Maxima — максимум свободы символьных вычислений

Автор: Тихон Тарнавский.

Что такое символьные вычисления

Так как в этом цикле статей речь пойдет о математической программе для символьных вычислений, для начала пару слов о том, что из себя представляют эти самые символьные или, как их еще называют, аналитические вычисления, в отличие от численных расчетов. Компьютеры, как известно, оперируют с числами (целыми и с плавающей запятой). К примеру, решения уравнения $x^2 = 2x + 1$ можно получить как −0.41421356 и 2.41421356, а $3x = 1$ — как 0.33333333. А ведь хотелось бы увидеть не приближенную цифровую запись, а точную величину, т. е. $1\pm\sqrt{2}$ в первом случае и 1/3 во втором. С этого простейшего примера и начинается разница между численными и символьными вычислениями. Но кроме этого, есть еще задачи, которые вообще невозможно решить численно. Например, параметрические уравнения, где в виде решения нужно выразить неизвестное через параметр; или нахождение производной от функции; да практически любую достаточно общую задачу можно решить только в символьном виде. Поэтому неудивительно, что и для такого класса задач появились компьютерные программы, оперирующие уже не только числами, а почти любыми математическими объектами, от векторов до тензоров, от функций до интегро-дифференциальных уравнений и т. д.

Максима в науке и образовании

Среди математического ПО для аналитических (символьных) вычислений наиболее широко известно коммерческое (*Maple*, *Mathematica*); это очень мощный инструмент для ученого или преподавателя, аспиранта или студента, позволяющий автоматизировать наиболее рутинную и требующую повышенного внимания часть работы, оперирующий при этом аналитической записью данных, т.е. фактически математическими формулами. Такую программу можно назвать средой программирования, с той разницей, что в качестве элементов языка программирования выступают привычные человеку математические обозначения.

Программа, которая стала темой статьи, работает на тех же принципах и предоставляет похожий функционал; самое радикальное ее отличие— то, что она не является ни коммерческой, ни закрытой. Другими словами, речь идет о свободной программе. На самом деле использование свободного ПО более естественно для фундаментальной науки, нежели коммерческого, так как модель, которая используется в свободном ПО— это модель открытости и общедоступности всех наработок. Очевидно, эти же свойства присущи и результатам научной деятельности. Используя такую схожесть подходов, можно фактически рассматривать расширения функционала свободных программ или дополнительные библиотеки, которые могут создаваться для своих нужд в процессе научных исследований, как неотъемлемую часть результатов таких исследований. И эти результаты могут использоваться и распространяться на усмотрение пользователя без оглядки на ограничения, налагаемые лицензиями исходного ПО. В случае же коммерческого ПО, которое находится в собственности его производителя, такого рода свободы значительно ограничены, начиная от невозможности свободно (и законно) передавать само такое ПО вместе с наработками и вплоть до возможных патентных исков от компании-разработчика ПО в случае распространения самодельных дополнительных библиотек к нему.

С другой стороны, основное направление, кроме научных разработок, где такие программы востребованы— это высшее образование; а использование для учебных нужд именно свободного ПО — это реальная возможность и для вуза, и для студентов и преподавателей иметь в своем распоряжении легальные копии такого ПО без больших, и даже сколь-нибудь существенных, денежных затрат.

Эта статья открывает цикл, посвященный свободной программе аналитических вычислений *Maxima*. Этим циклом я постараюсь дать вам наиболее полное впечатление о программе: он будет посвящен как принципам и основам работы с Maxima, так и описанию более широких ее возможностей и практическим примерам.

Немного истории

История проекта, известного ныне под именем Maxima, началась еще в конце 60-х годов в

легендарном MIT (Massachusetts Institute of Technology— Массачусетский Технологический институт), когда в рамках существовавшего в те годы большого проекта MAC началась работа над программой символьных вычислений, которая получила имя Macsyma (от MAC Symbolic MAnipulation). Архитектура системы была разработана к июлю 1968 г., непосредственно программирование началось в июле 1969. в качестве языка для разработки системы был выбран Lisp, и история показала, насколько это был правильный выбор: из существующих в то время языков программирования он единственный продолжает развиваться и сейчас — спустя почти полвека после старта проекта. Принципы, положенные в основу проекта, позднее были заимствованы наиболее активно развивающимися ныне коммерческими программами — Mathematica и Maple; таким образом, Macsyma фактически стала родоначальником всего направления программ символьной математики. Естественно, Macsyma была закрытым коммерческим проектом; его финансировали государственные и частные организации, среди которых были вошедшее в историю ARPA (Advanced Research Projects Agency; помните ARPAnet— предок интернета?), Энергетический и Оборонный Департаменты США (Departments of Energy & Defence, DOE and DOD). Проект активно развивался, а организации, контролирующие его, менялись не раз, как это всегда бывает с долгоживущими закрытыми проектами. в 1982 году профессор уильям Шелтер (William Schelter) начал разрабатывать свою версию на основе этого же кода, под названием Maxima. в 1998 году Шелтеру удалось получить от DOE права на публикацию кода по лицензии GPL. Первоначальный проект Macsyma прекратил свое существование в 1999 году. Уильям Шелтер продолжал заниматься разработкой Maxima вплоть до своей смерти в 2001 году. Но, что характерно для открытого ПО, проект не умер вместе со своим автором и куратором. Сейчас проект продолжает активно развиваться, и участие в нем является лучшей визитной карточкой для математиков и программистов всего мира.

Пару слов о программе

На данный момент Maxima выпускается под две платформы: Unix-совместимые системы, т.е. Linux и *BSD, и MS Windows. Я, конечно же, буду вести речь о Linux-версии.

Сама по себе Maxima— консольная программа, и все математические формулы отрисовывает обычными текстовыми символами. В этом есть как минимум два плюса. С одной стороны, саму Maxima можно использовать как ядро, надстраивая поверх нее графические интерфейсы на любой вкус. Их на сегодняшний день существует немало; в этот раз я остановлюсь на двух самых популярных (см. врезку)— и наиболее наглядных и удобных в работе, а об остальных поговорим в следующих выпусках; они тоже по-своему интересны, хотя более специфичны.

С другой стороны, сама по себе, без каких-либо интерфейсных надстроек, Maxima нетребовательна к железу и может работать на таких компьютерах, которые сейчас и за компьютеры уже никто не считает (это может оказаться актуальным, к примеру, для вуза или научной лаборатории, у которых денег на обновление парка машин скорее всего нет, а потребность в ПО для символьных вычислений возникнуть может).

Имена функций и переменных в Максиме чувствительны к регистру, то есть прописные и строчные буквы в них различаются. Это не будет в новинку любому, кто уже имел дело с POSIX-совместимыми системами или с такими языками программирования, как, скажем, C или Perl. Удобно это и с точки зрения математика, для которого тоже привычно, что заглавными и строчными буквами могут обозначаться разные объекты (например, множества и их элементы, соответственно).

Для того, чтобы начать работать с программой, вам понадобится пакет Maxima; если в стандартных репозитариях вашего дистрибутива его не окажется, то взять его можно на сайте проекта, адрес которого приведен во врезке.

Принципы работы с программой не зависят от того, какой интерфейс к ней вы выберете, поэтому я постараюсь Максимально абстрагироваться от конкретного интерфейса, ограничиваясь лишь небольшими комментариями в тех случаях, когда они ведут себя по-разному.

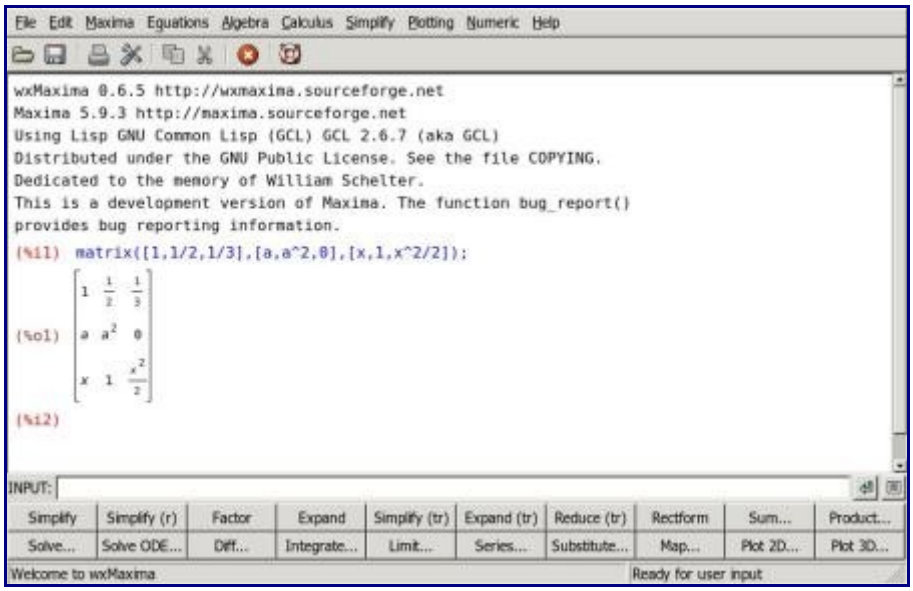
На данный момент последняя версия программы — 5.9.3, именно о ней я и буду говорить; если в вашем дистрибутиве пока присутствует более старая версия, вы в принципе можете использовать ее: и актуальная еще несколько месяцев назад 5.9.2, и вышедшая в конце прошлого года 5.9.1 не имеют с нынешней принципиальных различий.

Графические интерфейсы к Максиме

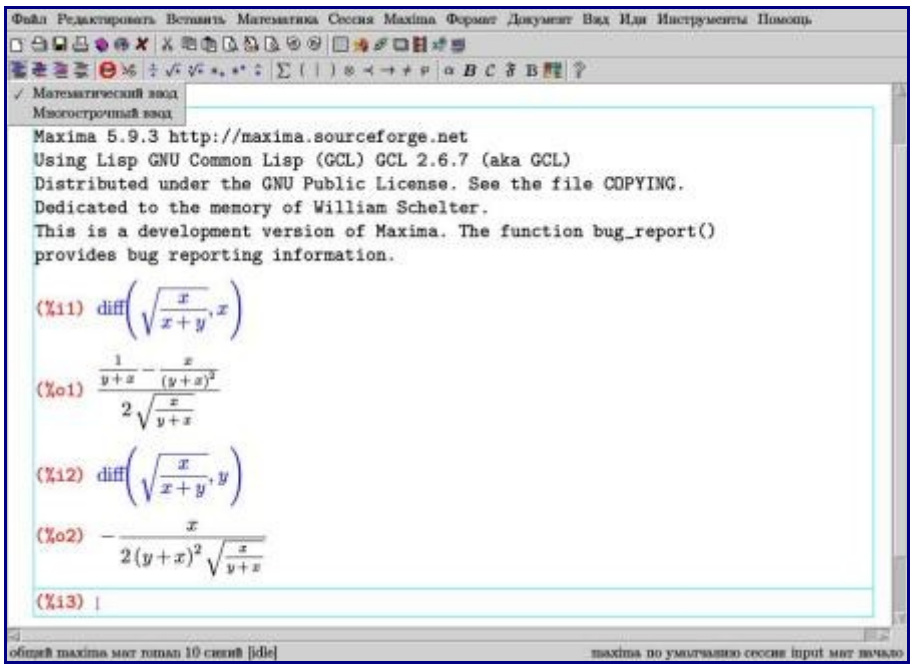
С точки зрения ознакомления с самой Maxima наибольший интерес представляют два интерфейса.

Первый— это отдельная самостоятельная графическая программа по имени *wxMaxima*. Она, как и

сама Maxima, помимо Linux/*BSD существует еще и в версии для MS Windows. В wxMaxima вы вводите формулы в текстовом виде, а вывод Максимы отображается графически, привычными математическими символами. Кроме того, большой упор здесь сделан на удобство ввода: командная строка отделена от окна ввода-вывода, а дополнительные кнопки и система меню позволяют вводить команды не только в текстовом, но и в диалоговом режиме. Так называемое «автодополнение» в командной строке на самом деле с таковым имеет лишь то сходство, что вызывается клавишей «`Tab`». Ведет же оно себя, к сожалению, всего лишь как умная история команд, т. е. вызывает ту команду из уже введенных в этой сессии, которая начинается с заданных в командной строке символов, но не дополняет до имен команд и их параметров. Таким образом, этот интерфейс наиболее удобен в том случае, когда вам нужно много вычислять и видеть результаты на экране; и еще, возможно, в том случае, если вы не очень любите вводить все команды с клавиатуры. Кроме того, wxMaxima предоставляет удобный интерфейс к документации по системе; хотя, так как документация поставляется в формате html, вместо этого можно использовать обычный браузер.



Второй достаточно интересный интерфейс к Maxima — это дополнительный режим в редакторе *TeXmacs*. Хотя этот редактор имеет общее историческое прошлое с широко известным Emacs, что явствует из названия, но практического сходства между ними мало. TeXmacs разрабатывается для визуального редактирования текстов научной тематики, при котором вы видите на экране редактируемый текст практически в том же виде, в котором он будет распечатан. В частности, он имеет так называемый математический режим ввода, очень удобный для работы с самыми разнообразными формулами, и умеет импортировать/экспортировать текст в LaTeX и XML/HTML. Именно возможностями по работе с формулами пользуется Maxima, вызванная из TeXmacs'а. Фактически, формулы отображаются в привычной математической нотации, но при этом их можно редактировать и копировать в другие документы наподобие обыкновенного текста. Maxima-сессия вызывается из меню: «*вставить* → *Сессия* → *Maxima*», при этом появляется дополнительное меню с командами Максимы. После запуска сессии можно уже внутри нее перейти в математический режим ввода (меню режимов ввода вызывается первой кнопкой на панели ввода) и при вводе также использовать элементы математической нотации. Этот интерфейс будет наиболее удобен тем, кто хочет использовать результаты вычислений в своих текстах и любит редактировать их в визуальном режиме.

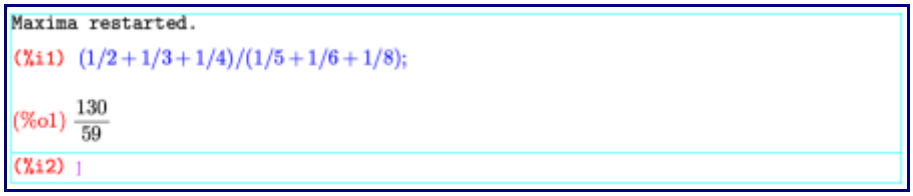


Приступаем к работе

После запуска Maxima-сессии мы видим перед собой такие строки:

```
Maxima restarted.
(%i1)
```

Первая— это сообщение о том, что ядро Максимы только что запустилось (вместо нее, в зависимости от версии и конкретной сборки, может выводиться краткая информация о программе); вторая— приглашение к вводу первой команды. Команда в Максиме— это любая комбинация математических выражений и встроенных функций, завершенная, в простейшем случае, точкой с запятой. После ввода команды и нажатия «`Enter`» Maxima выведет результат и будет ожидать следующей команды:



Для арифметических действий используются традиционные обозначения: `-`, `+`, `*`, `/`; `**` или `^` для возведения в степень, `sqrt()` для квадратного корня.

Далее я часто буду пользоваться для наглядности упомянутым во врезке математическим режимом ввода редактора TeXmacs. К примеру, приведенный выше код выглядел бы так:

$$(\%i1)\quad \frac{\frac{1}{2}+\frac{1}{3}+\frac{1}{4}}{\frac{1}{5}+\frac{1}{6}+\frac{1}{8}};$$

$$(\%o1)\quad \frac{130}{59}$$

Если для каких-то обозначений будет неочевидно, как записать их в строку, я буду пояснять это по ходу изложения.

Как видите, каждая ячейка имеет свою метку; эта метка— заключенное в скобки имя ячейки. Ячейки ввода именуются как `%i` с номером (`i` от *input* — ввод), ячейки вывода — как `%o` с соответствующим номером (`o` от *output* — вывод). Со знака `%` начинаются все встроенные служебные имена: чтобы, с одной стороны сделать их достаточно короткими и удобными в использовании, а с другой — избежать

возможных накладок с пользовательскими именами, которые тоже часто удобно делать короткими. Благодаря такому единообразию вам не придется запоминать, как часто бывает в других системах, какие из таких коротких и удобных имен зарезервированы программой, а какие вы можете использовать для своих нужд. К примеру, внутренними именами `%e` и `%pi` обозначены общеизвестные математические постоянные; а через `%c` с номером обозначаются константы, используемые при интегрировании, для которых использование буквы «c» традиционно в математике.

При вводе мы можем обращаться к любой из предыдущих ячеек по ее имени, подставляя его в любые выражения. Кроме того последняя ячейка вывода обозначается через `%`, а последняя ячейка ввода — через `_`. Это позволяет обращаться к последнему результату, не отвлекаясь на то, каков его номер.

```
(%i2) %+47/59;
```

$$(\%o2)\ 3$$

Здесь `%+47/59` — то же самое, что `%o1+47/59`.

Вывод результата вычисления не всегда нужен на экране; его можно заглушить, завершив команду символом `$` вместо `;`. Заглушенный результат при этом все равно вычисляется; как видите, в этом примере ячейки `%o1` и `%o2` доступны, хотя и не показаны (к ячейке `%o2` обращение идет через символ `%`, смысл которого расшифрован выше):

```
(%i1) sqrt(2)+3$
(%i2) 2*sqrt(2)+1$
(%i3) %-%o1;
```

$$(\%o3)\ \sqrt{2}-2$$

Каждую следующую команду не обязательно писать с новой строки; если ввести несколько команд в одну строчку, каждой из них все равно будет соответствовать свое имя ячейки. К примеру, здесь в строке после метки `%i1` введены ячейки от `%i1` до `%i4`; в ячейке `%i3` используются `%i1` и `%i2` (обозначенная как `_` — предыдущий ввод):

```
(%i1) asin(1/2)$acos(1/2);%i1+_;%o1+%;
```

$$(\%o2)\ \frac{\pi}{3}$$

$$(\%o3)\ \frac{\pi}{2}$$

$$(\%o4)\ \frac{2\,\pi}{3}$$

В wxMaxima и TeXmacs последнюю или единственную команду в строке можно не снабжать завершающим символом — это сработает так же, как если бы она была завершена `;` , т.е. вывод заглушен не будет. В дальнейших примерах я часто буду опускать `;` . Если вы выберете другой интерфейс, не забывайте ее добавлять.

Помимо использования имен ячеек, мы, естественно, можем и сами давать имена любым выражениям. По-другому можно сказать, что мы присваиваем значения переменным, с той разницей, что в виде значения такой переменной может выступать любое математическое выражение. Делается это с помощью двоеточия — знак равенства оставлен уравнениям, которые, учитывая общий

математический контекст записи, проще и привычнее так читаются. И к тому же, так как основной конек Максимы — символьная запись и аналитические вычисления, уравнения достаточно часто используются. Например:

```
(%i1) equation: x^3 - x = 0$
(%i2) solve(equation)
```

$$(\%o2)\ [x=-1, x=1, x=0]$$

В каком-то смысле двоеточие даже нагляднее в таком контексте, чем знак равенства: это можно понимать так, что мы задаем некое обозначение, а затем через двоеточие расшифровываем, что именно оно обозначает. После того, как выражение поименовано, мы в любой момент можем вызвать его по имени:

```
(%i3) diff(equation, x)
```

$$(\%o3)\ 3x^2 - 1 = 0$$

Любое имя можно очистить от присвоенного ему выражения функцией `kill()`, и освободить занимаемую этим выражением память. Для этого нужно просто набрать `kill(name)`, где name — имя уничтожаемого выражения; причем это может быть как имя, назначенное вами, так и любая ячейка ввода или вывода. Точно так же можно очистить разом всю память и освободить все имена, введя `kill(all)`. В этом случае очистятся в том числе и все ячейки ввода-вывода, и их нумерация опять начнется с единицы. В дальнейшем, если по контексту будет иметься в виду логическое продолжение предыдущих строк ввода-вывода, я буду продолжать нумерацию (этим приемом я уже воспользовался выше). Когда же новый «сеанс» будет никак не связан с предыдущим, буду начинать нумерацию заново; это будет косвенным указанием сделать «`kill(all)`», если вы будете набирать примеры в Maxima, так как имена переменных и ячеек в таких «сеансах» могут повторяться.

Доступ к документации Максимы

В примерах выше мы воспользовались двумя встроенными функциями. Как нетрудно догадаться из контекста, `solve` — это функция решения уравнения, а `diff` — функция дифференцирования. Практически весь функционал Maxima реализован через такие встроенные функции. Функция в Maxima может иметь переменное число аргументов. Например, функция `solve`, которую мы использовали с одним аргументом, чаще вызывается с двумя аргументами. Первый задает уравнение или функцию, чьи корни надо найти; второй — переменную, относительно которой нужно решать уравнение:

```
(%i1) a/x + a x = a^2$
(%i2) solve(%, a)
```

$$(\%o2)\ \left[a=\frac{x^2+1}{x}, a=0\right]$$

```
(%i3) solve(%i1, x)
```

$$(\%o3)\ \left[x=-\frac{\sqrt{a^2-4}-a}{2}, x=\frac{\sqrt{a^2-4}+a}{2}\right]$$

Если формула, задающая решаемое уравнение, содержит только один символ, как в предыдущем примере, то второй аргумент можно опустить, так как выбор, относительно чего нужно решать уравнение, все равно однозначен.

Вторая функция из наших новых знакомых — `diff` — также может принимать один аргумент; в этом случае она находит дифференциал заданного выражения:

(%i1) $\mathrm{diff}\left(x\,y+\frac{y}{x}\right)$

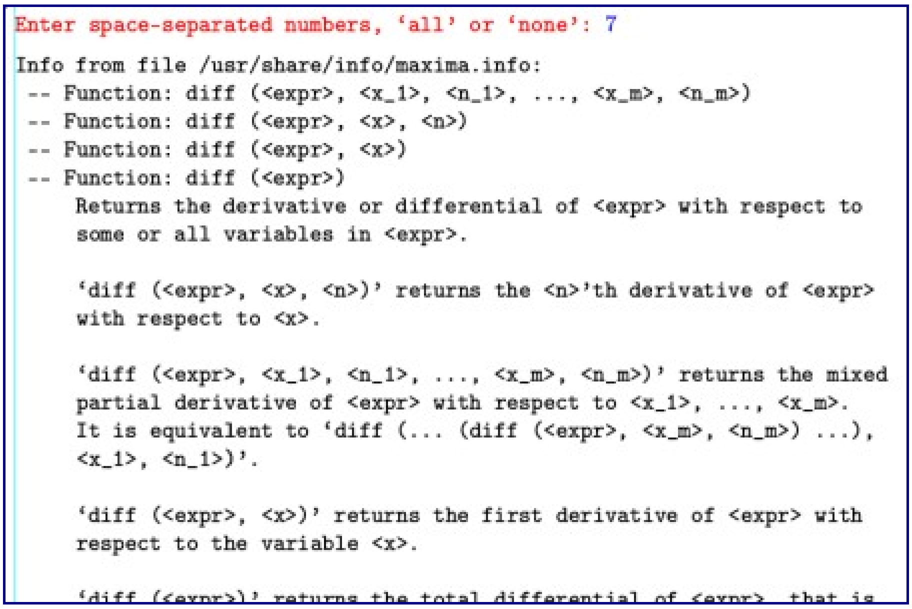
(%o1) $\left(x+\frac{1}{x}\right)\mathrm{del}(y)+\left(y-\frac{y}{x^2}\right)\mathrm{del}(x)$

Через `del(x)` и `del(y)` здесь обозначены дифференциалы соответствующих символов.

Для каждой встроенной функции есть описание в документации по Maxima. Оно содержит сведения о том, какие аргументы и в каких вариантах принимает функция, а также описание ее действия в разных случаях и конкретные примеры применения. Но, конечно, искать описание каждой нужной функции в html-документации или info-страницах не всегда удобно, тем более, что нужна эта информация, как правило, прямо в процессе работы. Поэтому в Maxima есть специальная функция — `describe()`, которая выдает информацию из документации по конкретным словам. Более того, специально для удобства получения справочной информации существует сокращенная версия вызова этой функции: `?` name вместо `describe(name)`. Здесь `?` — это имя оператора, и аргумент нужно отделять от него пробелом (выражение `?name` используется для вызова функции Lisp с именем `name`). Функция `describe` и оператор `?` выдают список тех разделов помощи и имен функций, которые содержат заданный текст, после чего предлагают ввести номер того раздела или описания той функции, которые вы хотите посмотреть:

```
(%i5) ?diff

0: (maxima.info)Differentiation.
1: Definitions for Differentiation.
2: Differential Equations.
3: Definitions for Differential Equations.
4: antidiff :Definitions for Differentiation.
5: covdiff :Definitions for itensor.
6: diff <1> :Definitions for itensor.
7: diff :Definitions for Differentiation.
8: evundiff :Definitions for itensor.
9: extdiff :Definitions for itensor.
10: idiff :Definitions for itensor.
11: liediff :Definitions for itensor.
12: poisdiff :Definitions for Special Functions.
13: ratdiff :Definitions for Polynomials.
14: rediff :Definitions for itensor.
15: setdifference :Definitions for Sets.
16: symmdifference :Definitions for Sets.
17: undiff :Definitions for itensor.
Enter space-separated numbers, 'all' or 'none':
```

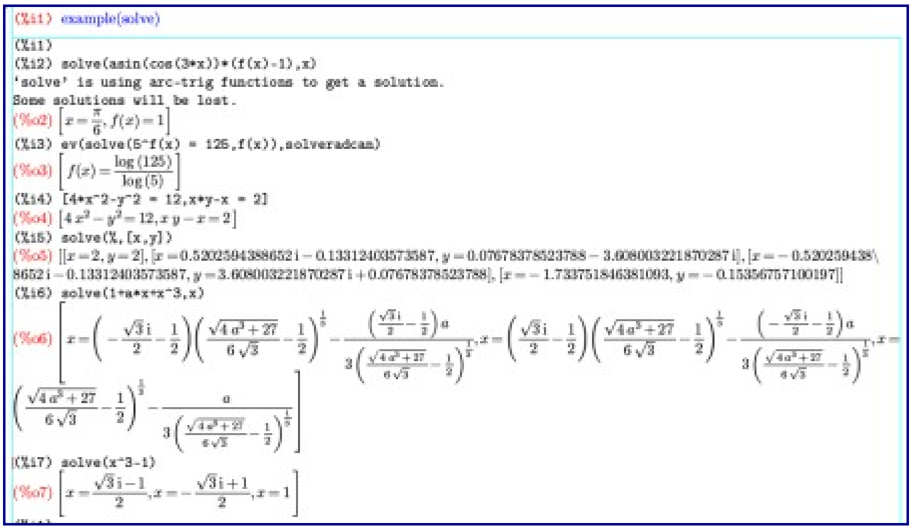
Когда вы выберете раздел, будет выдано его содержимое:



```
Enter space-separated numbers, 'all' or 'none': 7

Info from file /usr/share/info/maxima.info:
 -- Function: diff (<expr>, <x_1>, <n_1>, ..., <x_m>, <n_m>)
 -- Function: diff (<expr>, <x>, <n>)
 -- Function: diff (<expr>, <x>)
 -- Function: diff (<expr>)
     Returns the derivative or differential of <expr> with respect to
     some or all variables in <expr>.

     'diff (<expr>, <x>, <n>)' returns the <n>'th derivative of <expr>
     with respect to <x>.

     'diff (<expr>, <x_1>, <n_1>, ..., <x_m>, <n_m>)' returns the mixed
     partial derivative of <expr> with respect to <x_1>, ..., <x_m>.
     It is equivalent to 'diff (... (diff (<expr>, <x_m>, <n_m>) ...),
     <x_1>, <n_1>)'.

     'diff (<expr>, <x>)' returns the first derivative of <expr> with
     respect to the variable <x>.
```

Если для слова, которое вы ввели после `?` или `describe`, найдено единственное совпадение, его описание будет показано сразу.

Кроме справки, по многим функциям Maxima есть примеры их использования. Пример можно загрузить функцией `example()`. Вызов этой функции без аргумента отобразит список всех имен доступных примеров; вызов вида `example(name)` загрузит в текущую сессию и выполнит указанный файл примера:



Решение проблемы с запуском из-под TeXmacs

Если у вас возникли проблемы с запуском Maxima-сессии из TeXmacs, обратите внимание на то, кто у вас в системе выступает под именем `/bin/sh`. Дело в том, что инициализация всех разнообразных сессий реализована в TeXmacs'е через shell-скрипты, вызываемые именно с помощью `/bin/sh`. И в скрипте, отвечающем за сессию Maxima, используется возможность, которая не стандартизирована как обязательная для `/bin/sh`, но присутствует в его эмуляции bash. Другими словами, если у вас `/bin/sh` является не ссылкой на `/bin/bash`, а чем-то другим, то именно это может послужить причиной невозможности открыть Maxima-сессию (к примеру, в Debian и основанных на нем дистрибутивах кроме bash ссылку `/bin/sh` на себя может захотеть поставить еще и более легкий `dash`; в этом случае восстановить статус-кво можно с помощью `dpkg-reconfigure dash`). Если сделать `/bin/sh` ссылкой на `/bin/bash` не представляется возможным, можете попробовать поменять `#!/bin/sh` на `#!/bin/bash` в файле `/usr/lib/texmacs/TeXmacs/bin/maxima_detect`. Я написал об этой проблеме разработчикам TeXmacs, но еще не получил никакой их реакции, так что не могу пока сказать, будет ли исправлена эта недоработка в ближайших версиях.

Основные принципы

То, что Максима написана на Lisp, человеку, знакомому с этим языком, становится понятно уже в начале работы с программой. Действительно, в Максиме четко прослеживается «лисповский» принцип работы с данными, который оказывается очень кстати в контексте символьной математики и аналитических вычислений. Дело в том, что в Lisp, по большому счету, нет разделения на объекты и данные: имена переменных и выражения могут использоваться практически в одном и том же контексте. В Maxima же это свойство развито еще сильнее: фактически, мы можем использовать любой символ вне зависимости от того, присвоено ли ему какое-то выражение. По умолчанию символ, связанный с любым выражением, будет представлять это выражение; символ, не связанный ни с чем, будет представлять самого себя, трактуемого опять-таки как выражение. Поясним на примере:

```
(%i1) a b
```

$$a\,b$$

```
(%o1) a b
```

```
(%i2) a:x+y$ b:x-y$
(%i4) a b
```

$$(x-y)\,(y+x)$$

```
(%o4) (x-y) (y+x)
```

Из этого следует, в частности, что в выражение автоматически подставляется значение входящего в него символа только в том случае, если это значение было приписано символу до определения выражения:

```
(%i5) x:1/2$y:1/3$
(%i7) c:x-y$
(%i8) a b;a c
```

$$(x-y)\,(y+x)$$

```
(%o8) (x-y) (y+x)
```

$$\frac{y+x}{6}$$

```
(%o9) (y+x)/6
```

Если некоторый символ уже имеет какое-то значение, можем ли мы использовать в выражении сам этот символ, а не его значение? Конечно. Сделать это можно с помощью знака апострофа — введенный перед любым символом или выражением, он предотвращает его вычисление:

```
(%i10) b+y
```

$$-y+x+\frac{1}{3}$$

```
(%o10) -y+x+1/3
(%i11) b+'y
(%o11) x
(%i12) '(b+y)
```

$$y+b$$

```
(%o12) y+b
```

Результат выражения `%i12` был бы аналогичен и в том случае, если бы `b` и `y` не имели на тот момент никаких значений; таким образом, мы можем смело блокировать вычисление символа, даже не запоминая (или не зная), присвоены ли им вообще какие-то выражения.

Точно так же можно поступить с любой встроенной функцией, если мы хотим не выполнить ее, а

использовать в своем математическом контексте. Например, уже упомянутая функция дифференцирования может пригодиться нам для обозначения производной в дифференциальном уравнении; в этом случае, конечно, вычислять ее не надо:

(%i1) $'\mathrm{diff}(y,x)=y$

(%o1) $$\frac{d}{dx}y=y$$

Благодаря описанным особенностям работа в Максиме, с одной стороны, становится во многом похожей на традиционную «ручную» работу с математическими формулами, что практически сводит на нет психологический барьер в начале работы с программой. С другой стороны, даже на этом начальном этапе вы фактически избавлены от наиболее рутинной ручной работы, вроде отслеживания текущих значений символов, и можете полностью сосредоточиться на самой задаче. Конечно, блокировка вычислений — это не единственный способ влиять на то, как Максима будет вычислять то или иное выражение; этим процессом можно управлять довольно гибко.