JACAL

Symbolic Mathematics System
Version 1c3, May 2013

Aubrey Jaffer

This manual is for JACAL (version 1c3, May 2013), an interactive symbolic mathematics system.

Table of Contents

1 Overview

JACAL is a symbolic mathematics system for the simplification and manipulation of equations and single and multiple valued algebraic expressions constructed of numbers, variables, radicals, and algebraic functions, differential, and holonomic functions. In addition, vectors and matrices of the above objects are included.

JACAL 1c3 was released May 2013. Current information about JACAL can be found on JACAL's *WWW* home page:

`http://people.csail.mit.edu/jaffer/JACAL`

JACAL, part of the GNU project, is free software, and you are welcome to redistribute it under certain conditions; See the file COPYING with this program or type `(terms)();` to JACAL for details.

For a list of the features that have changed since the last JACAL release, see the file '`ANNOUNCE`'. For a list of the features that have changed over time, see the file '`ChangeLog`'.

1.1 Authors and Bibliography

Aubrey Jaffer
Most of JACAL

Michael Thomas
Polynomial Factoring.

Jerry D. Hedden
Tensors.

The maintainer can be reached as '`agj@alum.mit.edu`'.

Bibliography

[ACP] Donald Ervin Knuth.
The Art of Computer Programming : Seminumerical Algorithms (Vol 2).
2nd Ed (1981) Addison-Wesley Pub Co; ISBN: 0-201-03822-6

[GCL] Keith O. Geddes, Stephen R. Czapor, George Labahn.
Algorithms for Computer Algebra.
(October 1992) Kluwer Academic Pub; ISBN: 0-7923-9259-0

[Siret] Y. Siret (Editor), E. Tournier, J. H. Davenport, F. Tournier.
Computer Algebra: Systems and Algorithms for Algebraic Computation
2nd edition (June 1993) Academic Press; ISBN: 0-122-04232-8

[R5RS] Richard Kelsey and William Clinger and Jonathan (Rees, editors)
Revised(5) Report on the Algorithmic Language Scheme,
Higher-Order and Symbolic Computation Volume 11, Number 1 (1998), pp. 7-105, or
ACM SIGPLAN Notices 33(9), September 1998.

[SLIB] Todd R. Eigenschink and Aubrey Jaffer.
SLIB; The Portable Scheme Library

1.2 Installation

The JACAL program is written in the Algorithmic Language *Scheme*. So you must obtain and install a Scheme implementation in order to run it. The installation procedures given here use the SCM Scheme implementation. If your system has a Scheme (or Guile) implementation installed, then the '`scm`' steps are unnecessary.

JACAL also requires the SLIB Portable Scheme library which is available from http://people.csail.mit.edu/jaffer/SLIB.

`x86_64` *GNU/Linux with Redhat Package Manager* (*rpm*) [System]

```
wget http://groups.csail.mit.edu/mac/ftpdir/scm/scm-5f1-1.x86_64.rpm
wget http://groups.csail.mit.edu/mac/ftpdir/scm/slib-3b4-1.noarch.rpm
wget http://groups.csail.mit.edu/mac/ftpdir/scm/jacal-1c3-1.noarch.rpm
rpm -U scm-5f1-1.x86_64.rpm slib-3b4-1.noarch.rpm jacal-1c3-1.noarch.rpm
rm scm-5f1-1.x86_64.rpm slib-3b4-1.noarch.rpm jacal-1c3-1.noarch.rpm
```

The command '`jacal`' will start an interactive session.

`Unix` [System]

`GNU/Linux` [System]

```
wget http://groups.csail.mit.edu/mac/ftpdir/scm/scm-5f1.zip
wget http://groups.csail.mit.edu/mac/ftpdir/scm/slib-3b4.zip
wget http://groups.csail.mit.edu/mac/ftpdir/scm/jacal-1c3.zip
unzip -ao scm-5f1.zip
unzip -ao slib-3b4.zip
unzip -ao jacal-1c3.zip
(cd slib; ./configure --prefix=/usr/local/; make install)
(cd scm; ./configure --prefix=/usr/local/; make scm; make install)
(cd jacal; ./configure --prefix=/usr/local/; make install)
rm scm-5f1.zip slib-3b4.zip jacal-1c3.zip
```

The command '`jacal`' will start an interactive session using ELK, Gambit, Gauche, Guile, Larceny, MIT-Scheme, MzScheme, Scheme48, SCM, or SISC. Type '`jacal --help`' for instructions.

`Apple` [System]

http://www.io.com/~cobblers/scm/ has downloads and utilities for installing SCM and SLIB on Macintosh computers.

`x86` *Microsoft* [System]

Download and run http://groups.csail.mit.edu/mac/ftpdir/scm/slib-3b4-1.exe,

http://groups.csail.mit.edu/mac/ftpdir/scm/scm-5f1-1.exe, and http://groups.csail.mit.edu/mac/ftpdir/scm/jacal-1c3-1.exe.

Compiling Jacal

For Scheme implementations with compilers, it is worthwhile to compile SLIB files, and the JACAL files '`types.scm`' and '`poly.scm`'.

1.3 Running Jacal

If you successfully executed one of the installations of the previous section, then typing '`jacal`' or clicking an icon will begin an interactive session.

To manually start jacal, start your Scheme implementation with SLIB. This may involve setting up that implementation's initialization file or LOADing a '`.init`' file from the '`slib`' directory. Then type:

```
(slib:load "/usr/local/lib/jacal/math")
```

where '`/usr/local/lib/jacal/`' is a path to the JACAL directory. JACAL should then print:

```
JACAL version 1c3, Copyright 1989-1999, 2002 Aubrey Jaffer
JACAL comes with ABSOLUTELY NO WARRANTY; for details type '(terms)'.
This is free software, and you are welcome to redistribute it
under certain conditions; type '(terms)' for details.
;;; Type (math) to begin.
```

Do as it says:

```
(math)
⇒
type qed; to return to scheme, type help; for help.
e0 :
```

And you are ready to try the commands described in the rest of the manual.

Demonstrating Jacal

There are several demonstration files in the '`jacal`' directory. To run, use the batch command Chapter 6 [Miscellaneous], page 45.

'`batch("demo");`'
: Demonstrates a variety of JACAL features.

'`batch("test.math");`'
: Tests each operator.

'`batch("rw.math");`'
: Demonstrates tensors and The Robertson-Walker Cosmology Model.

Recovery from Errors

As JACAL is a complicated program there are bugs which will occasionally cause the program to stop with some sort of error reported by the underlying Scheme system. In interactive implementations (such as SCM) you can usually continue your session by typing `(math)`. The expression which was input to JACAL just before the error will be lost but you should be able to otherwise continue with your session.

Stopping Jacal

The command `quit();` will end your JACAL session.

With non-interactive Scheme implementations the JACAL command `qed();` or typing the end-of-file character (C-Z on MS-DOS and VMS, C-D on others) will end your JACAL session.

The command `qed();` will return to the interactive Scheme session. Typing `(math)` will return to the JACAL session.

From the interactive Scheme session `(exit)` or possibly an end-of-file character will terminate the session.

1.4 Release Notes

With the standard input grammar, the precedence of '`-`' as a prefix behaves strangely. `a^-b*c` becomes `a^(-b*c)` while ⇒`a^b*c (a^b)*c`.

Using `divide` to divide a polynomial by an integer does not work.

The command `example` executes the example it gives. This can lead to unpredictable results if the variables and constants in the example have already been given values by the user.

The function `minor` should be modified to accept lists for *row* and *col*.

Resultant might be modified to compute the resultant of a system of polynomials with respect to a list of variables.

Conventions

Things that are labeled as Operators can occur in expressions output by Jacal. Things that are labeled as Commands act upon their arguments and do not generally occur in expressions output by Jacal. Things that are labeled as flags are `set` to control aspects of the Jacal environment.

The examples throughout this text were produced using `SCM`.

Jacal has several grammers it understands. The `standard` grammar is used in this manual. It is like simple `TeX` grammar and algol family computer languages.

Identifier names are case sensitive and can be any number of characters long.

Manifest

COPYING
: details the LACK OF WARRANTY for Jacal and the conditions for distributing Jacal.

HELP
: is online introduction to using Jacal.

ChangeLog
: documents changes to Jacal.

jacal
: is a unix (sh) script to start an interactive jacal session.

demo
: demonstrates batch file use. "batch(demo);" to use in jacal.

rw.math
: is a batch file of Robertson-Walker model of General Relativity.

test.math
: is a batch file which tests Jacal.

jacal.texi
: is documentation on how to use jacal in TeXinfo format.

DOC
: has files telling about how jacal works.

- algdenom — gives an algorithm for clearing radicals and other algebraic field extensions from denominators.
- grammar — explains how to create new grammars.
- history — gives a little history of jacal.
- lambda — explains mid-level data formats. From a Dr. Dobbs article.
- ratint.tex — article explaining jacal's eventual integration algorithm.

math.scm — is the file you load into scheme in order to run jacal.

toploads.scm
: contains comments describing the rest of the files.

modeinit.scm
: has initializations for modes in Jacal.

view.scm — is a program for viewing TeX expressions.

1.5 GNU Free Documentation License

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

2 Algebra

2.1 Algebraic Operators

`+` *augend addend* [Operator]

Addition of scalar quantities or componentwise addition of bunches is accomplished by means of the infix operator `+`. For example,

```
e2 : a:[[1, 3, 5], [2, 4, 7]];

    [1  3  5]
e2: [       ]
    [2  4  7]

e3 : b:[2, 4];

e3: [2, 4]

e4 : a + b;

    [3  5  7 ]
e4: [        ]
    [6  8  11]

e5 : 3 + 2;

e5: 5

e6 : c + b;

e6: [2 + c, 4 + c]

e7 : e1 + e5;

          2   2
e7: 5 + (8 a + 12 a ) b
```

`-` *minuend subtrahend* [Operator]

`-` *subtrahend* [Operator]

The symbol `-` is used to denote either the binary infix operator subtraction or the unary minus.

```
e1 : -[1,2,3];

e1: [-1, -2, -3]

e2 : 3-7;

e2: -4
```

`+/-` *minuend subtrahend* [Operator]
`-/+` *minuend subtrahend* [Operator]
`+/-` *augend* [Operator]
`-/+` *augend* [Operator]

Jacal allows the use of `+/-` and `-/+` as ambiguous signs (unary plus-or-minus, unary minus-or-plus) and as ambiguous infix operators (binary plus-or-minus, binary minus-or-plus). The value `+/-` is also represented by the constant `%sqrt1`, while `-/+` is represented by `-%sqrt1`.

```
e7 : u:+/-3;

e7: 3 %sqrt1

e8 : u^2;

e8: 9

e9 : +/-(u);

e9: 3

e10 : u-/+3;

e10: b-/+(3 %sqrt1, 3)
```

`*` *multiplicand1 multiplicand2* [Operator]

Multiplication of scalar expressions such as numbers, polynomials, rational functions and algebraic functions is denoted by the infix operator `*`. For example,

```
e1 : (2 + 3 * a) * 4 * a * b^2;

                   2   2
e1: (8 a + 12 a ) b
```

One can also use `*` as an infix operator on bunches. In that case, it operates componentwise, in an appropriate sense. If the bunches are square matrices, the operator `*` multiplies corresponding entries of the two factors. It does not perform matrix multiplication. To multiply matrices one instead uses the operator `.` (i.e., a period). More generally, any binary scalar operator other than `^` can be used on bunches and acts componentwise.

`/` *dividend divisor* [Operator]

The symbol for division in Jacal is `/`. For example, the value returned by `6 / 2` is `3`.

```
e3 : (x^2 - y^2) / (x - y);

e3: x + y
```

`^` *expression exponent* [Operator]

The infix operator `^` is used for exponentiation of scalar quantitites or for componentwise exponentiation of bunches. For example, `2^5` returns 32. Unlike the other scalar infix operators, one cannot use `^` for component-wise operations on bunches. Furthermore, one should not try to use `^` to raise a square matrix to a power. Instead, one should use `^^`.

```
e7 : (1+x)^4;

                        2       3    4
e7: 1 + 4 x + 6 x  + 4 x  + x
```

`=` *expression1 expression2* [Operator]

In Jacal, the equals sign `=` is *not* used for conditionals and it is *not* used for assignments. To assign one value to another, use either `:` or `:=`. The operator `=` merely returns a value of the form `0 = expression`. The value returned by `a = b`, for example is `0 = a - b`.

```
e6 : 1=2;

e6: 0 = -1
```

`||` *Z1 Z2* [Operator]

The infix operator `||` is from electrical engineering and represents the effective impedance of the parallel connection of components of impedances *Z1* and *Z2*:

```
e1 : Z1 || Z2;

      Z1 Z2
e1: -------
    Z1 + Z2
```

2.2 Algebraic Commands

`eliminate` [*eqn_1 eqn_2* ...] [*var_1 var_2* ...] [Command]

Here *eqn_i* is an equation for $i = 1 \ldots n$ and where *var_j* is a variable for $j = 1 \ldots m$. `eliminate` returns a list of equations obtained by eliminating the variables *var_1*, ..., *var_m* from the equations *eqn_1*, ..., *eqn_n*.

```
e39 : eliminate([x^2+y=0,x^3+y=0],[x]);

                    2
e39: 0 = - y - y

e40 : eliminate([x+y+z=3,x^2+y^2+z^2=3,x^3+y^3+z^3=3],[x,y]);

e40: 0 = 1 - z
```

suchthat *var eqn* [Command]

The equation *eqn* must contain an occurence of variable *var*. `suchthat` returns an expression for all complex values of *var* satisfying *eqn*. `suchthat` is useful for extracting an expression from an equation.

```
e0 : a*x+b*y+c = 0;

e0: 0 = c + a x + b y

e1 : suchthat(x, e0);

    - c - b y
e1: ---------
        a
```

suchthat *var exp* [Command]

If an expression rather than an equation is given to `suchthat`, it is as though the equation *exp*=0 was given.

```
e2 : suchthat(x, e0);

    - c - b y
e2: ---------
        a
```

| *var exp_or_eqn* [Operator]

An alternative infix notation is also available for `suchthat`.

When used in combination with the '`{ }`' notation for `or`, the set notation used by some textbooks results.

If *var* in *eqn* has multiple roots, a named *field extension* will be introduced to represent any one of those roots. When multiple values are returned, the result (in `disp2d` and `standard` grammars) is wrapped with '`{ }`'.

```
e3 : x | a*x^2 + b*x + c;

                              2
ext3: {:@ | 0 = c + b :@ + a :@ }
e3: ext3

e4 : e3 ^ 2;
```

```
        - c - b ext3
    e4: ------------
             a
```

`extrule` *extsym* [Command]

Returns the rule defining named field extension *extsym*.

```
    e5 : extrule(ext3);

                                  2
    e5: 0 = c + b ext3 + a ext3
```

`or` *expr_1 . . .* [Command]

`or` *eqn_1 . . .* [Command]

The function `or` takes as inputs one or more equations or values. If the inputs are equations, then `or` returns an equation which is equivalent to the assertion that at least one of the input equations holds. If the inputs to `or` are values instead of two equations, then the function `or` returns a multiple value. If the inputs to `or` consist of both equations and values, then `or` will return the multiple values.

```
    e1 : or(x=2,y=3);

    e1: 0 = -6 + 3 x + (2 - x) y

    e2 : or(2,3);

                                   2
    e2: {:@ | 0 = -6 + 5 :@ - :@ }

    e3 : e2^2;

                                     2
    e3: {:@ | 0 = -36 + 13 :@ - :@ }

    e4 : or(x=2,17);

    e4: 17
```

'`{eqn, ... }`' can be used as an alternate syntax for `or`:

```
    e5 : {+1, -1};

                              2
    e5: {:@ | 0 = -1 + :@ }
```

2.3 Rational Expression

`num` *expr* [Command]

The function `num` takes a rational expression as input and returns a numerator of the expression.

```
e25 : num((x^2+y^2)/(x^2-y^2));

        2    2
e25: - x  - y

e26 : num(7/4);

e26: 7

e27 : num(7/(4/3));

e27: 21
```

`denom` *rational-expression* [Operator]

The Jacal command `denom` is used to obtain the denominator of a rational expression.

```
e26 : denom(4/5);

e26: 5
```

`listofvars` *expr* [Command]

The command `listofvars` takes as input a rational expression and returns a list of the variables that occur in that expression.

```
e7 : listofvars(x^2+y^3);

e7: [x, y]

e8 : listofvars((x^2+y^3)/(2*x^7+y*x+z));

e8: [z, x, y]
```

`imagpart` *z* [Command]

Returns the coefficient of `%i` in expression *z*;

`realpart` *z* [Command]

Returns all but the coefficient of `%i` in expression *z*;

`abs` *z* [Command]

`cabs` *z* [Command]

`| z |`

Returns the square root of the sum of the squares of the `realpart` and the `imagpart` of *z*.

2.4 Polynomials

`degree` *poly var* [Operator]

Returns the degree of polynomial or equation *poly* in variable *var*.

degree *poly* [Operator]

Returns the total-degree, the degree of its highest degree monomial, of polynomial or equation *poly*.

```
e26 : degree(a*x*x + b*y*x + c*y*y + d*x + e*y + f, y);

e26: 2

e27 : degree(a*x*x + b*y*x + c*y*y + d*x + e*y + f);

e27: 3
```

coeff *poly var* [Operator]
coeff *poly var deg* [Operator]
coeffs *poly var* [Operator]

The command `coeff` is used to determine the coefficient of a certain power of a variable in a given polynomial. Here *poly* is a polynomial and *var* is a variable. If the optional third argument is omitted, then Jacal returns the coefficient of the variable *var* in *poly*. Otherwise it returns the coefficient of *var*^*deg* in *poly*. The function `coeffs` returns a list of all of the coefficients. For example,

```
e14 : coeff((x + 2)^4, x, 3);

e14: 8

e15 : (x + 2)^4;

                         2      3    4
e15: 16 + 32 x + 24 x  + 8 x  + x

e16 : coeff((x + 2)^4, x);

e16: 32

e18 : coeffs((x + 2)^4, x);

e18: [16, 32, 24, 8, 1]
```

poly *var vect* [Operator]
poly *var coeff1 . . .* [Operator]

The function `poly` provides an inverse to the function `coeffs`, allowing one to recover a polynomial from its vector or list of coefficients.

```
e15 : poly(y, [16, 32, 24, 8, 1]);

                           2      3    4
e15: 16 + 32 y + 24 y  + 8 y  + y

e16 : poly(y, 16, 32, 24, 8, 1);

                           2      3    4
e16: 16 + 32 y + 24 y  + 8 y  + y
```

`poly` *eqn* [Operator]

The function `poly` returns the expression equal to 0 in equation *eqn*. Be aware that the sign and scaling of the returned polynomial will not necessarily match those in the equation creating *eqn*.

```
e17 : 2*a = 4*c;

e17: 0 = - a + 2 c

e18 : poly(e17);

e18: - a + 2 c
```

`content` *poly var* [Operator]

Returns a list of content and primitive part of a polynomial with respect to the variable. The content is the GCD of the coefficients of the polynomial in the variable. The primitive part is *poly* divided by the content.

```
e24 : content(2*x*y+4*x^2*y^2,y);

                           2
e24: [2 x, y + 2 x y ]
```

`divide` *dividend divisor var* [Operator]
`divide` *dividend divisor* [Operator]

The command `divide` treats *divident* and *divisor* as polynomials in the variable *var* and returns a pair '`[quotient, remainder]`' such that `dividend = divisor * quotient + remainder`. If the third argument *var* is omitted Jacal will choose a variable on its own with respect to which it will do the division. In particular, of *dividend* and *divisor* are both numerical, one can safely omit the third argument.

```
e5 : divide(x^2+y^2,x-7*y^2,x);

                 2    2       4
e5: [x + 7 y , y  + 49 y ]

e6 : divide(-7,3);

e6: [-2, -1]

e11 : divide(x^2+y^2+z^2,x+y+z);

                        2              2
e11: [- x - y + z, 2 x  + 2 x y + 2 y ]

e14 : divide(x^2+y^2+z^2,x+y+z,y);

                        2              2
e14: [- x + y - z, 2 x  + 2 x z + 2 z ]

e15 : divide(x^2+y^2+z^2,x+y+z,z);

                        2              2
e15: [- x - y + z, 2 x  + 2 x y + 2 y ]
```

`mod` *poly1 eqn var* [Command]
`mod` *poly1 poly2 var* [Command]
`mod` *poly1 poly2* [Command]

Returns *poly1* reduced with respect to *poly2* (or *eqn*) and *var*. If *poly2* is univariate, the third argument is not needed.

`mod` *poly1 n* [Command]

Returns *poly1* with all the coefficients taken modulo *n*. `mod` *poly1*

Returns *poly1* with all the coefficients taken modulo the current modulus.

If the modulus (*n* or the current modulus) is negative, then the results use symmetric representation.

```
e19 : x^4+4 mod 3;

          4
e19: 1 + x

e20 : x^4+4 mod x^2=2;

e20: 8

e22 : mod(x^3*a*7+x*8+34, -3);

                3
e22: 1 - x + a x

e23 : mod(5,2);

e23: 1

e24 : mod(x^4+4,x^2=2,x);

e24: 8
```

`gcd` *poly_1 poly_2* [Command]

The Jacal function `gcd` takes as arguments two polynomials with integer coefficients and returns a greatest common divisor of the two polynomials. This includes the case where the polynomials are integers.

```
e1 : gcd(x^4-y^4,x^6+y^6);

      2    2
e1: x   + y

e2 : gcd(4,10);

e2: 2
```

`discriminant` *poly var* [Command]

Here *poly* is a polynomial and *var* is a variable. This function returns the square of the product of the differences of the roots of the polynomial *poly* with respect to the variable *var*.

```
e7 : discriminant(x^3 - 1, x);

e7: -27
```

`resultant` *poly_1 poly_2 var* [Command]

The function `resultant` returns the resultant of the polynomials *poly_1* and *poly_2* with respect to the variable *var*.

```
e2 : resultant(x^2 + a, x^3 + a, x);

     2    3
e2: a  + a
```

`equatecoeffs` *z1 z2 var* [Command]

Returns the list of equations formed by equating each coefficient of variable *var*^n in *z1* to the corresponding coefficient of *var*^n in *z2*. *z1* and *z2* can be polynomials or ratios of polynomials.

2.5 Interpolation

`interp` *mat* [Command]

`interp` *vec1 vec2 . . .* [Command]

The only argument, *mat*, must be an array having at least one row of two expressions: [[*x1*,*y1*],[*x2*,*y2*],. . .]. It is an error if there are any duplicates in the first column of the second argument,

`interp` returns a polynomial function *poly*(`@`*1*) such that *mat*[1,2]=*poly*(*mat*[1,1]), *mat*[2,2]=*poly*(*mat*[2,1]), etc.

There is a variant of the `interp` command that takes multiple vector arguments instead of a matrix. These vectors represent points to be interpolated over. The same constraints apply as in the matrix version. All the variants of the interpolation procedure described later have both these forms.

```
e9 : interp([[2, 3], [0, -1]]);

e9 : lambda([@1], -1 + 2 @1)

e10 : interp([[2, 3], [1, z]]);

e10 : lambda([@1], -3 + 2 z + (3 - z) @1)

e11 : interp([2, 3], [y, z]);

                   3 y - 2 z + (-3 + z) @1
e11 : lambda([@1], -----------------------)
                           -2 + y
```

`interp.lagrange` *mat* [Command]

`interp.lagrange` *vec1 vec2 . . .* [Command]

This is the same as the `interp` command.

`interp.newton` *mat* [Command]

`interp.newton` *vec1 vec2 . . .* [Command]

This is similar to `interp` command with an added option of including derivative values when defining points. The same constraints apply as in `interp`. You can choose to specify some number of derivatives for each point. That number does not have to be the same for all points.

```
e0 : interp.newton([-1, 0], [0, 1], [1, 0]);

                             2
e0: lambda([@1], 1 - @1 )

e1 : interp.newton([-1, 0], [0, 1, 0, 20], [1, 0]);

                                2        4
e1: lambda([@1], 1 + 10 @1  - 11 @1 )

e2 : interp.newton([-1, 0], [0, 1, 0, a], [1, 0]);

                               2              4
                 2 + a @1  + (-2 - a) @1
e2: lambda([@1], ------------------------)
                                  2
```

interp.neville *mat* [Command]

interp.neville *vec1 vec2 . . .* [Command]

The same as `interp` in its functionality, but uses newtons form when constructing the polynomial.

2.6 Factoring

factor *int* [Command]

The Jacal command `factor` takes as input an integer and returns a list of the prime numbers that divide it, each occurring with the appropriate multiplicity in the list. If the number is negative, the list will begin with -1.

The results of the `factor` command are shown in a special *factored* format, which appears as the product of the factors.

```
e0 : factor(120);

     3
e0: 2  3 5

e1 : factor(-120);

        3
e1: -1 2  3 5
```

factor *polyratio* [Command]

Given a univariate ratio of polynomials *polyratio*, returns a matrix of factors and exponents.

As above, the results are shown in factored form.

```
e2 : factor((14*x^4-10/68*x^-5)/(5*x^2+1));

                9
     -5 + 476 x
e2: ------------------
                2   5
    2 17 (1 + 5 x ) x

e3 : (14*x^4-10/68*x^-5)/(5*x^2+1);

              9
   -5 + 476 x
e3: --------------
        5        7
    34 x  + 170 x

e4 : (476*x^9-5)/(34*(5*x^2+1)*x^5);

              9
   -5 + 476 x
e4: --------------
        5        7
    34 x  + 170 x

e5 : factor(x*y);

e5: y x

e6 : factor((x+a)*(y^4-z));

                  4
e6: -1 (a + x) (- y  + z)

e7 : factor((x+u*a^3)*(y^4-z));

         3            4
e7: -1 (a  u + x) (- y  + z)

e8 : factor((x+u*a^3)^2*(y^4-z)/((x+1)*(u^2-v^2)));

         4        3       2
     (- y  + z) (a  u + x)
e8: -------------------------
    (1 + x) (- u + v) (u + v)

e9 : factor(200*(-1*x+1+y)*(u-r^6)*(21*x+2-t^4));
```

```
     3  2     6                        4
e9: 2  5  (- r  + u) (1 - x + y) (2 - t  + 21 x)

e10 : factor(2*(a+u)*(-v+b)*(a*x+y)^2);

                                     2
e10: -1 2 (a + u) (- b + v) (a x + y)

e11 : factor(2*(a+u)*(-v+b)*(a*x+y)^2/((u^2-v^2)*(11*x+55)));

                                  2
     2 (a + u) (- b + v) (a x + y)
e11: ------------------------------
      11 (5 + x) (- u + v) (u + v)

e12 : factor(2*(a+u)*(-v+b)*(a*x+y)^2/((u^2-v^2)*x^4*(11*x+55)));

                                  2
     2 (a + u) (- b + v) (a x + y)
e12: -------------------------------
                                   4
     11 (5 + x) (- u + v) (u + v) x

e13 : factor((c^3*u+b*a)*(b*b*a+v*p^2*q^2*c));

             3        2      2  2
e13: (a b + c  u) (a b  + c p  q  v)

e14 : factor((2*z+y-x)*(y^3-a*x^2)*(b*z^2+y));

                             2        2    3
e14: (- x + y + 2 z) (y + b z ) (- a x  + y )

e15 : factor((a*a*b*z+d)*(2*a*b*b*z+c));

           2                2
e15: (d + a  b z) (c + 2 a b  z)

e16 : factor((a*a*b*z+d)*(2*a*b*b*z+c)*((u+a)*x+1));

                           2                2
e16: (1 + (a + u) x) (d + a  b z) (c + 2 a b  z)

e17 : factor((c*z+a)*(a*z+b)*(b*z+c));

e17: (b + a z) (c + b z) (a + c z)
```

```
e18 : factor((a*a*b*(x+w)*z+d)*(2*a*b*b*z+c));

               2        2                  2
e18: (d + (a  b w + a  b x) z) (c + 2 a b  z)

e19 : factor(((x+w)^2*z-u*d)*(-2*a*b*z+c));

                                    2            2
e19: -1 (- c + 2 a b z) (- d u + (w  + 2 w x + x ) z)

e20 : factor((-200*%i*x-c)*(x-d-z^5)/(a*(b^3-(a+u)*z)));

                                  5
     -1 (c + 200 %i x) (d - x + z )
e20: -----------------------------
                3
          a (- b  + (a + u) z)
```

The rest of this section documents commands from the factoring package. To use this package, execute the following command from the JACAL prompt:

```
require("ff");
```

Several of these commands return a matrix. The first column contains the factors and the second column contains the corresponding exponent.

`sff` *poly* [Command]

Given a primitive univariate polynomial *poly*, calculate the square free factorisation of *poly*. A *primitive* polynomial is one with no factors (other than units) common to all its coefficients.

`ffsff` *poly p* [Command]

`ffsff` *poly p m* [Command]

Given a monic polynomial *poly*, a prime *p*, and a positive integer *m*, calculate the square free factorisation of *poly* in GF(p^m)[x]. If *m* is not supplied, `1` is assumed.

```
e0 : ffsff(x^5+x^3+1, 53);

    [                2    3   ]
    [16 - 22 x + 26 x  + x   1]
e0: [                         ]
    [       -13 + x          2]
```

`berl` *poly n* [Command]

Given a square-free univariate polynomial *poly* and an integer power of a prime, *q*, returns (as a bunch) the irreducible factors of *poly*.

```
e2 : berl(x^5+x^3+2, 53);

                          2                2
e2: [1 + x, 5 - 26 x + x , 11 + 25 x + x ]
```

`parfrac` *polyratio* [Command]

Returns the partial fraction expansion of a rational univariate polynomial *polyratio*. The denominator of *polyratio* must be square free. This code is still being developed.

3 Calculus

3.1 Differential Operator

`differential` *expr* [Operator]
`'` *expr* [Operator]

The Jacal command `differential` computes the derivative of the expression *expr* with respect to a generic derivation. It is generic in the sense that nothing is assumed about its effect on the individual variables. The derivation is denoted by a right quote.

```
e6 : differential(x^2+y^3);

                     2
e6: 2 x x' + 3 y  y'

e7 : (x^2+y^3)';

                     2
e7: 2 x x' + 3 y  y'
```

3.2 Derivative Commands

`diff` *expr var1 . . .* [Command]

The Jacal command `diff` computes the derivative of the expression *expr* with respect to *var1*, ...

```
e6 : diff(x^2+y^3,y);

            2
e6: 3 y
```

`partial` *expr var1 . . .* [Command]

The Jacal command `partial` computes the partial derivative of the expression *expr* with respect to *var1*, ...

```
e6 : partial(x^2+@1^3,1);

             2
e6: 3 @1
```

`PolyDiff` *poly var1 . . .* [Command]

The Jacal command `PolyDiff` computes the derivative of the expression *poly* with respect to *var1*, ... It is faster than `diff` but *poly* must be a polynomial.

4 Matrices and Tensors

In JACAL, a matrix is just a `bunch` of equal length `bunch`s, and this is the structure that the matrix operations currently supported by JACAL (ncmult(), `^^`, transpose(), etc.) expect.

4.1 Generating Matrices

`bunch` *elt_1 elt_2 . . .* [Operator]

`[`*`elt_1`*`, `*`elt_2`*`, ...]`

To collect any number of Jacal objects into a bunch, simply enclose them in square brackets. For example, to make the bunch whose elements are 1, 2, 4, type `[1, 2, 4]`. One can also nest bunches, for example, `[1, [[1, 3], [2, 5]], [1, 4]]`. Note however that the bunch whose only element is `[1, 2, 3]` is `[1 2 3]`. It is importance to notice that one has commas and the other doesn't.

```
e3 : a:bunch(1, 2, 3);

e3: [1, 2, 3]

e4 : b:[a];

e4: [1  2  3]

e5 : c:[b];

e5: [[1, 2, 3]]

e6 : [[[1, 2, 3]]];

e6: [[1, 2, 3]]
```

`flatten` *bnch* [Operator]

Removes bunch nesting from *bnch*, returning a single bunch of the constituent expressions and equations.

```
e0 : flatten([a, [b, [c, d]], [5]]);

e0: [a, b, c, d, 5]
```

`ident` *n* [Command]

The command `ident` takes as argument a positive integer *n* and returns an *n*x*n* identity matrix. This is sometimes more convenient than obtaining this same matrix using the command `scalarmatrix`.

```
e6 : ident(4);

    [1  0  0  0]
    [          ]
    [0  1  0  0]
e6: [          ]
    [0  0  1  0]
    [          ]
    [0  0  0  1]
```

`scalarmatrix` *size entry* [Command]

The command `scalarmatrix` takes as inputs a positive integer *size* and an algebraic expression *entry* and returns an `n * n` diagonal matrix whose diagonal entries are all equal to *entry*, where `n = size`.

```
e1 : scalarmatrix(3, 6);

    [6  0  0]
    [       ]
e1: [0  6  0]
    [       ]
    [0  0  6]
```

`diagmatrix` *list* [Command]

The Jacal command `diagmatrix` takes as input a list of objects and returns the diagonal matrix having those objects as diagonal entries. In case one wants all of the diagonal entries to be equal, it is more convenient to use the command `scalarmatrix`.

```
e3 : diagmatrix(12,3,a,s^2);

    [12  0  0  0 ]
    [            ]
    [0   3  0  0 ]
e3: [            ]
    [0   0  a  0 ]
    [            ]
    [0   0  0   2]
    [          s ]

e4 : diagmatrix([1,2],2);

    [[1, 2]  0]
e4: [         ]
    [  0     2]
```

`sylvester` *poly_1 poly_2 var* [Command]

Here, *poly_1* and *poly_2* are polynomials and *var* is a variable. The function `sylvester` returns the matrix introduced by Sylvester (*A Method of Determining By Mere Inspection the Derivatives from Two Equations of Any Degree, Phil.Mag.*

`16` *(1840) pp. 132-135, Mathematical Papers, vol. I, pp. 54-57*) for computing the resultant of the two polynomials *poly_1* and *poly_2* with respect to the variable *var*. If one wants to compute the resultant itself, one can simply use the command `resultant` with the same syntax.

```
e5 : sylvester(a0 + a1*x + a2*x^2 + a3*x^3, b0 + b1*x + b2*x^2, x);

    [a3  a2  a1  a0  0 ]
    [                  ]
    [0   a3  a2  a1  a0]
    [                  ]
e5: [b2  b1  b0  0   0 ]
    [                  ]
    [0   b2  b1  b0  0 ]
    [                  ]
    [0   0   b2  b1  b0]
```

`genmatrix` *function rows cols* [Command]

The function `genmatrix` takes as arguments a *function* of two variables and two positive integers, *rows* and *cols*. It returns a matrix with the indicated numbers of rows and columns in which the \$(i,j)\$th entry is obtained by evaluating *function* at \$(i,j)\$. The function may be defined in any of the ways available in Jacal, i.e previously by an explicit algebraic definition, by an explicit lambda expression or by an implicit lambda expression.

```
e4 : @1^2+@2^2;

                         2      2
e4: lambda([@1, @2], @1   + @2 )

e5 : genmatrix(e4,3,5);

    [2   5   10  17  26]
    [                  ]
e5: [5   8   13  20  29]
    [                  ]
    [10  13  18  25  34]
```

4.2 Matrix Parts

`rank` *matrix* [Command]

The rank of *matrix* is the maximal number of linearly independent columns of *matrix*, which is always equalt to the maximal number of linearly independent rows of *matrix*.

```
e13 : rank([[0,0],[0,0]]);

e13: 0

e14 : rank([[0,0],[0,1]]);

e14: 1

e15 : rank([[2,0],[0,1]]);

e15: 2

e17 : rank([[b,c],[0,a]]);

e17: 2

e18 : rank([[b,c,d],[a,0,a],[e,f,a]]);

e18: 3
```

`row` *matrix i* [Command]

The command `row` returns the *i*th row of the matrix *matrix*, where `i = int`. If *int* is larger than the number of rows of *matrix*, then Jacal prints an error message. The corresponding command for columns of a matrix is `col`.

```
e3 : u:[[1, 2, 3], [1, 5, 3]];

    [1  2  3]
e3: [       ]
    [1  5  3]

e4 : row(u, 2);

e4: [1, 5, 3]
```

`col` *matrix integer* [Command]

The command `col` is used to extract a column of a matrix. Here, *matrix* is a matrix and *integer* is a positive integer. If that integer exceeds the number of columns, an error message such as

```
ERROR: list-ref: Wrong type in arg1 ()
```

appears. Here is an example of correct use of the command `col`:

```
e19 : a:[[1,2,4],[2,5,6]];

          [1  2  4]
     e19: [       ]
          [2  5  6]

e20 : col(a,2);

          [2]
     e20: [ ]
          [5]
```

`minor` *matrix i j* [Command]

The command `minor` returns the submatrix of *matrix* obtained by deleting the *i*th row and the *j*th column.

```
e21 : b:[[1,2,3],[3,1,5],[5,2,7]];

          [1  2  3]
          [       ]
     e21: [3  1  5]
          [       ]
          [5  2  7]

e22 : minor(b,3,1);

          [2  3]
     e22: [    ]
          [1  5]
```

`cofactor` *matrix i j* [Command]

The command `cofactor` returns the determinant of the *i*, *j* `minor` of *matrix*.

`rapply` *bunch int_1 int_2 . . .* [Command]

The function `rapply` is used to access elements of bunches. It can also access elements nested at lower levels in a bunch. In particular, it can also access matrix elements. In the above syntax, *bunch* is the bunch whose parts one wishes to access, and *n*, *int_1*, *int_2*, . . ., *int_n* are positive integers. It returns the *int_n*-th element of the *int_{n-1}*-th element of . . . of the *int_2*-th element of the *int_1*-th element of *bunch*. One can have `n = 0`. In that case, `rapply` simply returns the bunch.

```
e2 : rapply([[1,2,3],[1,4,6],3],2,3);

e2: 6

e6 : rapply([a,b],2);

e6: b

e7 : rapply([a,b]);

e7: [a, b]
```

4.3 Matrix commands

`transpose` *matrix* [Command]

Computes the transpose of (*matrix*).

`determinant` *matrix* [Command]

The Jacal command `determinant` computes the determinant of a square matrix. Attempting to take the determinant of a non-square matrix will produce an error message.

```
e1 : a:[[1,2],[6,7]];

    [1  2]
e1: [    ]
    [6  7]

e2 : determinant(a);

e2: -5
```

`charpoly` *matrix var* [Command]

The characteristic polynomial of *matrix*:

determinant(*matrix* - I *var*)

`.` *matrix1 matrix2* [Command]

Matrix multiplication.

```
e1 : a:[[1, 2, 3], [5, 2, 7]];

    [1  2  3]
e1: [       ]
    [5  2  7]

e2 : b:[[3, 2], [6, 4]];

    [3  2]
e2: [    ]
    [6  4]

e3 : b . a;

    [13  10  23]
e3: [          ]
    [26  20  46]
```

^^ *matrix exponent* [Command]

The infix operator ^^ is used for raising a square matrix to an integral power.

```
e8 : a:[[1, 0], [-1, 1]];

    [1   0]
e8: [     ]
    [-1  1]

e9 : a^^3;

    [1   0]
e9: [     ]
    [-3  1]
```

Negative exponents raise the inverse matrix to a power.

```
e8 : [[a, b], [c, d]];

    [a  b]
e8: [    ]
    [c  d]

e9 : e8^^-1;

    [     d           - b    ]
    [-----------  -----------]
    [- b c + a d  - b c + a d]
    [                        ]
e9: [    - c           a     ]
    [-----------  -----------]
    [- b c + a d  - b c + a d]

e10 : e8^^-2;

     [              2                   - a b - b d       ]
     [       b c + d            -------------------------]
     [-------------------------  2  2                 2  2]
     [ 2  2                2  2  b  c  - 2 a b c d + a  d ]
     [b  c  - 2 a b c d + a  d                             ]
     [                                      2              ]
e10: [      - a c - c d                  a  + b c          ]
     [-------------------------  -------------------------]
     [ 2  2                2  2   2  2                 2  2]
     [b  c  - 2 a b c d + a  d   b  c  - 2 a b c d + a  d ]

e11 : e8 . e9;

     [1  0]
e11: [    ]
     [0  1]

e12 : e9 . e8;

     [1  0]
e12: [    ]
     [0  1]

e13 : e10 . e8 . e8;

     [1  0]
e13: [    ]
     [0  1]
```

`dotproduct` *vector_1 vector_2* [Command]

The Jacal function `dotproduct` returns the dot product of two row vectors of the same length. It will also give the dot product of two matrices of the same size by computing the sum of the dot products of the corresponding rows or, what is the same, the trace of one matrix times the transpose of the other one.

```
e28 : a:[1,2,3]; b:[3,1,5];

e28: [1, 2, 3]

e29 :
e29: [3, 1, 5]

e30 : dotproduct(a,b);

e30: 20
```

`crossproduct` *vector_1 vector_2* [Command]

The Jacal command `crossproduct` computes the cross product of two vectors. By definition, the two vectors must each have three components.

```
e24: [2 x, y + 2 x y ]

e25 : crossproduct([1,2,3],[4,2,5]);

e25: [4, 7, -6]
```

4.4 Tensors

The `tensors` supported by JACAL are an extension of the matrix structure (i.e., a bunch of bunches of bunches ...) with the added stipulation that all `dimensions` of the tensor be the same length (e.g., 4x4x4). The number of dimensions (indices) in a tensor is its rank: A scalar is a tensor of rank 0; a vector is a rank 1 tensor; a matrix has rank 2; and so on.

Further, just as matrix binary operations place restrictions on the matrices involved (e.g., the row/column length requirement for matrix multiplication), the tensor binary operations require that the dimensions of each tensor be of the same length. For example, you could not multiply a 3x3 tensor and a 4x4x4 tensor.

JACAL's tensors do not support the construct of contravariant and covariant indices. Users must keep track of this information themselves, and perform the necessary operations with an appropriate metric so that the "index gymnastics" is performed correctly.

Before using any of JACAL's tensor operations, execute the following command from the JACAL prompt:

require("tensor");

This loads the file '`tensor.scm`' into JACAL, and makes the tensor operations available for use.

JACAL currently supports four tensor operations: `tmult`, `contract`, `indexshift`, and `indexswap`. Each of these is described in detail below.

4.5 Tensor Multiplication

`tmult` *matrix_1 matrix_2 index_1 index_2* [Command]

`tmult` takes a minimum of two arguments which are the tensors on which the multiplication operation is to be performed.

With no additional arguments, `tmult` will produce the outer product of the two input tensors. The rank of the resulting tensor is the sum of the inputs' ranks, and the components of the result are formed from the pair-wise products of components of the inputs. For example, for the input tensors `x[a,b]` and `y[c]`

```
⇒z:tmult(x,y); z[a,b,c] = x[a,b]*y[c]
```

With an additional argument, `tmult` will produce the inner product of the two tensors on the specified index. For example, given `x[i,j]` and `y[k,l,m]`

```
z:tmult(x,y,3);
⇒
                  length
                  -----
                  \
      z[a,b,c] =   >   x[a,q] * y[b,c,q]
                  /
                  -----
                  q = 1
```

Note that in this case *x* only has 2 indices. All of JACAL's tensor operations modify index inputs to be between 1 and the rank of the tensor. Thus, in this example, the 3 is modified to 2 in the case of *x*. As another example, with `x[i,j,k]` and `y[l,m,n]`

```
z:tmult(x,y,2);
⇒
                    length
                    -----
                    \
      z[a,b,c,d] =   >   x[a,q,b] * y[c,q,d]
                    /
                    -----
                    q = 1
```

With four arguments, `tmult` produces an inner product of the two tensors on the specified indices. For example, for `x[i,j]` and `y[k,l,m]`

```
z:tmult(x,y,1,3);
⇒
                  length
                  -----
                  \
      z[a,b,c] =   >   x[q,a] * y[b,c,q]
                  /
                  -----
                  q = 1
```

Note that matrix multiplication is the special case of an inner product (of two "two dimensional matrices") on the second and first indices, respectively: `tmult(x,y,2,1)` $\equiv$ `ncmult(x,y)`

Finally, tmult handles the case of a scalar times a tensor, in which case each component of the tensor is multiplied by the scalar.

4.6 Tensor contraction

`contract` *matrix index1 . . .* [Command]

The contraction operation produces a tensor of rank two less than a given tensor. It does this by performing a summation over two of the indices of the given tensor, as clarified in the examples below.

`contract` takes at least one argument which is the tensor on which the contraction operation is to be performed. One or two additional arguments may be provided to specify the indices to be used in the summation. If no additional arguments are provided, the summation is performed over the first and second indices. With one additional argument, the summation is over the specified index and the one following it (e.g., if 3 is specified, the third and fourth indices are used). With two additional arguments, the summation is performed over the indices specified. The actual indices used will be constrained to be between 1 and the rank of the tensor.

Examples:

1) For a square matrix (tensor of rank 2), `contract` returns a scalar that is the sum of the diagonal elements of the matrix.

2) Given `x[i,j,k,l]`, the command

```
y:contract(x,2,4);
```

produces:

```
               length
               -----
               \
    y[a,b] =    >   x[a,q,b,q]
               /
               -----
               q = 1
```

Special cases: If `contract` is given a scalar (rank 0 tensor) as input, it just returns the scalar. For a vector (tensor of rank 1), `contract` returns a scalar that is the sum of the elements of the vector.

4.7 Shifting of Tensor Indices

`indexshift` *matrix index1 . . .* [Command]

`indexshift` rearranges the indices of a tensor. It is one of two generalizations of the matrix transpose operation (cf. `indexswap`).

`indexshift` takes at least one argument which is the tensor on which the index shifting is to be performed. One or two additional arguments may be provided to specify the index and the position to which it is to be shifted. If no additional arguments are provided, the first index of the tensor is shifted to the second position (equivalent the matrix transpose operation). If one additional argument is provided, it specifies the index to be shifted, and that index will be shifted "to the right" one position (e.g., if 3 is specified, the third index will be shifted to the forth position). If two additional arguments are provided, the first specifies the index and the second specifies the position to which it is to be shifted. The actual index shifted and its shifted position will be constrained to be between 1 and the rank of the tensor.

For example, given `x[a,b,c,d]`, the command `y:indexshift(x,1,3);` produces a tensor *y* such that `y[a,b,c,d]` ≡ `x[b,c,a,d]`. In this example, the element that was in position `[a,b,c,d]` in *x* will be in position `[b,c,a,d]` in *y*.

Special cases: If `indexshift` is given a scalar (rank 0 tensor) as input, it just returns the scalar. For a vector (tensor of rank 1), `indexshift` transposes the 1-by-n matrix (row vector) to an n-by-1 matrix (column vector).

4.8 Swapping of Tensor Indices

`indexswap` *tensor ...* [Command]

`indexswap` rearranges the indices of a tensor. It is one of two generalizations of the matrix transpose operation (cf. `indexshift`).

`indexswap` takes at least one argument which is the tensor on which index swapping is to be performed. One or two additional arguments may be provided to specify the indices to be swapped. If no additional arguments are provided, the first and second indices of the tensor are swapped (equivalent the matrix transpose operation). With one additional argument, the specified index is swapped with the one following it (e.g., if 2 is specified, the second and third indices will be swapped). If two additional arguments are provided, they specify the indices to be swapped. The actual indices used will be constrained to be between 1 and the rank of the tensor.

For example, given x[a,b,c,d], the command `y:indexswap(x,2,4);` produces a tensor *y* such that y[a,b,c,d] = x[a,d,c,b]. In this example, the element that was in position [a,b,c,d] in *x* will be in position [a,d,c,b] in *y*.

Special cases: If `indexswap` is given a scalar (rank 0 tensor) as input, it just returns the scalar. For a vector (tensor of rank 1), `indexswap` transposes the 1-by-n matrix (row vector) to an n-by-1 matrix (column vector).

5 Lambda Calculus

`lambda` *varlist expression* [Operator]

Jacal has the ability to work with lambda expressions, via the command `lambda`. Furthermore, Jacal always converts user definitions of functions by any method into lambda expressions and converts the dummy variables of the function definition into symbols such as *1*, *2*, ... Jacal can manipulate lambda expressions by manipulating their function parts, as in '`e14`' below. Jacal can also invert a function using the command `finv`.

```
e12 : lambda([x],x^2);

                         2
e12: lambda([@1], @1 )

e13 : lambda([x,y,z],x*y*z);

e13: lambda([@1, @2, @3], @1 @2 @3)

e14 : e12+e13;

                                 2
e14: lambda([@1, @2, @3], @1  + @1 @2 @3)
```

`elementwise` *function matrix1 matrix2 ...* [Command]

The arguments *matrix1*, *matrix2*, ... must have the same shape. The command `elementwise` returns a new matrix formed by applying *function* to each tuple of elements of *matrix1*, *matrix2*, ...

```
e9 : elementwise(foo,[a, b], [c, d]);

e9: [foo(a, c), foo(b, d)]

e10 : elementwise(@1+5*@2,[a, b], [c, d]);

e10: [a + 5 c, b + 5 d]

e1 : elementwise(@1-@2,[9,8,7],[[1,0],[4,5],[6,3]]);

    [ 8  9 ]
    [      ]
e1: [ 4  3 ]
    [      ]
    [ 1  4 ]
```

`finv` *function* [Command]

function`^^-1`

The command `finv` takes as input a function of one variable and returns the inverse of that function. The function may be defined in any of the ways permitted in Jacal, i.e. by an explicit algebraic definition, by an explicit lambda expression or by an implicit lamba expression. If f is the function, then typing `f^^-1` has the same effect as typing `finv(`*f*`)`.

```
e0 : w(t):=t+1;

w(t): lambda([@1], 1 + @1)

e0 : finv(w);

e0: lambda([@1], -1 + @1)
```

6 Miscellaneous

`%` [Command]

The symbol `%` represents the last expression obtained by Jacal. It can be used in formulas like any other constant or variable or expression.

```
e21: 5

e22 : %;

e22: 5

e23 : %^2;

e23: 25
```

`batch` *filename* [Command]

The command `batch` is used to read in a file containing programs written in Jacal. Here, *filename* is a string in double quotes. The precise way in which one refers to a file is, of course, system dependent.

```
batch("demo");
```

of the file demo in the JACAL directory will give a demonstration of JACAL's capabilities.

`tex` *expr* [Command]
`scheme` *expr* [Command]
`disp2d` *expr* [Command]
`standard` *expr* [Command]

Displays *expr* in TeX, Jacal's two-dimensional output format, or Jacal's infix input format, respectively.

`tex` *string* [Command]

Read TeX expression *string*.

`scheme` *string* [Command]

Read Scheme expression *string*.

`disp2d` *string* [Command]
`standard` *string* [Command]

Reads *string* in Jacal's infix input format.

```
e24 : b^2-4*a*c;

      2
e24: b  - 4 a c

e25 : tex(e24);
b^{2} - 4\,a\,c
e25 : tex("b^{2} - 4\,a\,c");

      2
e25: b  - 4 a c

e26 : disp2d(e25);
 2
b  - 4 a c
e26 : disp2d("b^2-4*a*c");

      2
e26: b  - 4 a c

e27 : scheme(e26);
(- (^ b 2) (* 4 a c))

e27 : scheme("(- (^ b 2) (* 4 a c))");

      2
e27: b  - 4 a c
```

`commands` [Command]

The command `commands` produces a list of all of the command available in Jacal. It is called as s function of no arguments.

e21 : commands();

```
% * + - / = ^ ^^ abs args augcoefmatrix b+/- b-/+ batch bunch cabs
cartprod chain charpoly coeff coeffs coefmatrix cofactor col commands
content continue crossproduct degree denom depends describe
determinant diagmatrix diff differential discriminant disp2d divide
dotproduct elementwise eliminate equatecoeffs example extrule factor
factorial factors finv flatten func gcd genmatrix help ident imagpart
interp interp.lagrange interp.neville interp.newton jacobi jacobian
listofvars load matrix minor mod ncmult negate num or over parallel
partial poly polydiff polyelim prime? qed quit rank rapply realpart
require restart resultant row scalarmatrix scheme set shadow show
standard sylvester system terms tex transcript transpose u+/- u-/+
verify wronski wronskian
```

describe *command* [Command]

The command `describe` is the heart of the online help facility of Jacal. Here, *command* is a string which is the name of a command and `describe` produces a brief description of the command and in many cases includes an example of its use. Together with the command `commands()`, which prints a list of all available Jacal commands, and the command `example`, which gives an example of the use of the command, one can in principle use Jacal without a manual after one has learned how to get started.

```
e27 : describe(col);
column.  column of a matrix
e27 : describe(resultant);
resultant.  The result of eliminating a variable between 2
equations (or polynomials).
27 : describe(+);
Addition, plus.
a + b
```

example *command* [Command]

Here, *command* is a string which is the name of a Jacal command. `example` gives an example of the use of the command. See also Chapter 6 [Miscellaneous], page 45.

```
e43 : example(+);
a + b

e43: a + b
```

load *string* [Command]

The Jacal command `load` takes as input a string and reads in a '`Scheme`' file whose name is obtained by appending the extension '`.scm`' to the string. If you want to read in a file of Jacal commands, do not use `load`. Instead use the command `batch`. To load in the file '`foo.scm`',

```
e9 : load("foo");

e9: foo
```

qed [Command]

Exit from Jacal to Scheme. With interactive Scheme systems (such as SCM), It does not return you to the operating system. Instead it suspends Jacal and returns you to the underlying scheme. You can return to the Jacal session where you left off by simply typing `(math)`. If you do not wish to return to Jacal but really want to terminate the session and return to the operating system, then after typing `qed();`, type `(slib:exit)` or use `quit`.

quit [Command]

Exit directly from Jacal to the operating system. You will not be able to continue your Jacal session.

```
type qed(); to return to scheme
e1 : qed();
scheme
> (math)
type qed(); to return to scheme
e2 : quit();
unix>
```

`system` *command* [Command]

One can issue commands to the operating system without leaving Jacal. To do this, one uses the command `system`. For example, in a UNIX operating system, the command `system("ls");` will print the directory. One way in which the command `system` might be especially useful is to edit files containing Jacal scripts without leaving Jacal, particularly in non-UNIX machines or on machines without GNU emacs.

```
e0 : system("echo hi there");
hi there

e0: 0
```

`terms` [Command]

Prints a copy of the GNU General Public License

```
e1 : terms();
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.
```

[rest deleted for brevity]

`transcript` *string* [Command]

The command `transcript` allows one to record a Jacal session. It is called with the syntax `transcript(`*string*`);`, where *string* is the name of the file in which one wants to keep the transcript of the session. When one wishes to stop recording, one types `transcript();`. One is then free to use `transcript` again later in the session on another file. One can use it on the same file, but the file is overwritten. Presently, the command `transcript` does not echo commands to a file.

```
e9 : a:[1,2,3];

e9: [1, 2, 3]

e10 : transcript("foo");

e10: foo

e11 : a;

e11: [1, 2, 3]

e12 : transcript();
e12 : system("cat foo");

e10: foo

e11 : a;

e11: [1, 2, 3]

e12 : transcript();

e12: 0
```

`set` *flag value* [Command]

There are various flags that the Jacal user can control, namely the Jacal command line prompt, the priority for printing terms in Jacal output, the input grammar and the output grammar. For a discussion of the various grammars please See Chapter 7 [Flags], page 51. The command `show` is closely related, allowing one to see what the current settings are.

`show` *flag* [Command]

The command `show` enables the Jacal user to examine the current setting of various flags as well as to list the flags that can be set by the user and to display other information. To change the settings of the flags, use the command `set`. To see all the information accessible through the `show` command, type `show all`. To see the available grammars, type `show grammars`. To see the current input grammar type `show ingrammar`. To see the current output grammar, type `show outgrammar`. To see the current priority for printing expressions, type `show priority`.

e1 : show all;

```
all debug echogrammar grammars horner ingrammar linkradicals outgrammar
page phases priority prompt trace version width
```

e1 : show prompt;

e1: e1

e3 : show priority;

```
:@ (differential :@) @3 @2 @1 y x wronskian wronski verify u-/+ u+/-
transpose transcript tex terms t system sylvester standard show shadow
set scheme scalarmatrix row resultant restart require realpart rapply
rank quit qed prompt priority prime? polyelim polydiff poly partial
parallel over or num negate ncmult mod minor matrix load listofvars
jacobian jacobi interp.newton interp.neville interp.lagrange interp
imagpart ident help genmatrix gcd func flatten finv factors factorial
factor extrule example equatecoeffs eliminate elementwise e0 dotproduct
divide disp2d discriminant differential diff diagmatrix determinant
describe depends denom degree crossproduct continue content commands col
cofactor coefmatrix coeffs coeff charpoly chain cartprod cabs c bunch
batch b-/+ b+/- b augcoefmatrix args all abs a ^^ ^ ? = ::@ / - + * %
%sqrt1 %i
```

e3 : show outgrammar;

e3: disp2d

e4 : show ingrammar;

e4: standard

e5 : show grammars;

e5: [scheme, null, schemepretty, standard, disp2d, tex]

7 Flags

`prompt` *string* [Flag]

If one changes the prompt, *string* is a string of alphanumeric characters without quotes. After this command is executed, subsequent commands will cause new prompts to be obtained from *string* by incrementing it. If the prompt ends in a letter, it will be treated as a digit in base 26 and incremented. If it ends in a string of digits, that string will be treated as a number in base 10 and incremented. The remaining characters in the string will play no role in this incrementation.

```
e1 : set prompt az9Z;
e1 : a+b;

az9Z: a + b

az9AA : a+b;

az9AA: a + b

az9AB : set prompt ok99;
az9AB : a+b;

ok99: a + b

ok100 : a+b;

ok100: a + b

ok101 :
```

`ingrammar` *grammar* [Flag]
`outgrammar` *grammar* [Flag]

The following examples show how one changes the input grammar or the output grammar.

```
e1 : a:[[[1,2,3]]];

e1: [[1, 2, 3]]

e2 : set outgrammar standard;
e2 : a;

e2: [[[1, 2, 3]]]

e3 : set outgrammar scheme;
e3 : a;

(define e3 #(#(#(1 2 3))))
e4 : (1+x)^5;

(define e4 (+ 1 (* 5 x) (* 10 (^ x 2)) (* 10 (^ x 3)) (* 5 (^ x 4)) (^ x 5)))
e6 : set ingrammar scheme;
e6 : (+ e4 1);

(define e6 (+ 2 (* 5 x) (* 10 (^ x 2)) (* 10 (^ x 3)) (* 5 (^ x 4)) (^ x 5)))
e7 : (set ingrammar disp2d)
e7 : diagmatrix(3,6);

(define e7 #(#(3 0) #(0 6)))
e8 : set outgrammar disp2d;
e8 : e7;

    [3  0]
e8: [    ]
    [0  6]

e9 : set outgrammar standard;
e9 : e7;

e9: [[3, 0], [0, 6]]
```

Note that in the above examples, it is possible to input and output expressions in scheme by setting the ingrammar and/or outgrammar to `scheme`. Doing so result in linear output (as with `standard grammar`) as opposed to a two dimensional display (as with `disp2d`). The analogue of `disp2d` for scheme output is scheme pretty-printing. To have such output, set the output grammar to `schemepretty`.

```
e4 : set outgrammar schemepretty;
e4 : (1+x)^5;

(define e4
  (+ 1
     (* 5 x)
     (* 10 (^ x 2))
     (* 10 (^ x 3))
     (* 5 (^ x 4))
     (^ x 5)))
```

Jacal also allows for output to be automatically typeset in TeX. This can be quite useful if one wants to use the results of one's computations in published articles. Continuing with the example of `(1+x)^5` above, we have:

```
e5 : set outgrammar tex;
e5 : e4;

e5: 1 + 5 x + 10 x^{2} + 10 x^{3} + 5 x^{4} + x^{5}

e6 : (1+1/x)^3/(1-1/y)^4;

e6: {\left(1 + 3 x + 3 x^{2} + x^{3}\right) y^{4}}\over{x^{3} - 4 x^{3} y +

                6 x^{3} y^{2} - 4 x^{3} y^{3} + x^{3} y^{4}}
```

After being included in a document in math display mode, these two examples will appear in the following way.

$$1 + 5x + 10x^2 + 10x^3 + 5x^4 + x^5$$

$$\frac{\left(1 + 3x + 3x^2 + x^3\right) y^4}{x^3 - 4x^3y + 6x^3y^2 - 4x^3y^3 + x^3y^4}$$

`priority` *int* [Flag]

The following examples show how to set the priority of printing terms.

```
e10 : a;

e10: [[[1, 2, 3]]]

e11 : show priority a;

;;; not a simple variable:  (((1 2 3) . ()) . ())

e12 : show priority b;

e12: 128

e13 : show priority c;

e13: 128

e14 : b+c;

e14: b + c

e15 : c+b;

e15: b + c

e16 : set priority b 200;
e16 : b+c;
```

Index

G

I

L

M

N

O

P

Q

R

S

T

U

X