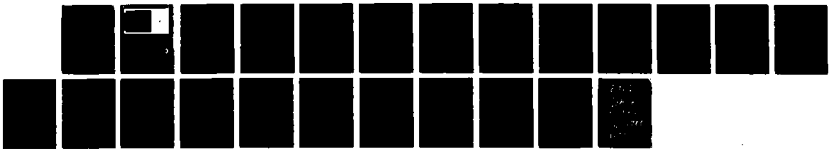
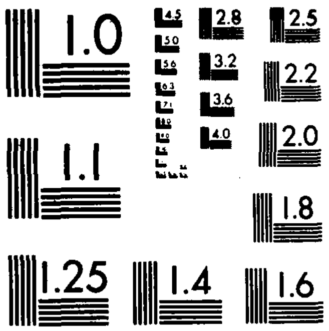
AD-A193 476 MACSYMA AT CMS VERSION 3093(U) WISCONSIN UNIV-MADISON CENTER FOR MATHEMATICAL SCIENCES W HEREMAN ET AL. AUG 87 CMS-TSR-88-3 DAAL03-87-K-0028 1/1

UNCLASSIFIED F/G 12/5 NL




1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

CMS Technical Summary Report #88-3

MACSYMA AT CMS. VERSION 309.3

W. Hereman, Y. Nagel and
J. Strikwerda

UNIVERSITY OF WISCONSIN

CENTER FOR THE MATHEMATICAL SCIENCES

Center for the Mathematical Sciences
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53705

August 1987

(Received August 7, 1987)

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

Air Force Office of Scientific Research
Bolling AFB
Washington, DC 20332

UNIVERSITY OF WISCONSIN-MADISON
CENTER FOR THE MATHEMATICAL SCIENCES

MACSYMA AT CMS. VERSION 309.3

W. Hereman, Y. Nagel and J. Strikwerda

Technical Summary Report #88-3
August 1987

ABSTRACT

This report provides a brief introduction to the use of **MACSYMA**, a symbolic manipulation program for mathematics.

The first Chapter outlines invoking Macsyma, running tutorials and demos, saving transcripts and programs, editing and plotting facilities.

The second Chapter provides 12 examples of Macsyma use. The following topics are touched upon: factorization, integration, matrix multiplications, eigenvalue problem, infinite series, recursion definitions and relations, solving systems of polynomial equations and ODEs.

Some examples in Chapter 2 and the intricate example given in Chapter 3 demonstrate the possibilities of programming in Macsyma.

AMS (MOS) Subject Classification: 68Q40

Key Words: macsyma, symbolic manipulation, symbolic programming

Supported in part by the U. S. Army Research Office under Grant No. DAAL03-87-K-0028 and the Air Force Office of Scientific Research under Grant No. 85-0263.

Contents

MACSYMA AT CMS. VERSION 309.3

W. Hereman, Y. Nagel and J. Strikwerda

Chapter 1

Introduction

Macsyma is a symbolic manipulation language for mathematics written in Lisp. It was originally developed by the Mathlab group of the MIT Computer Science laboratory. It is now supported and distributed by Symbolics, Inc. of Cambridge, Massachusettes. It can perform complicated mathematical computations involving integration, differentiation and matrix or function manipulations. It can also plot curves on suitable graphics terminals.

This document is an introductory tutorial to using Macsyma on the CMS Vax 11/780. It is not intended to be a comprehensive guide or reference. Basic references for Macsyma are:

Vax Unix Macsyma	Applications of	Macsyma: A Program for
Reference Manual	Macsyma to Calculations	Computer Algebraic
Version 11	in Applied Mathematics	Manipulation
Symbolics, Inc.	M.Hussain & B. Noble	Naval Underwater Systems
Cambridge, Mass	GE Report#83CRD054	Center, Newport, R.I.

The first book is the basic guide to Macsyma. It is available at the MACC documentation center. It is not a good tutorial although it does contain a list of other references. The other two books have many examples suitable for learning Macsyma. Some of these examples are quite complicated.

1.1 Invoking Macsyma

To use Macsyma on the CMS VAX you must first log in to the computer (see the System Manager if you do not have an account). When you first log in you will see the VMS **$** prompt. The CMS VAX has Eunice, a unix emulator, as well as the basic VMS operating system. You must run Macsyma from Eunice. To enter Eunice, you first type **unix** or **eunice**. You will then get the unix prompt - usually **a %** or ->. Now type in **/usr/macsyma** to start macsyma. Macsyma will return the **(c1)** prompt.

```
$ unix
% /usr/macsyma
(c1):
```

As you work through an example, Macsyma prints (cn), where "n" is an integer, for user input lines and **(dn)** or **(en)** for its own replies. The **(en)** lines denote intermediate calculations used by Macsyma for especially long computations or lists of output. Lines are numbered consecutively and can be used to recall previous commands or responses. All Macsyma commands must end with either a **$** or a ;. Use the **$** if you don't want a d-line to echo your input or to avoid intermediate output. Lines enclosed in double quotes or /* · · · */ are treated as comments (see Example 12 in Chapter 2). You can include such lines (terminate them with a **$**) in your Macsyma session to

Supported in part by the U. S. Army Research Office under Grant No. DAAL03-87-K-0028 and the Air Force Office of Scientific Research under Grant No. 85-0263.

remind you of the purpose of the calculation. See the next section for instructions on saving your output in a transcript file. The percent sign, %, can be used to refer to the last expression. In the examples, **boldface** indicates the characters typed in by the user. Normal letters show characters produced by the computer.

Some special characters are:

%i	for the imaginary number,
%e	for the base of the natural logarithms,
%pi	for π,
%	refers to expression on previous line
%th(n)	refers to expression on nth previous line
inf	for ∞.

The Macsyma **part** command allows you to refer to a part of the nth previous expression (see the examples in Chapter 2.)

Note that the exponential function with argument x can be typed in two different ways

% e^x
or **exp(x)**

To exit from Macsyma use **quit()**;. **Ctrl c** will force an interrupt. You can then type **h** to see what options you have. The most useful ones are **reset** which returns Macsyma to a c-line and **quit** which will make you exit from Macsyma and return to Eunice. Use **ctrl c** or **ctrl y** to end a runaway calculation. **Ctrl z** will toggle you into or out of LISP.

1.2 Tutorial and Demos

An on-line tutorial for Macsyma is available. To access it, type in **/usr/macsyma** then

$\boxed{\textbf{(c1)primer();}}$

A menu will appear. Typing in the number of an item on the menu will cause a Macsyma script to appear with appropriate examples and instructions.

Macsyma also has two built-in libraries of examples. The *share* library **/usr/mac309/share** can be used with the **example** command. If at any point (after typing in **/usr/macsyma** you would like to see how a function can be used, type in **example(*function*)** for a brief tutorial. For instance,

(cn) **example(ode2);**

will yield a brief introduction (with examples) of the ode2 solver. To read a description of a *command, option*, or *concept* type, e.g.

(cn) **describe(stringout);**

will give you a description of the command *stringout*. See the list of special forms in the manual index for other possible *commands*. This is a list of the **.dem** files in **/usr/mac309/share**:

absimp	dimen	nchrpl	rncomb
antid	eigen	ode2	sqdnst
dblint	facexp	optmiz	submac
delta	fourie	pfaff	trgsmp
differ	lrats	polsol	vect

The demonstration library **/usr/mac309/demo** also contains many examples. You can select demos from this library by typing

```
(c1)batch("/usr/mac309/demo/function.dem";
```

or

```
(c1)demo("/usr/mac309/demo/function.dem");
```

The **batch** command causes Macsyma to run through the demo without stopping (as though you were looking at a movie). With the **demo** command. Macsyma will pause after each display and show an underscore _ cursor. Hitting the **Return** key will cause it to continue. To get a list of the files in these two directories type **ls /usr/mac309/demo** or **ls /usr/mac309/share** from unix - i.e. at the % prompt.

Here is a list of the **.dem** files in **/usr/mac309/demo**:

algfac	differ	matrix	series
algsys	difficult	mpg	short
array	ezgcd	nalgfc	simpl
ball	factor	newfac	sin
begin	gen	nisimp	solder
c2cyl	iap	pfacto	solve
cf	int	plot	specfn
combin	laplac	poplot	subscr
cyl2c	legen	qpr	sum
cyl2s	limit	radcan	taylor
decomp	macex	ratsimp	trace
defint	macro	ratsub	trig
demall	maneg2	risch	
demtest	maneg3	rough	
dice	mat	rpart	

1.3 Saving Macsyma Output

You can save your Macsyma output as either a TRANSCRIPT file or a LISP file. The TRANSCRIPT file contains all the c- and d-lines which appear on your terminal during an interactive session with Macsyma. The LISP file contains LISP commands generated by Macsyma and can be used to playback or restart your last interactive session.

1.3.1 Transcript file

To save just the screen output (with displayed equations) do the following, type **writefile** at the beginning and **closefile** at the end of a computation as in the following example

```
(cm) writefile(foo):      - logs to file "foo"
  •
  •
  •
(cn) closefile();         - closes log file "foo"
```

Get out of macsyma. Send "foo" to the printer. "foo" cannot be reloaded to Macsyma. See section 10-17 of the Macsyma manual. If you type **quit();** without doing a **closefile();**, a transcript file will be created and saved in your directory anyway. If you forget to create a **writefile** when you begin your macsyma session, you can create one at any time.

```
(cm) writefile(foo);          - logs to file "foo"
(cm+1) playback(all);         - repeats your entire session
(cn) closefile();             - closes log file "foo"
```

If we type for line (cm+1)

playback(c35,c39);

Macsyma would only playback lines c35 thru c39. This allows you to be more selective about what you save.

1.3.2 LISP file

To save LISP commands, suitable for reloading or playing back your session see section 10-18 of the Macsyma manual.

Enter various Macsyma commands. At end of session do this:

```
(cm) save(soo,all);   - saves all Macsyma commands in Lisp
                        in the file "soo"

(cn) quit();          Exit from Macsyma
```

If you **quit();** before typing **save(soo);**, you will not be able to recover the LISP transcript of your session. To reload the LISP file just re-enter Macsyma and issue this as the first command:

```
(c1) loadfile(soo);     - Reloads "soo" Lisp file into Macsyma
(c2) playback(soo);     - Recreates your last session with "soo"
                        This is optional. It's not required to do
                        further computations. See 10-4 in manual
```

An efficient way to save a single line of output (say the final result of your macsyma computations) in LISP form is shown in the following example:

```
(c1) depends([f,v],[x1,x2,x3]);
(c2) f:(x1)^2*(x2)*(x3) + (x1)*(x2)^2*(x3) -   (x1)*(x2)*(x3)^2;
(c3) v(f):[diff(f,x1),diff(f,x2),diff(f,x3)];  - v(f) is the gradient of f
(d3)      •  •  •                              - Macsyma's answer is on this line
(c4) stringout(grad,c4);                       save (c4) in Macsyma file grad
(c5) quit();                                   out of macsyma
% /usr/macsyma                                 reenter macsyma
(c1) f:x1 + x2^2 + x3^3;                       declare f
(c2) batch(grad);                              input definition of grad
(c3) v(f):[diff(f,x1),diff(f,x2),diff(f,x3)];  machine returns answer
```

You can use the *stringout* command to save selected **c-**, **d-**, or **e=** lines. The most useful possibilities are

```
stringout(filename,ck)       saves line (ck)
stringout(filename,cj,ck)    saves lines (cj) & (ck)
stringout(filename,[j,k])    saves lines (cj) thru (ck)
```

You can examine the contents of *grad* by typing **type grad** at the VMS dollar sign prompt. You can also import *grad* into Macsyma via the command **batch(grad);**

1.3.3 Combining two transcripts

Suppose we have saved only the Lisp commands in the file "soo". Now we want to get the displayed equations. Do this

```
(c1) writefile(doo);
(c2) loadfile(soo);
(c3) playback();        don't need to mention soo
(c4) closefile();
(c5) quit();            get out of Macsyma
% lpr doo               send "doo" to laser printer
```

1.4 Running Jobs in the Background

MACSYMA version 309.3, unlike the older version of MACSYMA cannot be called from VMS. This means that macsyma cannot be run on the VMS batch queues. Instead, you can run a macsyma job in the background as follows:

1. First, use an editor to create a file, called, say, JUNK. which contains the MACSYMA commands you would have typed at the terminal if you were using MACSYMA interactively. A sample file might consist of these lines

```
(c1) writefile(trash)$
(c2) f(x):= 1/tan(x);
(c3) diff(f(x),x);
(c4) closefile();
(c5) quit();
```

2. Notice that JUNK and TRASH are different files. JUNK is a file you create with an editor. TRASH is the file Macsyma will create to save the output of your program. There are three possible ways to pipe JUNK into MACSYMA

   (a) **/usr/macsyma < junk**
   This is closest to running the job interactively.

   (b) **/usr/macsyma < junk. &**
   This pipes JUNK into MACSYMA and puts it into the background

   (c) **nice + <PN> /usr/macsyma < junk. &**
   This pipes JUNK into MACSYMA, puts it into the background, and assigns it PRIORITY PN. Acceptable choices for PN are integers in the range 10 thru 20. The higher the number, the lower the priority; 10 is the default.

1.5 Using the "vi" Editor with Macsyma

The default editor for MACSYMA is TECO. If you wish to use a different editor, say "vi" with Macsyma, create a file called MACSYMA.MAC (with edt or vi or any editor you like) which contains the following lines:

```
/*-*-Macsyma-*-*/
load("ucb//newedit.o");
editorcom:"vi macsyma.buf";
```

MACSYMA.MAC is an initialization file for MACSYMA. It must be in your home directory (the directory you are in when you first log on). Next, type the following VMS command

```
$ set term/noline_edit
```

Follow the procedure given above for invoking MACSYMA interactively. Now suppose, that while I am in macsyma, I type the following at command line (c1):

`(c1) absdj;` - no <RET>

If I now hit the <ESC> and <RET> keys, I will put this line into the editor, suitable for editing with vi. If by mistake, I enter LISP, I can type:

```
quit
```

to leave MACSYMA altogether or **ctrl z** to return from LISP to (cn).

If I have gone considerably beyond line (c1) but want to re-edit it to use it again, I would type:

```
string(c1); <ESC> <RET>
```

and line (c1) will be put back into the buffer. Although this technique will allow you to edit long lines, it is better to enter long lines by breaking them up into shorter pieces and combining them step by step.

1.6 Plotting on a Tektronix with Macsyma

Macsyma will plot on any terminal which emulates a Tektronix 4010 or 4014 terminal. At CMS some of the Visual 102 graphics terminals do this. IBM PCs with suitable graphics cards, graphics monitors and Tektronix emulation software can do this also. You must first put your terminal or PC in Tektronix emulation mode before attempting to plot. Here is an example using the Macsyma function *plot2*

First, put your PC or terminal into Tektronix emulation mode

```
$ unix
% set term=4014
% /usr/macsyma
(c1) plotmode:tek:
(c2) plot2([x+1,x^2+1],x,-1,1);
```

Chapter 2

Examples

You can try the examples (below) to get an easy introduction to Macsyma

2.1 Example 1 - polynomial factoring

To factor the polynomial $x^{99}+1$ enter:

```
(c1) x^99 + 1;
(c2) factor(%);
```

2.2 Example 2 - repeating a command

```
(c1) f(x):= 1/tan(x);
   (c2) integrate(f(x),x);
   (c3) f(x):= x/((x+1)*(x^2+1)^2);
   (c4) ' '(c2);
   (c5) quit();
```

Note the use of the two single quotes on line (c4) to repeat the command on line (c2).

2.3 Example 3 - matrix multiplication

Macsyma has two different kinds of matrix products. An asterisk * will produce an element-wise product, while a period • yields a true matrix product. Here is an example

(c1) entermatrix(2,2);
(c2) M:%;
Macsyma will assign the label m (or M - Macsyma is <u>case-insensitive</u> to the matrix above
(c3) M*M;
Macsyma will square each element of the matrix
(c4) M2;**
This will yield the same result as line (c3)
(c5) M.M;
Macsyma will return the true matrix product
(c6) M^^2;
This will yield the same result as line (c5)

2.4 Example 4 - matrix eigenvalues

To find the eigenvalues of the matrix

$$\begin{pmatrix} 1 & a \\ b & 1 \end{pmatrix}$$

enter:

```
(c1) entermatrix(2,2);
Is the matrix 1.Diagonal 2.Symmetric 3.Antisymmetric 4.General
Answer 1, 2, 3, or 4
4;
Row 1 Column 1: 1;
· · · Put in remaining entries. Remember the semicolon!
(c2) eigenvalues(%);
```

will give you the eigenvalues.

2.5 Example 5 - infinite sums

To evaluate the sum $\sum_{j=0}^{\infty} x^j$ enter:

```
(c1) assume( abs(x) < 1);
(c2) sum(x^j , j, 0, inf);
(c3) simpsum:true$
(c4) ev(d2,sum);
(c5) forget( abs(x) < 1)$
(c6) ev(d2,sum);
Is abs(x) - 1 positive, negative, or zero?
negative;
```

$$\frac{1}{1-x}$$

```
(c7) ev(d2,sum);
Is abs(x) - 1 positive, negative, or zero?
zero;
```

undefined

```
(c8) ev(d2,sum);
Is abs(x) - 1 positive, negative, or zero?
positive;
```

inf

If you don't put in the *assume* instruction (line (c1)) you will be queried on the size of x as in lines (c6) thru (c8).

2.6 Example 6 - adding and subtracting series

This example shows how to add and subtract infinite sums.

```
(c1) sum(a[k] + b[k],k,0,inf);
(c2) sum(b[k],k,0,inf);
(c3) d1 - d2;
(c4) simpsum:false;
(c5) intosum(%);
```

```
    (c6) sumcontract(%);
```
The *simpsum* option must be set to false in order for **intosum** to work properly. Without the *intosum* command the *sumcontract* will not do anything, because of the minus sign in front of the second sum. If the sums were added then there is no need for the *intosum*. For example,
```
    (c1) sum(a[k] - b[k],k,0,inf);
    (c2) sum(b[k],k,0,inf);
    (c3) d1 + d2;
    (c4) sumcontract(%);
```
will produce the same answer as above, i.e. the sum of the a_k.

2.7 Example 7 - a bug in macsyma

To evaluate the integral

$$\int_0^\infty \frac{1}{x^2+a}dx$$

enter:
```
    (c1) 'integrate(1/(x^2 + a), x,0,inf);
    Macsyma will respond by displaying the integral.
    (c2) ev(%,integrate,a=1);
    will produce the result:
    (d2)                                    %pi /2
```
The use of *'integrate* produces the expression of the integral, while using *integrate* would cause evaluation of the expression. Continuing this example, we may wish to evaluate the derivative of (c1) with respect to a.
```
    (c3) diff(c1,a);
    (c4) assume(a>0);
    (c5) ev(d3:integrate);
```
Macsyma gives a strange result for this,

$$-i\frac{\log(-1)}{4}.$$

If we now type
```
    (c6) lognegint:true$
```
Macsyma will return

$$\frac{\pi}{4a^{3/2}}$$

which has the wrong sign

2.8 Example 8 - checking an equation

This example shows that the polynomials, $p_k(x)$, defined by

$$\exp(\frac{xs}{1-s}) = \sum_{k=0}^{\infty} s^k p_k(x)$$

satisfy the equation

$$xp_k'(x) = kp_k(x) - (k-1)p_{k-1}(x).$$

We will only verify it for k from 0 through 6.

```
    (c1) exp((x*s)/(1-s));
    (c2) taylor(%,s,0,6);
    (c3) expand(%);
    (c4) for i from 0 thru 6 do a[i]:ratcoeff(d3,s,i);
    note that if we had written instead
       for i thru 6 do a[i]:ratcoeff(c3,s,i);
    macsyma would accept this command but do only terms 1 thru 6
    (c5) for i from 0 thru 6 do b[i]:x*diff(a[i],x) - i*a[i] +(i-1)*a[i-1];
    At this point the command
    (c6) for i from 0 thru 6 do ldisplay(b[i]);
will display polynomials but they will not be simplified yet. Use
    (c7) for i from 0 thru 6 do b[i]:expand(b[i]);
    (c8) for i from 0 thru 6 do ldisplay(b[i]);
    to see all the expected zeroes. Note that the command
    (c7) for i from 0 thru 6 do expand(b[i]);
```

would do no good since the result of the *expand* is not kept anywhere. Instead of *expand* one could also use *ratsimp* or *gfactor*

2.9 Example 9 - solving systems of equations

The solve command can be used on systems of linear or polynomial equations in several variables. To find all complex solutions to $x + y + z = 2$, $x^2 + y^2 + z^2 = 26$, $x^3 + y^3 + z^3 = 38$, enter:

```
    (c1) x+y+z=2;
    (c2) x**2+y**2+z**2=26;
    (c3) x**3+y**3+z**3=38;
    (c4) solve([d1,d2,d3],[x,y,z]);

(d4) [[x = - 3, y = 1, z = 4], [x = - 3, y = 4, z = 1],

[x = 4, y = - 3, z = 1], [x = 4, y = 1, z = - 3], [x = 1, y = - 3, z = 4],

[x = 1, y = 4, z = - 3]]
```

The brackets indicate the solutions are stored in an array. To remove the first solution enter:

```
    (c5) d4[1];
    (d5)                              [x = - 3, y = 1, z = 4]
    (c6) d5[1];
    (d6)                                     x = - 3
    (c7) d5[2];
    (d7)                                      y = 1
    (c8) d5[3];
    (d8)                                      z = 4
```

To obtain parts of an equation or expression use the 'part' command.
For example, enter

```
    (c9) part(d6,1);
    (d9)                                        x
    (c10) part(d6,2);
    (d10)                                       - 3
```

This example is due to Paul Terwilliger

2.10 Example 10 - Pascal's triangle and recursive definitions

In this example we generate Pascal's Triangle.
Define the binomial coefficient recursively by

```
    (c1) c(n,k):= if (k = -1 or k= n+1) then 0 else if
    (k = 0 and n = 0) then 1 else c(n-1,k-1)+c(n-1,k);
    (d1) c(n, k) := IF k = - 1 OR k = n + 1 THEN 0
    ELSE (IF k = 0 AND n = 0 THEN 1 ELSE c(n - 1, k - 1) + c(n - 1, k))
```

We may now evaluate any entry in the Triangle, for example

```
    (c2) c(6,3);            /* c(6,3) refers to the fourth element of the seventh row */
    (d2)                           20
```

To print out the first 7 lines of the Triangle, enter

```
    (c3) for i : 0 thru 6 do (array(line,i),for j:0 thru i do line[j]:c(i,j),
    disp(listarray(line)));
                      /usr/mac309/maxsrc/array.o being loaded.
                                      [1]
                                    [1, 1]
                                  [1, 2, 1]
                                [1, 3, 3, 1]
                              [1, 4, 6, 4, 1]
                            [1, 5, 10, 10, 5, 1]
                          [1, 6, 15, 20, 15, 6, 1]
    (d3)                         done
```

As the variable i above ranged from 0 to 6, Macsyma ran through a procedure that printed out the ith line of the Triangle. First, an array 'line' was defined, with entries indexed by 0,1,...,i. Then as the variable j ranges over these indices, line[j] gets assigned the value c(i, j). After the line is filled its entries are displayed. This example is due to Paul Terwilliger

2.11 Example 11 - solving ODEs

This example shows how Macsyma handles ODEs
Let us first try a second order ODE

```
    (c1) diffeq: x^2*'diff(y,x,2)-3*x*'diff(y,x)+4*y = 0;
    (d1)
```

$$x^2\frac{d^2y}{dx^2} - 3x\frac{dy}{dx} + 4y = 0$$

```
    (c2) ode2(diffeq,y,x);
    Batching the file /usr/mac309/share/ode2.mac ...
    Batching done
    (d2)
```

$$y = x^2(\%k2log(x) - \%k1)$$

Now let us try a third order ODE

```
    (c3) diffeq: x^3*'diff(y,x,3)-3*x*'diff(y,x)+3*y = 0;
    (d3)
```

$$x^3\frac{d^3y}{dx^3} - 3x\frac{dy}{dx} + 3y = 0$$

```
    (c4) ode2(diffeq,y,x);
```

(e4)

$$x^3\frac{d^3y}{dx^3} - 3x\frac{dy}{dx} + 3y = 0$$

Not a proper differential equation

(d4) false

Now let us try a first order ODE

(c5) **x^2*'diff(y,x) + 3*x*y = sin(x)/x;**

(d5)

$$x^2\frac{dy}{dx} + 3xy = \frac{\sin(x)}{x}$$

(c6) **soln1:ode2(%,y,x);**

(d6)

$$y = \frac{\%c - \cos(x)}{x^3}$$

Now we give the initial conditions:
Let us require that for x=π, y = 0

(c7) **ic1(soln1,x=%pi,y=0);**

(d7)

$$y = -\frac{\cos(x) + 1}{x^3}$$

Here is another example:

(c8) ***'diff(y,x,2) + y*'diff(y,x)^3 = 0;***

(d8)

$$\frac{d^2y}{dx^2} + y(\frac{dy}{dx})^3 = 0$$

(c9) **soln2:ode2(%,y,x);**

(d9)

$$\frac{y^3 + 6\%k1y}{6} = x + \%k2$$

(c10) **ratsimp(ic2(soln2,x=0,y=0,'diff(y,x)=2));**

(d10)

$$-\frac{2y - 3y}{6} = x$$

(c11) **bc2(soln2,x=0,y=1,x=1,y=3);**

(d11)

$$\frac{y^3 - 10y}{6} = x - \frac{3}{2}$$

2.12 Example 12 - programming in Macsyma

This example shows how to use a file of commands to execute the commands from within macsyma. We will illustrate this by using the algorithm for testing if a polynomial is a *Schur polynomial.* A

Schur polynomial is one which has all of its roots inside the unit circle. An algorithm to test whether or not a polynomial is a *Schur polynomial* is as follows. Given a polynomial $\phi_n(z)$ of degree n,

$$\phi_n(z) = \sum_{i=0}^{n} a_i z^i,$$

we define $\tilde{\phi}_n(z)$ as

$$\tilde{\phi}_n(z) = \sum_{i=0}^{n} \bar{a}_{n-i} z^i,$$

where the bar on the coefficients denotes the complex conjugate. The polynomial $\phi_{n-1}(z)$ is defined by

$$\phi_{n-1}(z) = \frac{\tilde{\phi}_n(0)\phi_n(z) - \phi_n(0)\tilde{\phi}_n(z)}{z}.$$

The algorithm is based on the fact that $\phi_n(z)$ is a Schur polynomial if and only if $|\phi_n(0)|$ is less than $|\tilde{\phi}_n(0)|$ and $\phi_{n-1}(z)$ is a Schur polynomial.

The following macsyma instructions have been placed in a file called *schur*. Comments are given between the delimiters /* and */.

```
    /*  Code for testing for Schur polynomials
        the polynomial must be p
        the value of  n must be the degree of p
        If all the parameters t[j] are nonnegative then
        p is a Schur polynomial */
   /*  Run by using batch(schur) */
     schur:  true;              /* Initialize schur to true */
     while n > 0 and schur = true do (
        display(n),
        for i from 0 thru n do aa[i]:ratcoef(p,z,i),
   /*        Compute the check on the product of roots.  */
        x0:     realpart(aa[0]),
        y0:     imagpart(aa[0]),
        xn:     realpart(aa[n]),
        yn:     imagpart(aa[n]),
        t[n]: xn^ 2 + yn^ 2 -x0^ 2 -y0^ 2,
        ldisplay(t[n]),
        if t[n] <= 0 then schur: false ,
        r:      sum(aa[n-i]*z^ i,i,0,n),    /*  r is the "reverse" polynomial  of p */
        q:      subst(-%i,%i,r),            /*  q is the tilde of p */
        bb[0]: ratcoef(q,z,0),
        r:      bb[0]*p - aa[0]*q,
        r:      ratsimp(r),
        p:      ratsimp(r/z),
        ldisplay(p),
        n:      n-1
        );                              /*  end of while loop */
     ldisplay(schur);
```

This set of commands illustrates the use of the 'while loop' and the 'for loops.' Note that the commands within the loops end with a comma.

As an example of how to use this file we consider the three polynomials

$$(7 + 2i)z^2 - (8 - i)z + 1,$$

$$22z^3 - 20z^2 - 3z + 1,$$

and

$$(22 + i)z^3 - (20 - i)z^2 - 3z + 1.$$

We enter the polynomial and its degree and then call the file as follows.

```
(c1) p: (7+2*%i)*z^2 -(8-%i)*z +1;
(c2) n: 2$
(c3) batch(schur);
```

The use of the $ at the end of (c2) supresses the display of the value of n on the screen. The command in (c3) causes the file *schur* to be read and the executed. The file is also listed before it is executed. The final value of the logical variable schur will be displayed. (In this case it is true.) Next we enter the second polynomial.

```
(c11) p: 22*z^3 -20*z^2 -3*z +1;
(c12) n: 3$
(c13) batch(schur);
```

Again the commands in the file will be executed, and in this case the polynomial is not a Schur polynomial. (1 is a root.) Finally, we enter the third polynomial.

```
(c21) p: (22 +%i)*z^3 -(20-%i)*z^2 -3*z +1;
(c22) n: 3$
(c23) batch(schur);
```

Again the commands in the file will be executed, and in this case the polynomial is a Schur polynomial.

Chapter 3

Intricate Example

The Painlevé Property
Ref: J. Weiss, J. Math Phys., **24**(6), June 1983, 1405

3.1 Introduction

We say that a partial differential equation has the Painlevé property when the solutions of the pde are "single-valued" about the movable, singularity manifold. To be precise, if the singularity manifold is determined by

$$g(z_1, \dots, z_n) = 0 \tag{3.1}$$

and $f = f(z_1, \dots, z_n)$ is a solution of the partial differential equation, then we assume that

$$f = g^a \sum_{j=0}^{\infty} u_j g^j, \tag{3.2}$$

where

$$g = g(z_1, \dots, z_n), \qquad u_j = u_j(z_1, \dots, z_n), \qquad u_0 \neq 0.$$

are analytic functions of (z_j) in a neighborhood of the manifold (3.1) and a is an integer. Substitution of (3.2) into the partial differential equation determines the value(s) of a and defines the recursion relations for u_j, $j = 0, 1, 2, \dots$. When the ansatz (3.2) is correct, the pde is said to possess the Painlevé property and is conjectured to be integrable.

3.2 The Korteweg-de Vries Equation

The KdV equation

$$f_t + f f_x + b f_{xxx} = 0, \qquad b \in \mathbb{R} \tag{3.3}$$

possesses the Painlevé property. The expansion about the singular manifold has the form

$$f = g^{-2} \sum_{j=0}^{\infty} u_j g^j. \tag{3.4}$$

It is found that the "resonances" occur at

$$j = -1, 4, 6. \tag{3.5}$$

Resonances are those values of j at which it is possible to introduce arbitrary functions into the expansion (3.4). For each nontrivial resonance there occurs a compatibility condition that must be satisfied if the solution is to have a single-valued expansion. The resonance at $j = -1$ corresponds to the "arbitrary" function g defining the singular manifold (3.3). The compatability conditions at $j = 4$ and 6 are satisfied identically.

From the recursion relations, we find:

$$j = 0: \quad u_0 = -12bg_x^2; \tag{3.6}$$

$$j = 1: \quad u_1 = 12bg_{xx}; \tag{3.7}$$

$$j = 2: \quad g_x g_t + g_x^2 u_2 + 4bg_x g_{xxx} - 3bg_{xx}^2 = 0; \tag{3.8}$$

$$j = 3: \quad g_{xt} + g_{xx}u_2 - g_x^2 u_3 + bg_{xxxx} = 0; \tag{3.9}$$

$$j = 4: \quad \text{compatibility condition} : \frac{\partial}{\partial x}(g_{xt} + bg_{xxxx} + g_{xx}u_2 - g_x^2 u_3) = 0. \tag{3.10}$$

By (3.9) the compatibility condition (3.10) is satisfied identically. The compatibility condition at $j = 6$ is also satisfied identically and the KdV equation is thus shown to be Painlevé.

3.3 Macsyma calculations

We want to carry out the Painlevé test for the Korteweg-de Vries equation. Here is our Macsyma session. We have used **$** to suppress long expressions in the paper. The user should use a semicolon ; instead to see the results.

(c1) **writefile(painlevetest)$**

Let us first declare the dependencies of some functions. We will use

(c2) **depends([f,g,pd1t,pd1x,pd3x,kdv,rec0,rec1,rec2,rec3,rec4],[t,x])$**

We define f to be the following finite sum

(c3) **f:g**-2*'sum(u[j](t,x)*g**j, j, 0, n);**

(d3)

$$\frac{1}{g^2}\sum_{j=0}^{n} g^j u_j(t,x)$$

Let us calculate the following partial derivatives of this expression

(c4) **pd1t:diff(f,t,1)$**

(c5) **pd1x:diff(f,x,1)$**

(c6) **pd3x:diff(f,x,3)$**

Now we add these terms in accordance with the left-hand side of the KdV equation

(c7) **kdv : pd1t + f*pd1x + b*pd3x$**

The nonlinear term involves the product f*(df/dx). We want all products of sums to be converted in nested double sums. therefore we set

(c8) **sumexpand:true$**

(c9) **gfactor(kdv)$**

Note that the computer returns nested sums as requested.

There is a common denominator, g^5, which we remove by

(c10) **% * g**5 $**

Let us put factors, such as powers of g, inside the summations by setting

(c11) **intosum(%)$**

We have to choose a fixed value for n, for our purpose $n = 6$ suffices

```
(c12) n:6 $
(c13) ev(d11,sum)$
```
We calculate the coefficient of the term in g^0, and factor the result
```
(c14) factor(ratcoef(d13,g,0));
(d14)
```
$$-2\frac{dg}{dx}u_0(t,x)(u_0(t,x)+12b(\frac{dg}{dx})^2)$$
```
(c15) cf:inpart(%,[1,2,3]);
(d15)
```
$$-2\frac{dg}{dx}u_0(t,x)$$
```
(c16) rec0:d14/cf;
(d16)
```
$$u_0(t,x)+12b(\frac{dg}{dx})^2$$
We solve for u_0
```
(c17) (-1)*part(%,2)$
(c18) u[0](t,x) := d17$
(c19) u[0](t,x) ;
(d19)
```
$$-12b(\frac{dg}{dx})^2$$
Calculating the coefficent of g^1, substituting for u_0 and factoring gives
```
(c20) factor(ev(ratcoef(d13,g,1),diff));
(d20)
```
$$30b(\frac{dg}{dx})^3(u_1(t,x)-12b\frac{d^2g}{dx^2})$$
```
(c21) cf : inpart(%,[1,2,3])$
(c22) rec1 : d20/cf$
(c23) (-1)*part(%,2)$
(c24) u[1](t,x) := d23$
(c25) u[1](t,x);
(d28)
```
$$12b\frac{d^2g}{dx^2}$$
Calculating the coefficient of g^2, substituting u_0 and u_1 and factoring yields
```
(c26) factor(ev(ratcoef(d13,g,2),diff))$
(c27) cf : inpart(%,[1,2,3])$
(c28) rec2 : d26/cf$
```
From this we construct u_2 by taking parts, etc.
```
(c29) inpart(%,allbut(4))$
(c30) (-1)*(%)/diff(g,x,1)**2 $
```

```
(c31) u[2](t,x) := d30$
(c32) u[2](t,x);
(d32)
```

$$\frac{-4b\frac{dg}{dx}\frac{d^3g}{dx^3}+3b(\frac{d^2g}{dx^2})^2-\frac{dg}{dt}\frac{dg}{dx}}{(\frac{dg}{dx})^2}$$

Calculating the coefficient of g^2, inserting u_0, u_1 and u_2 and factoring yields:

```
(c33) factor(ev(ratcoeff(d13,g,3),diff));
(d33)
```

$$12b((\frac{dg}{dx})^4 u_3(t,x)-b(\frac{dg}{dx})^2\frac{d^4g}{dx^4}+4b\frac{dg}{dx}\frac{d^2g}{dx^2}\frac{d^3g}{dx^3}-3b(\frac{d^2g}{dx^2})^3+\frac{dg}{dt}\frac{dg}{dx}\frac{d^2g}{dx^2}-\frac{d^2g}{dt\,dx}(\frac{dg}{dx})^2)/\frac{dg}{dx}$$

This expression is presumably equivalent to the recursion relation (3.9) for $j = 3$ in the introduction. L't us verify this by doing the following steps:

```
(c34) d33/(12*b*diff(g,x,1))$
(c35) expand((-1)*(%))$
(c36) combine(%)$
(c37) u[2](t,x)$
(c38) factor(part(d36,[3,4]))$
(c39) ratsubst(u2,d37,d38);
(d39)
```

$$\frac{d^2g}{dx^2}u2$$

```
(c40) rec3 : part(d36,[1,2,5]) + d39;
(d40)
```

$$\frac{d^2g}{dx^2}u2-(\frac{dg}{dx})^2 u_3(t,x)+b\frac{d^4g}{dx^4}+\frac{d^2g}{dt\,dx}$$

```
(c41) u2 : u[2](t,x)$
(c42) eqrec3 : factor(ev(d40))$
```

This is indeed the same as expressing (d33) after division by (-12b)

Let us construct u_3

```
(c43) (-1)*(%)*diff(g,x,1)**2$
(c44) inpart(%,allbut(6))$
```

Note: for the computer the first term in (d43) is labelled as the sixth one!

```
(c45) u[3](t,x) := (-1)*(d44)/diff(g,x,1)**4$
(c46) u[3](t,x);
(d46)
```

$$\frac{b(\frac{dg}{dx})^2\frac{d^4g}{dx^4}-4b\frac{dg}{dx}\frac{d^2g}{dx^2}\frac{d^3g}{dx^3}-3b(\frac{d^2g}{dx^2})^3-\frac{dg}{dt}\frac{dg}{dx}\frac{d^2g}{dx^2}-\frac{d^2g}{dt\,dx}(\frac{dg}{dx})^2}{(\frac{dg}{dx})^4}$$

Finally, we verify that the coefficient of g^4 is now identically zero, after all the substitutions and simplifications (compatibility condition)

```
(c47) eqrec4 : factor(ev(ratcoef(d13,g,4),diff));
(d47)                                   0
```

BINGO! This last calculation took MACSYMA about 6 minutes. One could go on with this procedure to calculate **rec5** and then test the compatibility for **rec6** (i.e. eqrec6 = 0) but this has not been carried out here. Here is a summary of the results so far:

```
(c48) rec0$
(c49) rec1$
(c50) rec2$
(c51) rec3$
(c52) eqrec3$
(c53) eqrec4$
(c54) u[0](t,x)$
(c55) u[1](t,x)$
(c56) u[2](t,x)$
(c57) u[3](t,x)$
(c58) save(painlevetestlisp,all)$
(c59) closefile()$
```