assigned to sder i. Exception: If there is a single differential equation and no assignment has been made by the user, the highest derivative is taken by default.

When the appropriate assignments are made to the variable deq, the values of NN and MM (Table 2) are determined automatically, i.e. they have not to be assigned by the user.

The function CRESYS may be called with any number of arguments, i.e.

```
CRESYS(); or CRESYS(deq 1,deq 2,... );
```

are legal calls. If it is called without any argument, all current assignments to deq are taken into account. Example: If deq 1, deq 2 and deq 3 have been assigned a differential equation and the symmetry group of the full system comprising all three equations is desired, equivalent calls are

```
CRESYS();   or   CRESYS(deq 1,deq 2,deq 3);
```

The first alternative saves some typing. If later in the session the symmetry group of deq 1 alone has to be determined, the correct call is

```
CRESYS deq 1;
```

After the determining system has bee created, SIMPSYS which has no arguments may be called for solving it. The amount of intermediate output produced by SIMPSYS is controlled by the global variable PCLASS with the default value 0. With PCLASS equal to 0, no intermediate steps are shown. With PCLASS equal to 1, all intermediate steps are displayed so that the solution algorithm may be followed through in detail. Each time the algorithm passes through the top of the main solution loop the message

```
Entering main loop
```

is written. PCLASS equal 2 produces a lot of LISP output and is of no interest for the normal user.

If with PCLASS=0 the procedure SIMPSYS terminates without any response, the determining system is completely solved. In some cases SIMPSYS does not solve the determining system completely in a single run. In general this is true if there are only genuine differential equations left which the algorithm cannot handle at present. If a case like this occurs, SIMPSYS returns the remaining equations of the determining system. To proceed with the solution algorithm, appropriate assignments have to be transmitted by the user, e.g. the explicit solution for one of the returned differential equations. Any new functions which are introduced thereby must be operators of the form c(k) with the correct dependencies generated by a depend statement (see the "REDUCE User's Guide"). Its enumeration has to be chosen in agreement with the current number of functions which have alreday been introduced. This value is returned by SIMPSYS too.

After the determining system has been solved, the procedure RESULT, which has no arguments, may be called. It displays the infinitesimal generators and its non-vanishing commutators.

2 How to Use the Package

In this Section it is explained by way of several examples how the package SPDE is used interactively to determine the symmetry group of partial differential equations. Consider first the diffusion equation which in the notation given above may be written as

```
deq 1:=u(1,1)+u(1,2,2);
```

It has been assigned as the value of deq 1 by this statement. There is no need to assign a value to sder 1 here because the system comprises only a single equation.

The determining system is constructed by calling

```
  CRESYS(); or CRESYS deq 1;
```

The latter call is compulsory if there are other assignments to the operator deq i than for i=1.

The error message

```
  ***** Differential equations not defined
```

appears if there are no differential equations assigned to any deq.

If the user wants the determining system displayed for inspection before starting the solution algorithm he may call

```
  PRSYS();
```

and gets the answer

```
  GL(1):=2*DF(ETA(1),U(1),X(2)) - DF(XI(2),X(2),2) -
         DF(XI(2),X(1))

  GL(2):=DF(ETA(1),U(1),2) - 2*DF(XI(2),U(1),X(2))

  GL(3):=DF(ETA(1),X(2),2) + DF(ETA(1),X(1))

  GL(4):=DF(XI(2),U(1),2)

  GL(5):=DF(XI(2),U(1)) - DF(XI(1),U(1),X(2))

  GL(6):=2*DF(XI(2),X(2)) - DF(XI(1),X(2),2) - DF(XI(1),X(1))

  GL(7):=DF(XI(1),U(1),2)

  GL(8):=DF(XI(1),U(1))

  GL(9):=DF(XI(1),X(2))

The remaining dependencies

  XI(2) depends on U(1),X(2),X(1)

  XI(1) depends on U(1),X(2),X(1)
```

```
ETA(1) depends on U(1),X(2),X(1)
```

The last message means that all three functions XI(1), XI(2) and ETA(1) depend on X(1), X(2) and U(1). Without this information the nine equations GL(1) to GL(9) forming the determining system are meaningless. Now the solution algorithm may be activated by calling

```
 SIMPSYS();
```

If the print flag PCLASS has its default value which is 0 no intermediate output is produced and the answer is

```
Determining system is not completely solved

The remaining equations are

GL(1):=DF(C(1),X(2),2) + DF(C(1),X(1))

Number of functions is 16

The remaining dependencies

C(1) depends on X(2),X(1)
```

With PCLASS equal to 1 about 6 pages of intermediate output are obtained. It allows the user to follow through each step of the solution algorithm.

In this example the algorithm did not solve the determining system completely as it is shown by the last message. This was to be expected because the diffusion equation is linear and therefore the symmetry group contains a generator depending on a function which solves the original differential equation. In cases like this the user has to provide some additional information to the system so that the solution algorithm may continue. In the example under consideration the appropriate input is

```
 DF(C(1),X(1)) := - DF(C(1),X(2),2);
```

If now the solution algorithm is activated again by

```
SIMPSYS();
```

the solution algorithm terminates without any further message, i.e. there are no equations of the determining system left unsolved. To obtain the symmetry generators one has to say finally

```
RESULT();
```

and obtains the answer

```
The differential equation

DEQ(1):=U(1,2,2) + U(1,1)


The symmetry generators are

GEN(1):= DX(1)

GEN(2):= DX(2)

GEN(3):= 2*DX(2)*X(1) + DU(1)*U(1)*X(2)

GEN(4):= DU(1)*U(1)

GEN(5):= 2*DX(1)*X(1) + DX(2)*X(2)

                        2
GEN(6):= 4*DX(1)*X(1)

       + 4*DX(2)*X(2)*X(1)

                            2
         + DU(1)*U(1)*(X(2)  - 2*X(1))

GEN(7):= DU(1)*C(1)

The remaining dependencies

C(1) depends on X(2),X(1)


Constraints
```

```
DF(C(1),X(1)):= - DF(C(1),X(2),2)


The non-vanishing commutators of the finite subgroup


COMM(1,3):= 2*DX(2)

COMM(1,5):= 2*DX(1)

COMM(1,6):= 8*DX(1)*X(1) + 4*DX(2)*X(2) - 2*DU(1)*U(1)

COMM(2,3):= DU(1)*U(1)

COMM(2,5):= DX(2)

COMM(2,6):= 4*DX(2)*X(1) + 2*DU(1)*U(1)*X(2)

COMM(3,5):=  - (2*DX(2)*X(1) + DU(1)*U(1)*X(2))

                             2
COMM(5,6):= 8*DX(1)*X(1)

          + 8*DX(2)*X(2)*X(1)

                                  2
          + 2*DU(1)*U(1)*(X(2)  - 2*X(1))
```

The message “Constraints” which appears after the symmetry generators are displayed means that the function c(1) depends on x(1) and x(2) and satisfies the diffusion equation.

More examples which may used for test runs are given in the final section.

If the user wants to test a certain ansatz of a symmetry generator for given differential equations, the correct proceeding is as follows. Create the determining system as described above. Make the appropriate assignments for the generator and call PRSYS() after that. The determining system with this ansatz substituted is returned. Example: Assume again that the determining system for the diffusion equation has been created. To check the correctness for example of generator GEN 3 which has been obtained above, the assignments

```
XI(1):=0;  XI(2):=2*X(1);  ETA(1):=X(2)*U(1);
```

have to be made. If now PRSYS() is called all GL(K) are zero proving the correctness of this generator.

Sometimes a user only wants to know some of the functions ZETA for for various values of its possible arguments and given values of MM and NN. In these cases the user has to assign the desired values of MM and NN and may call the ZETAs after that. Example:

```
MM:=1;  NN:=2;

FACTOR U(1,2),U(1,1),U(1,1,2),U(1,1,1);

ON LIST;

ZETA(1,1);

-U(1,2)*U(1,1)*DF(XI(2),U(1))

-U(1,2)*DF(XI(2),X(1))

       2
-U(1,1) *DF(XI(1),U(1))

+U(1,1)*(DF(ETA(1),U(1)) -DF(XI(1),X(1)))

+DF(ETA(1),X(1))


ZETA(1,1,1);

-2*U(1,1,2)*U(1,1)*DF(XI(2),U(1))

-2*U(1,1,2)*DF(XI(2),X(1))

-U(1,1,1)*U(1,2)*DF(XI(2),U(1))
```

```
-3*U(1,1,1)*U(1,1)*DF(XI(1),U(1))

+U(1,1,1)*(DF(ETA(1),U(1)) -2*DF(XI(1),X(1)))

                2
-U(1,2)*U(1,1) *DF(XI(2),U(1),2)

-2*U(1,2)*U(1,1)*DF(XI(2),U(1),X(1))

-U(1,2)*DF(XI(2),X(1),2)

       3
-U(1,1) *DF(XI(1),U(1),2)

       2
+U(1,1) *(DF(ETA(1),U(1),2) -2*DF(XI(1),U(1),X(1)))

+U(1,1)*(2*DF(ETA(1),U(1),X(1)) -DF(XI(1),X(1),2))

+DF(ETA(1),X(1),2)
```

If by error no values to MM or NN and have been assigned the message

```
***** Number of variables not defined
```

is returned. Often the functions ZETA are desired for special values of its arguments ETA(ALFA) and XI(K). To this end they have to be assigned first to some other variable. After that they may be evaluated for the special arguments. In the previous example this may be achieved by

```
Z11:=ZETA(1,1)$   Z111:=ZETA(1,1,1)$
```

Now assign the following values to XI 1, XI 2 and ETA 1:

```
XI 1:=4*X(1)**2; XI 2:=4*X(2)*X(1);

ETA 1:=U(1)*(X(2)**2  - 2*X(1));
```

They correspond to the generator GEN 6 of the diffusion equation which has been obtained above. Now the desired expressions are obtained by calling

```
Z11;
```

```
                                          2
 - (4*U(1,2)*X(2) - U(1,1)*X(2)  + 10*U(1,1)*X(1) + 2*U(1))

  Z111;

                                              2
 - (8*U(1,1,2)*X(2) - U(1,1,1)*X(2)  + 18*U(1,1,1)*X(1) +
   12*U(1,1))
```

3 Test File

This appendix is a test file. The symmetry groups for various equations or systems of equations are determined. The variable PCLASS has the default value 0 and may be changed by the user before running it. The output may be compared with the results which are given in the references.

```
%The Burgers equations

deq 1:=u(1,1)+u 1*u(1,2)+u(1,2,2)$

cresys deq 1$ simpsys()$ result()$

%The Kadomtsev-Petviashvili equation

deq 1:=3*u(1,3,3)+u(1,2,2,2,2)+6*u(1,2,2)*u 1

       +6*u(1,2)**2+4*u(1,1,2)$

cresys deq 1$ simpsys()$ result()$

%The modified Kadomtsev-Petviashvili equation

deq 1:=u(1,1,2)-u(1,2,2,2,2)-3*u(1,3,3)

       +6*u(1,2)**2*u(1,2,2)+6*u(1,3)*u(1,2,2)$

cresys deq 1$ simpsys()$ result()$
```

```
%The real- and the imaginary part of the nonlinear
%Schroedinger equation

deq 1:= u(1,1)+u(2,2,2)+2*u 1**2*u 2+2*u 2**3$

deq 2:=-u(2,1)+u(1,2,2)+2*u 1*u 2**2+2*u 1**3$

%Because this is not a single equation the two assignments

sder 1:=u(2,2,2)$  sder 2:=u(1,2,2)$

%are necessary.

cresys()$ simpsys()$ result()$

%The symmetries of the system comprising the four equations

deq 1:=u(1,1)+u 1*u(1,2)+u(1,2,2)$

deq 2:=u(2,1)+u(2,2,2)$

deq 3:=u 1*u 2-2*u(2,2)$

deq 4:=4*u(2,1)+u 2*(u 1**2+2*u(1,2))$

sder 1:=u(1,2,2)$ sder 2:=u(2,2,2)$ sder 3:=u(2,2)$
sder 4:=u(2,1)$

%is obtained by calling

cresys()$ simpsys()$

df(c 5,x 1):=-df(c 5,x 2,2)$

df(c 5,x 2,x 1):=-df(c 5,x 2,3)$

simpsys()$  result()$

% The symmetries of the subsystem comprising equation 1
%  and 3 are obtained by

cresys(deq 1,deq 3)$ simpsys()$ result()$

% The result for all possible subsystems is discussed in
% detail in ``Symmetries and Involution Systems: Some
% Experiments in Computer Algebra'', contribution to the
% Proceedings of the Oberwolfach Meeting on Nonlinear
% Evolution Equations, Summer 1986, to appear.
```