ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
"ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ"

Н. В. Курбатова

“Аналитические вычисления в “Maple” ”

(семестровый курс лекций)

Ростов-на-Дону
2008 г.

Курбатова Н. В. “Аналитические вычисления в “Maple” ”. Семестровый курс лекций. Ростов-на-Дону, 2008. 146 с.

Курс лекций “Аналитические вычисления в “Maple” ” предусматривает освоение возможностей пакета символьных вычислений “Maple” ”. Современные версии пакета (“Maple 11”) рассчитаны не только для неосведомленного пользователя с поддержкой соответствующего интерфейса, но и продвинутого, пользователя, который стремится освоить предоставляемые пакетом возможности программирования, объектно-ориентированного подхода, а также процедуры и функции модулей, предназначенные для решения сложных задач специализированных предметных областей.

Оглавление

1 Введение

Пакет Maple – динамично развивающийся пакет символьных преобразований. Тенденции его развития приводят к тому, что этим пакетом легко, но с различным эффектом :) могут воспользоваться как подготовленные специалисты: математики и программисты, так и неподготовленные пользователи. В первом случае пакет предоставляет возможность программировать в среде Maple создавать пакеты процедур, направленные на выполнение сложных научно-технических задач с использованием встроенных функций и объектно-ориентированного подхода в программировании. Во втором случае широкие возможности использования математических нотаций облегчают написание выражений операторов, предназначенных для вычислений в среде пакета. В этом случае пользователь не должен быть знаком с синтаксисом Maple. К тому же предлагаемые тематические модули, снабженные удобным интерфейсом, для решения проблем в целом ряде предметных областей облегчает его использование и способствует скорому овладению пакетом. Серия интерактивных учебников и справочного инструментария, а также обучающий материал по математике, инженерным дисциплинам и физике дают возможность выполнять работу с минимальными усилиями.

В курсе лекций мы будем ориентированы на одиннадцатую версию пакета Maple. Эта версия пакета порадовала развитыми возможностями по созданию “умных” документов. С их помощью доступно создание таких документов, которые по своему качеству не уступают лучшим математическим изданиям. Режим слайд-шоу позволяет быстро превратить документ Maple в отличный дидактический материал, при этом во время показа слайдов все встроенные компоненты остаются активными. И тем и другим пользователям будет полезно освоить предлагаемый инструментарий для создания отчетов, документов и в том числе, научно-

методической литературы.

Аппарат математических вычислений пополнен пакетом Graph Theory, предназначенным для использования теории графов и 2–D, 3–D визуализации графов; пакетом Physics, для работы с инструментарием механики, теоретической физики и теории квантовых полей; пакетом дифференциальной геометрии Differential Geometry; расширены возможности пакета Linear Algebra, а также пакета, предназначенного для решения ОДУ, в том числе нелинейных, улучшены алгоритмы численного интегрирования и дифференцирования; улучшены компоненты быстродействия и как следствие – быстродействие операций с плавающей запятой. Предусмотрена возможность внешнего вызова компилятора FORTRAN. Пакет стал мощным средством не только учебных, но и научно-технических расчетов.

Глава 1

Maple–интерфейс. Среда символьных вычислений.

Первый раздел объединяет материал, который с одной стороны помогает пользователю Maple легко ориентироваться в предоставляемых возможностях пакета, освоить функции, “лежащие на поверхности”, наладить осмысленный контакт с системой, максимально облегчить работу и ускорить выполнение планируемых задач.

С другой стороны, материал текущей главы “знакомит” с основными типами данных, способами создания объектов рассматриваемых типов, преобразования и контроля.

Для возможности отложенного взаимодействия со средой приведены основные конструкции программирования, рассмотрена структура создания процедур и объединения процедур пользователя в личные библиотеки.

Таким образом, материал, предлагаемый в первой главе позволяет научиться не только интерактивно взаимодействовать с пакетом, но также используя все преимущества проблемных языков программируя.

1 Лекция 1. Maple–интерфейс. Форсированное освоение пакета

Первые шаги работы при запуске любого пакета традиционно начинаются с освоения предлагаемых функций панели команд. Интерфейс пакета Maple помимо панели команд, которая является типичной для системы Windows, содержит инструментарий, характерный для текстовых редакторов.

Режимы функционирования. Остановимся на создании нового файла при запуске пакета. Выбираемые опции будут определяться целью работы, а следовательно, и назначением результирующего электронного ресурса. В системе поддерживаются два режима функционирования, первый из них выбирается цепочкой опций $FILE \rightarrow NEW \rightarrow Worksheet\ Mode$, а второй – $FILE \rightarrow NEW \rightarrow Document\ Mode$. Они могут динамически изменяться в процессе работы, переходя один в другой (табл. 1.1). Очевидно, что работа в синтаксисе Maple рассчитана на продвинутого пользователя, но следует отметить, что в режиме документа можно достичь сравнимых результатов; в таблице 1.1 представлены основные отличия функционирования пакета в этих режимах.

На рис. 1.1 поля A выбирается режим $Worksheet$. Созданный в результате шаблон полученного в дальнейшем электронного ресурса будет ориентирован на традиционную для Maple форму взаимодействия с системой.

На том же рисунке (закладка B) приводится пример выбора опций, с использованием “специальной вставки” из “заготовок конструктора” математических выражений. Следующие шаги конкретизируют выбор опций и отклик системы:

- Выбор **1** – задание математической нотации;

Document Mode	**Worksheet Mode**
– свободная (от синтаксиса) форма, богатое содержание – не отображается знак (>) позиционирования строки ввода – по совместному нажатию $Ctrl$ и =; результат операции выводится в текущей строке – по нажатию $Enter$ – результат операции помещается в строке вывода – функции (методы) локального меню вызываются по нажатию правой кнопки мыши, как на строке ввода, так и на строке вывода – вставка знака (>) строки ввода с помощью меню команд переключает сессию в режим **Worksheet Mode**	– традиционного окружения $Maple$ – отображается знак (>) позиционирования строки ввода – по нажатию $Enter$ – осуществляется выполнение кода, и результат помещается в строке вывода – функции локального меню доступны по нажатию правой кнопки мыши только на строке вывода – по одновременному нажатию $Shift$ и $Enter$ осуществляется переход на новую строку ввода – переключение в режим **Document Mode** осуществляется при создании блока документа: $Format \longrightarrow CreateDocumentBlock$

Таблица 1.1: Основные отличия среды Maple в режимах Worksheet и Document

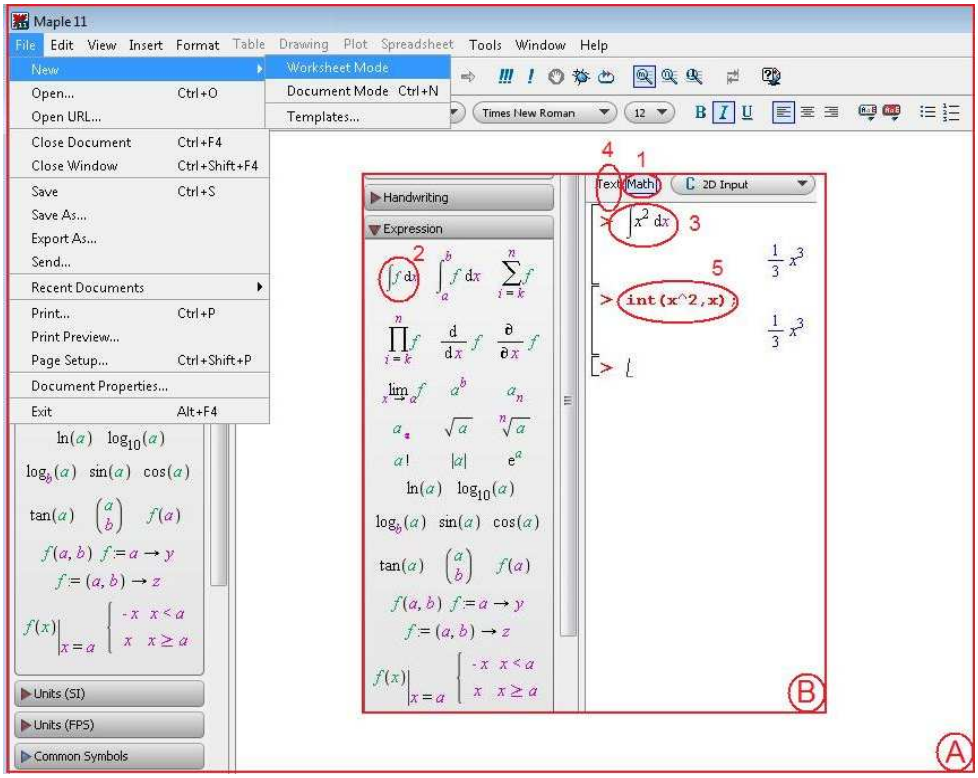


Рис. 1.1: Первые шаги. Панель команд

- В закладке *expression* из предлагаемых **2** – выбирается (здесь и далее в соответствующих контекстах имеется в виду с помощью мыши) необходимое математическое действие;
- **3** – в строке ввода редактируется вид подынтегральной функции; в строке ввода - результат интегрирования;
- в случае выбора опции *text* – **4**,
- **5** – осуществляются операции с использованием функций и синтаксиса языка Maple.

В этом случае функционирование в Maple рассчитано на пользователя системы Windows, который может ограничиться предоставляемыми возможностями, не погружаясь в синтаксис Maple. В режиме документа предусмотрены возможности редактирования типичных свойств документов Windows, к их числу относится, например, выбор шрифта, структурирование

документа (создание параграфов, секций, окружений). Наряду с перечисленными возможностями в документ Maple легко вставить (закладки *Insert*) таблицы *Excel*, изображения, графики, анимации. Далее на рис. 1.2 приводится пример создания блока документа, манипулирования стилями, выбора различных окружений: **1** – выбран режим Text; **2** – заголовок набран в стиле Annotation Title; **3** – автор в стиле Author; стили **4–6** подчеркнуты и интерпретируются предлагаемым текстом. В *Maple* можно

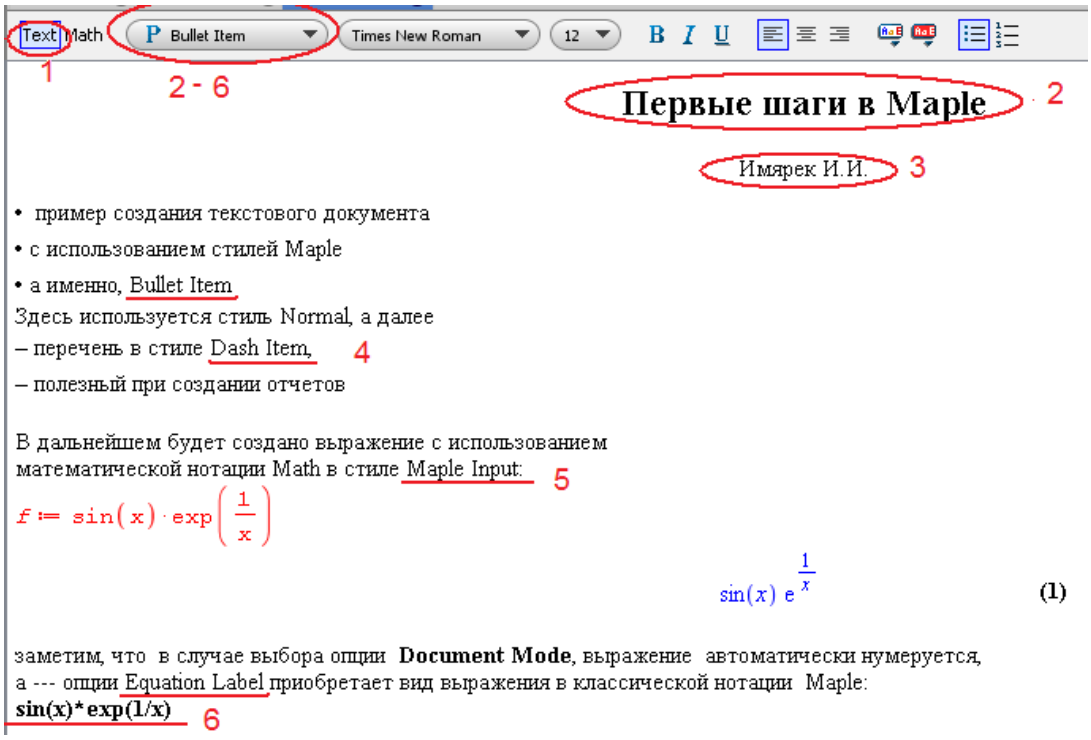


Рис. 1.2: Создание документа с активными элементами Maple

использовать буквы греческого алфавита. Для этого в командной строке набираются греческие буквы в шрифте *Times New Roman*, например, как указано на рисунке 1.2, и нотации, принятой в издательском пакете *LaTeX*. Логика соответствий, в том числе заглавных букв, прозрачно следует из примера, демонстрирующего их использование:

Для текстов, требующих написания греческих букв, можно выбрать шрифт *Symbol*, как показано на рис.1.3, или воспользоваться меню символов закладки *Greek* из “специальной вставки”.

Рис. 1.3: Пример соответствий греческих букв

Возвращаясь к основным функциям панели команд, заметим, что их назначение очевидно, однако нельзя не отметить новую функцию запуска слайд-шоу в режиме просмотра. Таким образом документ Maple не только является документом со всеми привычными преимуществами, обеспечиваемыми текстовыми редакторами, но и исполняемой программой с активными результантами (формулами, графикой, анимацией), а также является удобными в использовании презентационным материалом.

Рис. 1.4: меню *Greek*

Следует однако заметить, что работа в пакете, связанная с традиционным окружением, тоже претерпела некоторые изменения. В процессе работы система нумерует строки вывода и позволяет использовать результирующие выражения в качестве аргументов функций, для этого выполняется цепочка действий: *Insert* → *Label* и вместо

выражений используются метки как ссылки на соответствующие выражения (см. рис.). Нельзя забывать, что при освоении неистощимых возможностей Maple самым распространенным является метод “проб и ошибок”, который эффективен в сочетании с изучением содержимого

пакета и статей справки.

Комплект справочной поддержки пакета.

Справочная система по мере развития и совершенствования пакета тоже претерпевает заметные изменения. Наряду с традиционной справкой, включающей контекстный поиск, формирование закладок избранных статей, демоверсий, иллюстрирующих те или иные возможности *Help Resources* рис.1.7, система пополнена "быстрыми" интерфейсами *Quick Reference Card*, инструментарий которых позволяет моделировать операции в выбранных предметных областях и генерировать код выполняемых действий. Содержимое модулей пакета представлено на рисунке 1.6 и оказывается доступным в результате выполнения следующей цепочки панели команд: $FILE \rightarrow NEW \rightarrow Templates$

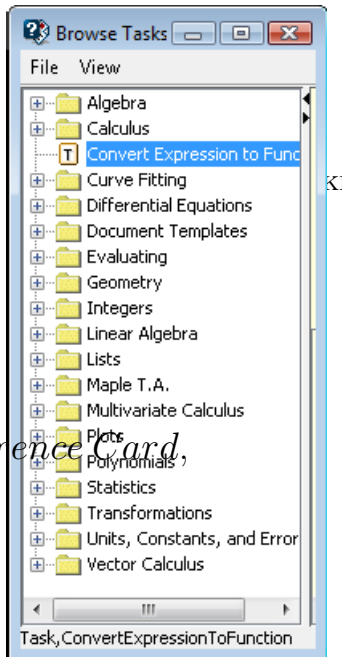


Рис. 1.6: Ресурс Maple

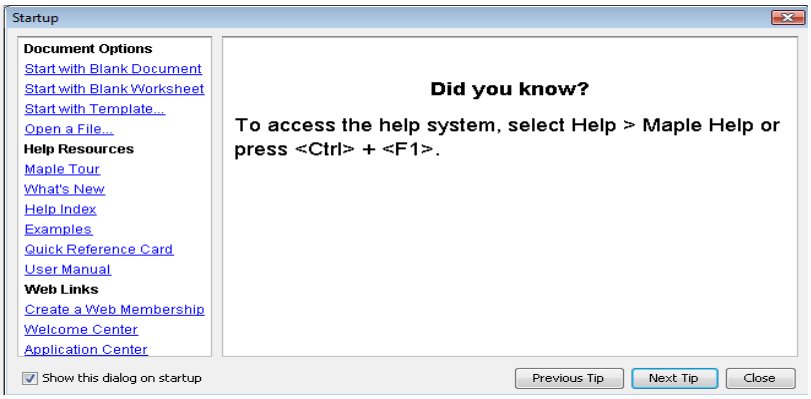


Рис. 1.7: Начало работы, стартовое меню

Интерактивная обучающая информация и схемы разработаны для следующих тематических зон: здесь (рис. 1.8) в **1** – приведены простейшие

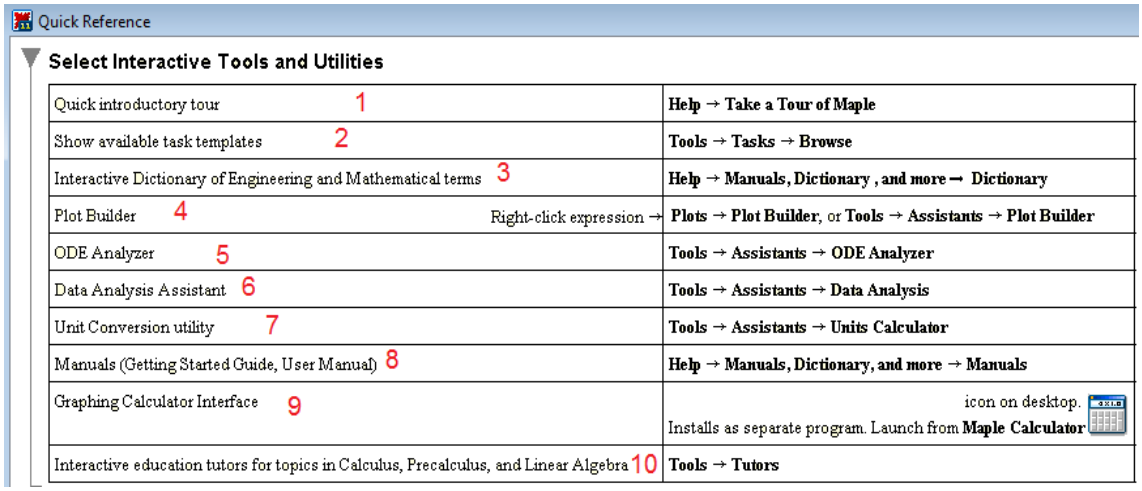


Рис. 1.8: Обучающие тематические интерфейсы

примеры создания документа, математических преобразований, экспресс-гид; указывается, как осуществляются **2** – активация информации о содержании модулей и полной справки по синтаксису и функциям каждого раздела; **3** – выбор закладок, ориентированных на математические вычисления и инженерное моделирование, снабженных справкой; **4** – редактирование активных графических объектов; **5** – классификация и анализ ОДЕ (обыкновенных дифференциальных уравнений) с помощью GUI; **6** – анализ данных, предлагается возможность визуализации, оценки основных статистик: среднего, медианы, размаха, асимметрии и эксцесса; **7** – преобразования, связанные с изменением единиц измерения; **8** – структурированная справка пользователя, с которой следует начинать знакомство с содержимым Maple; **9** – работа калькулятора Maple (самостоятельная программа, расположенная на рабочем столе); выполнение **10** – обучающих программ для серии предметных областей, с динамически задаваемыми параметрами и с возможностью последующей генерации программного кода.

В пакете предусмотрены механизмы конвертирования, которые обеспечивают использование информации, хранящейся во внешних файлах. Здесь (рис. 1.9) представлены следующие возможности: **1** – сохранение и чтение

Input and Output

Interactive data import assistant 1	**Tools** → **Assistants** → **Import Data**
Import audio or image file 2	**Tools** → **Assistants** → **Import Data**
Code generation (C, Fortran, Java™, Visual Basic®, MATLAB®) 3	Right-click expression → **Language Conversions**. See **?CodeGeneration** for help and details.
Publish document in HTML, LaTeX, or Microsoft® Word-RTF 4	**File** → **Export As** → select **HTML**, **LaTeX**, or **Rich Text Format**

Рис. 1.9: Активный ввод-вывод

документов и выражений, **2** – чтение файлов изображений; **3** – получение из *Maple* кодов кодов таких языков программирования, как $C++$, *Fortran*; **4** – конвертирование документа в $html-$, $tex-$, $rtf-$форматы.

Здесь приведен беглый обзор возможностей пакета и справочной системы, только практика позволит освоить Maple.

Активация библиотек. Когда запускается программа *Maple*, ни одна команда полностью не загружается в память. Некоторые команды находятся в стандартных или предметных библиотеках, которые подгружаются с помощью *with(NamePackage)*, вызов какой-нибудь одной команды *NameCommand* из пакета *NamePackage* можно осуществить, если набрать команду в специальном формате:

> *NamePackage[NameCommand](arguments);*.

Наиболее используемые пакеты *linalg*, *LinearAlgebra* – содержат операции линейной алгебры; *geometry* – функции для решения задач планиметрии; *geom3d* – стереометрии; библиотека *student* – команды аналитических преобразований, доступные с промежуточными вычислениями; список пакетов и их назначение приведены в приложении.

2 Лекция 2. Конструирование выражений. "Динамический" инструментарий

В данной лекции предлагается обзор "элементов конструктора", которые содержатся в закладках (выплывающих меню) главного интерфейса пакета. Работа в классическом варианте *Maple* проходит в режиме сессии – пользователь вводит предложения (команды, выражения, процедуры), которые обрабатываются *Maple*. Рабочее поле разделяется на три части: область ввода – состоит из командных строк. Каждая командная строка начинается с символа >; область вывода - содержит результаты обработки введенных команд в виде аналитических выражений, графических объектов или сообщений об ошибке; область текстовых комментариев в командной строке отделяется знаком # – "решетка".

Рис. 1.10: Символы

Наряду с конструкторами математических выражений в закладке *Expression*, о которой упоминалось выше, оперативно доступно использование математических констант и знаков специальных операций *Common Symbols*, содержащихся в меню см. рис. 1.10. Они помещаются в командную строку по нажатию левой кнопки мыши. Традиционное представление некоторых из них имеет вид: Pi – число π; I – мнимая единица i; $infinity$ – ∞, бесконечность; знаки отношений : $<$, $>$, $>=, <=$, $<>$, $=$. а также (вне меню, но необходимые) $true$, $false$ – логические константы, истинности и ложности выражений. Назначение знаков арифметических операций: $+$ - сложение; $-$ - вычитание; $*$ - умножение; $/$ - деление; $\wedge$ - возведение в степень; $!$ – факториал; $sqrt$ – $\sqrt{}$ корень квадратный, очевидно. Использование стандартных функций

с помощью закладок оправдано в интерактивном режиме, а для программирования пользователю необходима таблица 7.1 соответствий синтаксиса стандартных функций в приложении.

Знак равенства также используется для задания уравнений или ограничений, отметим, что выбирается правая (левая) часть уравнения с помощью функции rhs (lhs).

Элементарные вычисления в Maple. Пакет символьных вычислений изначально нацелен на аналитические преобразования. Если не задаются специальные преобразования, то вычисления не будут оптимизированы и записаны в окончательном виде.

Вычисления средствами $Maple$ осуществляются по умолчанию в поле рациональных либо иррациональных (содержащих радикалы) чисел. Если один из операндов записан в формате с плавающей точкой, то осуществляется преобразование типа результата, например,

$$> 28/703; \longrightarrow \frac{28}{703};$$

$$> 28/(703.); \longrightarrow 0.03982930299$$

Получить аналогичный результат можно по команде принудительного $eval$ выполнения в указанном классе объектов; $evalf$ – получить результат в формате вещественных чисел с плавающей точкой; $evalm$ – выполнить преобразование матриц; $evalc$ – комплексных преобразований. Комплексное число записывается в алгебраической форме $z = x+iy$, здесь i – мнимая единица, и в командной строке знак равенства заменяется знаком операции присвоения (:=). При вычислениях в пакете весьма востребованы следующие команды $frac$ – вычисление дробной части аргумента; $trunc$ – вычисление целой части; $round$ – округление выражения.

Синтаксис команд. Стандартная команда состоит из имени команды и ее параметров, которые отделяются запятыми. В конце каждой команды

в прежних версиях пакета должен быть знак (;) или (:), который подавляет вывод. В текущей версии для интерактивных вычислений можно обходиться без знака конца строки ввода в режиме Math; в режиме Text остается классический синтаксис.

Символ процента (%) служит для использования результата предыдущей команды в качестве аргумента новой операции, (%%) – результата двумя шагами раньше. Этот символ играет роль краткосрочной замены предыдущей команды с целью сокращения записи, например, рис.1.11

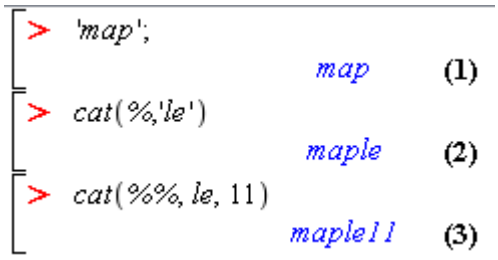


Рис. 1.11: Символ замещения

Такая возможность (оптимизации текста) предоставляется и новой закладкой *Label* функции панели команд *Insert*. Для присвоения переменной заданного значения используется знак присвоить (:=).

Оперирование с объектами в Maple. Пакет *Maple* – был создан как пакет символьной арифметики, с элементами symbol. Эти элементы лежат в основе конструирования алгебраических выражений, функций, процедур.

Maple обладает широкими возможностями для проведения аналитических преобразований алгебраических выражений, которые составляли суть математических формул. К ним относятся такие операции, как приведение подобных, разложение на множители, раскрытие скобок, приведение рациональной дроби к нормальному виду и многие другие. Любые объекты, с которыми приходится работать в пакете (уравнения, алгебраические выражения, матрицы, таблицы, списки и т.п.) являются следствием их конструирования из более мелких объектов, вплоть до базисных элементов. Объект Maple хранится в виде древовидной структуры (такая форма используется при программировании трансляторов). Ясно, что сложностью выражения и определяется число уровней такой структуры. Отвлекаясь от природы объекта, заметим, что для извлечения его частей

необходим метод, обеспечивающий такую процедуру. Для извлечения элементов первого уровня используется команда **op**, а элементов k–го уровня требуется k–кратное ее применение. Число элементов текущего уровня определяется командой **nops**. На рисунке 1.12 приведен пример,

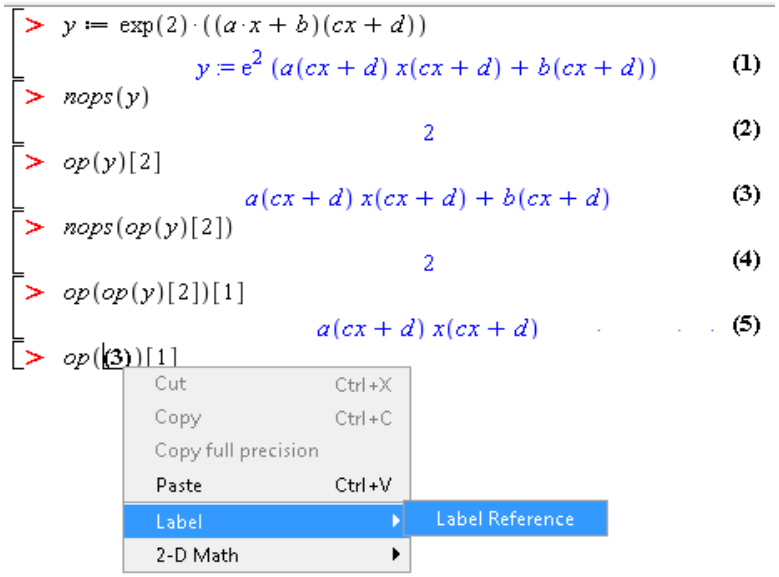


Рис. 1.12: Пример расщепления выражения

демонстрирующий сущность работы **nops** и **op**.

При интерактивном взаимодействии со средой проще использовать ссылку (*Insert Label*) на предшествующие результаты операции или редактировать метку ссылки, как показано на 1.12.

Если требуется выделить числитель и знаменатель рациональной дроби вида a/b, то это достигается командами **numer** и **denom**, соответственно.

```
> f:=(a 2+b)/(2*a-b);
```

$$f := \frac{a\text{^}2 + b}{2a - b}$$

```
> numer(f);
```

$a^2 + b$

```
> denom(f);
```

$2a - b$

В рассматриваемом примере можно обойтись и без использования этих функций, т.к. $op(f)[1]$ – числитель, а $1/op(f)[2]$ – знаменатель. Для работы с уравнениями важной командой выделения "части у целого" являются команды извлечения правой части уравнения **rhs(eq)** и левой – **lhs(eq)**.

> **eq:=leftpart=rightpart;** где **eq** – идентификатор объекта (уравнения), **rightpart** – правая часть уравнения, **leftpart** – левая.

> **eq:=a^ 2+b^2=c^2;**

$$eq := a^2 + b^2 = c^2$$

> **lhs(eq);**

$$a^2 + b^2$$

> **rhs(eq);**

$$c^2$$

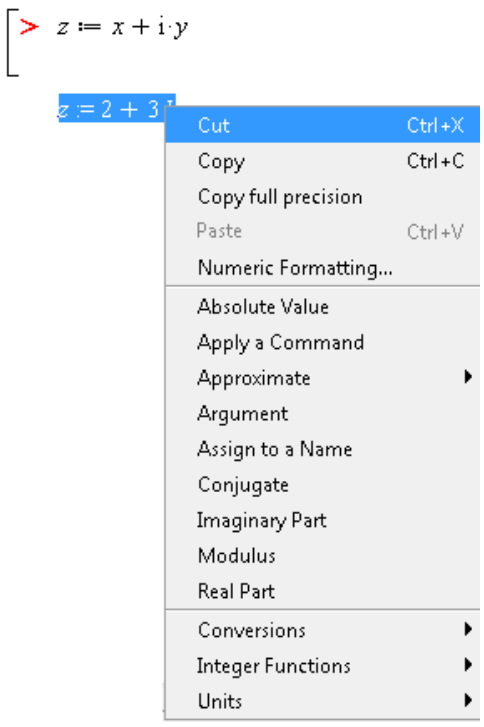


Рис. 1.13: Методы локального меню

Извлечение вещественной и мнимой части комплексного выражения с идентификатором z можно найти с помощью команд **Re(z)** и **Im(z)**, а комплексного сопряжения по команде **conjugate(z)**.

На рисунке 1.13 показана возможность использования функций локального меню. В Worksheet Mode при выделенной строке вывода правой кнопкой вызываются функции, которые поддерживаются для данного объекта. В рассматриваемом случае для комплексного числа доступны функции вычисления модуля и аргумента, сопряженного, обратного и т.д. В случае режима Document Mode методы могут быть вызваны и для строки ввода.

Следует отметить, что функции *op* и *nops* весьма полезны при работе с индексированными элементами, контейнерами, в этом случае *nops* определяет его длину (для списка, множества, последовательности, вектора), по команде **op(ContainerName)[k]** определяет k–й элемент слева, если k – положительно, и справа, если k – отрицательно.

3 Лекция 3. Базовые типы данных. Контроль типов

Выражения Maple могут присваиваться именам. Имя состоит из специального типа строки, которая в самой простой форме является буквенным символом с различением регистра (a-z, A-Z), цифрами (0-9), и символом подчеркивания (_). Кроме этого именем может служить любая строка, то есть любой набор символов, заключенных в обратные кавычки. Имя, обозначенное строкой, состоящей из разрешенных для имени символов совпадает с именем из тех же символов без кавычек. Например, ‘Name5‘ и Name5 одно и то же имя. Имена, начинающиеся символом подчеркивания используются в Maple как имена глобальных переменных.

Две обратные кавычки, вводимые последовательно в начале строки, интерпретируются как одна кавычка. Это позволяет включать символ обратной кавычки в текст строки.

Приведем примеры:

$>$ **‘ обратные “ кавычки ‘;**

обратные ‘ кавычки

$>$ **length(‘Very long string‘);** # имеет длину в 16 знаков, с пробелами.

Подстрока с 15-й по 20-ю позицию выбирается командой

$>$ **substring(‘ abcdefghijklmnopqrstuvwxyz ‘, 15 ..20);**

Приведем некоторые примеры допустимых имен.

Строковые имена:

$>$ MyVariable; whattype(%);

symbol

$>$ hello;

$>$ ‘ greatest variable ‘;

Функция type различает *имена* двух типов: string (строка) и indexed (индексное). Так индексные имена:

> A[1]; whattype(%); *indexed*

> A[i,j];

> A[i][j];

Оператор конкатенации

Для объединения строк применяется оператор конкатенации, который записывается в виде

cat(‘a‘, ‘b‘, ‘c‘, ...),

где *‘a‘, ‘b‘, ‘c‘, ... ‘* – строки, результатом является строка *‘abc...‘* .

Последовательность выражений. Один из более сложных объектов – последовательность выражений *sequences*. Последовательность выражений – некоторое количество выражений, отделенных запятыми и объединенных в единое образование знаком конца ввода. Данный тип востребован, поскольку последовательности выражений удобно использовать в качестве входных или выходных параметров, индексированных данных, которые легко преобразуются к данным более высокого уровня. Самый простой способ создавать последовательность выражений – непосредственное перечисление элементов.

> r:=1,2,3,4,5:

> convert(r,‘+‘);

15 # элементы последоватьности суммируются

> a+b,b+c, c+d, e+f, f+g;

a+b,b+c, c+d, e+f, f+g

качестве альтернативы, имеются еще способы создавать неявную последовательность выражений. Во-первых, с этой целью может использоваться оператор **$** (один, либо совместно с оператором диапазона, записываемым в виде **..** – двуточия). Этот оператор создает упорядоченные последовательности, например,

> x$6;

x, x, x, x, x, x

> $1..3;

1, 2, 3

> i^2$i=1..3;

1, 4, 9

> 2 * i$i=1..10;

> x[i] $ i = 1..3;

x_1, x_2, x_3

Заметим, что пробелы не изменяют синтаксис.

Во вторых, для этого предназначена функция–конструктор *seq* последовательности выражений, синтаксис и использование которой демонстрируют приведенные коды:

> seq (x/i, i=1 ..4);

$$\frac{x}{1}, \frac{x}{2}, \frac{x}{3}, \frac{x}{4}$$

> i:=evaln (i):

> seq(D(f), f=[sin,cos,tan,exp,ln]);

cos, $-sin$, $1 + tan^2$, exp, $z \longrightarrow 1/z$

Преимущество команды **seq** в скорости вычисления. В следующем примере показано как последовательность выражений используются в качестве аргумента в команде **max**( )

> max(Pi,exp(1)**,tan(5*Pi/6));**

Множества. Множество – неупорядоченная совокупность неповторяющихся выражений, заключенная в фигурные скобки, в нем могут содержаться элементы разных типов. Приведем примеры таких элементов.

> { 1, 1, 2, 3, 2 };

{ 1, 1, 2, 3, 2 };

> x:='x'; {a*x, 'rule of Maple',cat(my, lib),-3.141516};

$x := x; \{a\ x, `rule\ of\ Maple`, mylib, -3.141516\};$

> {'Ann', 'Kate', 'Polin'};

{Kate, Ann, Polin};

Как видно из последнего примера, порядок элементов множества может не совпадать с порядком, в котором их разместит *Maple*.

Операции над множествами: *union*, *intersect*, *minus*.

Так же как и в теории множеств для оперирования с такими элементами в *Maple* введены три основных оператора: оператор объединения *union*, он действует по правилам, справедливым для множеств, когда исключаются повторяющиеся элементы; оператор пересечения *intersect*, он создает набор элементов общих для перечисленных в качестве параметров множеств; и оператор исключения *minus*, который удаляет из первого множества элементы, содержащиеся во втором. Применение команд прозрачно иллюстрируется следующими примерами:

> { a, b, c, d } union { d, e, f };

или

> 'union'({ a, b, c, d }, { d, e, f });

$\{a, b, c, d, e, f\}$

> {x1, x2, x3, x4, x5} intersect {x2, x4, x6, x8, x10};

$\{x2, x4\}$

> {x1, x2, x3} minus {x1, y1};

$\{x2, x3\}$

Списки. Списки, также как последовательности и множества индексированные объекты, они заключаются в квадратные скобки. Списки – объекты с элементами, сохраняющими заданный порядок следования, при этом повторяющиеся элементы сохраняются в списке.

> [a, d, c, b, e]:

> [{ l, i, f, e }, { m, a, l, e }];

$[\{i, l, f, e\}, \{m, a, e, l\}]$

В последнем примере, каждый из трех множеств – элемент списка, порядок элементов внутри множеств может изменяться. Операции, которые справедливы для множеств справедливы после конвертирования списков.

Оперирование содержимым списка с помощью *select*, *remove*, *zip*, *sort*. Для извлечения элементов из списка по некоторому признаку существует команда *select*, которая по заданному правилу (логическому соотношению, являющемуся первым параметром аргумента команды) выбирает из списка (второй параметр) элементы и результат в той же последовательности помещает в список.

> large:=x-> is (x>3);

$large := x \longrightarrow is(x > 3)$

> L:=[6,1,9.5,Pi,sin(3)]:

> select(large,L);

$[6, 9.5, \pi]$

Команда *remove*, наоборот, исключает из списка удовлетворяющие заданному правилу элементы и выводит список оставшихся элементов.

> remove(large,L);

$[1, sin(3)]$

В качестве правила–признака может фигурировать также требуемый по логике программирования тип редактируемых данных. Так, например,

> select(type,L,numeric);

$[6, 1, 9.5]$

Объединение списков, например, X и Y выполняется с помощью команды

> [op(X),op(Y)]; # в этом случае *op* выбирает все элементы списков.

Легко объединить списки по заданному правилу командой *zip*. Первым параметром команды *zip* задается правило объединения, в рассматриваемом случае в попарные списки из множеств: **> zip((x,y)->{x,y},X,Y);**

или в список списков

> pare:=(x,y)->[x,y];

> P:=zip(pare,X,Y);

$[[1,2],[3,4],[5,6],[7,8]]$

для $X = [[1,3,5,7]$ и $Y = [2,4,6,8]$ в этом формате требуется задание точек для построения графиков.

С помощью *zip* легко выполнить конструирование имен:

> zip((x,y)->cat(x,y),[x,y,z,w,r],[1,2,3],1); $[x1, y2, z3, w1, r1]$

по заданному правилу. В следующей команде список является первым параметром аргумента, а правило – вторым.

> [sort([evalf(Pi),1,5,1/2],(x,y)->evalb(x>y));

$[5, 3.141592654, 1, 1/2]$

Здесь задано правило расположить элементы по убыванию

> bf:=(x,y)->is(x<y):

> sort([4.3,Pi,2/3,sin(5)],bf);

– по возрастанию.

Оператор композиции @ применяется для создания сложной функции. Он записывается в виде $f@g$ – для создания суперпозиции функций f и g

> (ln@sin)(x);

$ln(sin(x))$

или f@@n – для n-кратного применения функции f, например, создания цепных дробей

> f:= x-> 1/(1+x);(f@@3)(x);

$$\frac{1}{1+\dfrac{1}{1+\dfrac{1}{1+x}}}$$

или повторного дифференцирования $D@@2$.

Контроль типов. Каждому выражению или иному объекту в *Maple* в момент создания системой определяется его тип. Базисными типами объектов для выражений являются: **string,** integer**,** fraction**,** float , арифметические операторы ‘ +‘, ‘*‘, ‘ ‘ , и **function procedure**, **algebraic**. Для интерактивнрого определения типа объекта используется команда **whattype**:

> **whattype (15/37);**

fraction

> **whattype ([x,x,y,y,z]);**

list

> **whattype ((x+Pi) * (y-Pi/2));**

*

Осуществлять контроль типов при “отложенном” программировании позволяет команда

$$type(Argument, NameOfType)$$

Для анализа структуры объектов неоценима команда **hastype**, она позволяет проверить наличие элемента заданного типа, тогда как команда **has** позволяет выявить наличие заданного элемента в объекте.

Преобразование одного типа в другой, конвертирование, осуществляется по команде **convert**. Например: **convert(list, vector)** – преобразование некоторого списка **list** в вектор с теми же элементами; **convert(expr, string)** – преобразование математического выражения в его строковый аналог. Для вызова подробной информации о назначении параметров команды **convert** следует обратиться к справочной системе, набрав **convert[termin]**. Таблица всех типов объектов, существующих в Maple, представлена в приложении. Полная информация о типах может быть получена по справке **>?type**.

4 Лекция 4. Конструкции языка. Процедуры. Библиотеки

Базисные конструкции языка программирования *Maple* аналогичны соответствующим конструкциям языков высокого уровня. Перечислим наиболее полезные из них.

If/then/else/fi или **If/then/else/end if**

С помощью этой конструкции обозначаются условные переходы, например,

> x:=3; if x < 3 then x^2 else x^3 fi;

Признак конца конструкции fi или $end\ if$ и заканчивается ";" – точкой с запятой. После каждой из лексем **then** и **else** может записываться любое количество программных кодов. К конструкции условного перехода может также добавляться элемент **elif** :

If/then/elif/then/ .../else/fi

Это команда условного перехода с введением дополнительных условий **elif/then**, которые могут повторяться.

> if 0<x and x<10 then x^2 elif x<0 then x else x^3 fi;

Здесь для $x \in (0, 10)$ вычисляется квадрат числа, для отрицательного – само числа, а для всех остальных – его куб.

Следующая конструкция – циклы. Синтаксис циклов в *Maple* предполагает две конструкции, одна из которых – цикл с заданным числом шагов

for/from/by/to/do/od

для задания цикла используются верхняя (to) и нижняя ($from$) границы и величина шага для переменной (by), меняющейся в диапазоне от нижнего к верхнему значению границы. Цикл выполняет последовательность команд, заключенных между *do* и *od* (*end do*) определяемое число раз. Переменная диапазона может быть также включена в вычисления.

> for i from 0 by 10 to 30 do print(sin(i)): od;

В приведенном примере печать не подавляется.

Конструкция цикла с неизвестным числом шагов имеет вид

While/do/od

Maple будет повторять оператор, заключенный между **do** и **od** пока логическое соотношение, записанное между лексемами **while** и **do** не станет истиной.

> n:= 1:

> while n < 10 do n:= n^2+1; od;

n:=2

n:=5

n:=26

Создание процедур. В *Maple* имеется возможность создавать собственные процедуры. Процедура начинается с заголовка. Заголовок состоит из имени процедуры (его пользователь определяет сам), далее следует обязательный оператор присваивания **:=** и служебное слово **proc**, идентифицирующее процедуру; в круглых скобках указываются ее формальные параметры, отделяемые запятыми.

Язык программирования Maple, вообще говоря, относится к языкам с недекларируемыми типами данных, в классическом понимании. Учитывая экстенсивные тенденции развития пакета и объектно-ориентируемые принципы, в Maple различается более пятидесяти типов объектов. В том случае, если в теле процедуры не предусмотрен анализ допустимых по логике программы типов формальных параметров, то рекомендуется задать их типы в самом заголовке процедуры. Тип переменной отделяется от имени переменной :: – двойным двоеточием. Следует отметить, что также можно задекларировать тип результата, например,

> **name:=proc(var1::type1, var2::type2, . . .)::type_of_result**

Переменные, не используемые за пределами процедуры следует описать сразу после заголовка как локальными переменными с помощью

служебного слова **local**, после которого через запятую перечисляются локальные переменные.

Далее располагается тело процедуры, последовательность команд, причем последняя команда (с идентификатором или нет) будет выводить окончательный результат выполнения процедуры. Процедура должна обязательно заканчиваться служебным словом **end**.

Общий вид процедуры (стандартный синтаксис):

> **name:=proc(var1::type1, var2, . . .)::typeproc**
local vloc1::typev1, vloc2,. . . ;
> **expr1;**
> **expr2;**
. . . . . .
> **exprn;**
> **end;**

Заметим, что не всем переменным декларируется тип, это не является обязательным, однако нестрогая типизация требует строгого анализа типов данных в самом теле процедуры.

Рассмотрим пример процедуры, разделенной на логические части. Далее приводится задание "шапки" процедуры: *заголовка с входными параметрами (квадратные скобки указывают на необязательность синтаксических элементов определения типа), задание списка локальных и глобальных переменных, описание назначения процедуры строкой символов)*:

a) задание заголовка с входными параметрами:

usersfun:=proc(g[::algebraic],a[::float],b[::float],n[::integer])[::list];

b) описание локальных и глобальных переменных:

[local gg,k[::list]; global r:list;]

c) спецификация процедуры:

[option inline;] # процедура типа inline должна состоять из одного выражения или последовательности и не иметь локальных или иных переменных, рекурсий или условных операторов; анализ указанных типов формальных параметров при обращении не осуществляется, поэтому лучше использовать в теле обычной процедуры;

**option remember;** # помнит и использует предшествующие значения процедуры

option module; option builtin;

d) комментарий о назначении процедуры:

#description "plot interpolation algebraic expresion";

Тело процедуры**:**

```
gg:=unapply(g,x); k:=[n$3];
if type(op(gg),procedure) then
r:=[[evalf(a+(b-a)/(n-1)*(i-1)),evalf(gg(a+(b-a)/(n-1)*(i-1)))]$i=1..n];
print(whattype(r));
plot(r); else print(k); fi;
end proc;
```

из процедуры передаются глобальные переменные и результат последнего исполняемого оператора.

С целью прерывания процедуры и вывода соответствующей ошибки, например, при неправильном вводе типа параметра, в процедуре используется команда ERROR(‘строка сообщения‘).

```
> SUM:=proc(n)
local i,total;
if not type(n,integer) then
ERROR(‘Вводить можно только целое число‘);
fi;
total:=0;
for i from 1 to n do
```

```
total:=total+i;
od;
total;
end;
```

```
> SUM(a);
```

Error, (in SUM) Вводить можно только целое число

Вложенные процедуры. Рассмотрим примеры организации вложенных процедур. Как известно, команда map применяет указанную в качестве первого аргумента функцию ко всему списку (матрице и т.п.). Пусть задан список

```
> restart;lst:=[2,4,5,6];
```

и требуется его элементы возвести в степень, равную первому элементу списка ($lst[1] = 2$). Самый простой способ исчерпывается следующей командой:

```
> map(x->x^lst[1],lst);
```

Если программировать данную процедуру как подпроцедуру (процедуру вложенную в главную), возможны различные варианты реализации:

I-й способ предлагают следующие инструкции:

```
> restart; lst:=[2,3,4,5];
                              lst := [2, 3, 4, 5]
> map(x->x^lst[1],lst);
                               [4, 9, 16, 25]
> v:='v';out:=proc(x::list)
  local v;
  v:=x[1];
  map(y->y^v,x);
  end;
                            v := v
  out := proc(x::list) local v; v := x[1]; map(y→y^v, x) end proc
> out(lst);
                          [4, 9, 16, 25]
```

II-й способ заключается в том, чтобы продекларировать переменную v как глобальную в процедуре out.

III-й способ заключается в передаче переменной v во внутреннюю процедуру через параметр внутренней процедуры:

```
> out:=proc(x::list)
> map(proc(y,z) y z; end,x,x[1]);
> end;
```

IV-й способ – использование команды unapply вместо предлагаемого оператора (->) или proc().

Так следующие коды, по-видимому, являются самыми лаконичными

```
> out:=proc(x::list)
unapply(y x[1],y);
 map(%,x)
 end;
```

V-й способ – еще один прием - использование команды подстановки subs для замены глобальной переменной вложенной процедуры значением локальной переменной процедуры out.

```
>  out:=proc(x::list) local v;
global w;
print(‘это w‘,w);
v:=x[1];
map(subs('w'=v,y->y w),x);
end;
> out(lst);
```

Здесь w берется в кавычки, чтобы исключить ошибки в случаях, когда глобальной переменной w предварительно могло быть присвоено некоторое значение. В кавычках w используется как имя. Последний способ является наиболее надежным, но менее прозрачным.

Обобщая, заключаем, что для передачи значения некоторой переменной a в процедуру *int*(), вложенную в процедуру в *out*() применяются следующие приемы:

1. в процедуре out() определить переменную a как глобальную;

2. передать переменную a во вложенную процедуру int() через параметр вложенной процедуры: int(..,a);

3. определить (если это возможно) вложенную процедуру при помощи оператора-функции, созданного командой unapply;

4. передать значение локальной переменной a процедуры out() во вложенную процедуру int() при помощи вспомогательной глобальной переменной w процедуры out() и функции подстановки subs(w=a,inproc()).

Отладка процедур. Для отладки процедуры наиболее простым методом пошагового выполнения команд является применение средства printlevel. Это глобальная переменная, которая при начальной загрузке Maple равна единице. Если присвоить ей большее целое значение, то при выполнении процедуры на экран будут выводиться последовательность присвоений, вход и выход из процедуры с перечислением значений параметров процедуры (трассировка процедуры). Величина переменной printlevel определяет уровень (глубину) вложенных процедур, которые будут трассироваться при выполнении программы. Для того, чтобы читалась процедура с глубиной вложения n нужно задать $printlevel := n * 5$. Применим трассировку для процедуры *out* случай первого способа, в результате

> printlevel:=10;

> out([2,3,4,5]); -> enter out, args = [2, 3, 4, 5]

-> enter unknown, args = 2

<- exit unknown (now in out) = 4 -> enter unknown, args = 3

<- exit unknown (now in out) = 9 -> enter unknown, args = 4

<– exit unknown (now in out) = 16 –> enter unknown, args = 5

<– exit unknown (now in out) = 25

<– exit out (now at top level) = [4, 9, 16, 25]

Таким образом отслеживается работа процедуры. Если в программе встречается системная ошибка, то при значении $printlevel > 3$ Maple распечатает на экране параметры всех процедур, выполняемых на данный момент, значения локальных переменных и команду, которая выполнялась при возникновении ошибки.

Для трассировки процедур используются также команды Maple $trace$, $untrace$, которые, в отличие от выше описанного метода, позволяют выборочно трассировать некоторые из вложенных процедур, чтобы исключить громоздкий вывод на дисплей. Эти команды могут использовать в следующих вариантах: $trace(f); trace(f, g, h, ...); untrace(f); untrace(f, g, h, ...)$; здесь f, g, h, ... – имена процедур, которые будут трассироваться командой Команда trace инициирует в течение выполнения программы вывод на дисплей точек входа, результат выполнения операторов программы и точек выхода трассируемой процедуры. В точках входа выводятся значения фактических параметров процедур, в точках выхода – значения возвращаемых функций. Функция untrace выключает трассировку по указанным процедурам.

Для отладки процедур в Maple имеется также более мощное средство – отладчик процедур. Отладчик вызывается автоматически при встрече с одной из меток, установленных в процедуре: $stopat$, $stopwhen$, $stoperror$, которые означают инициирование режима отладки когда встретится вычисление заданного идентификатора, условия, ошибочная ситуация, соответственно. Когда отладчик вызывается он выводит на экран выражение, команду, которая должна выполняться следующей и приглашение ввода команды отладчика $DBG>$.

Команды $next$, $step$, $into$, $outfrom$, $return$ – инициируют выполнение

следующего шага цикла, шага, вход (выход) в (из) процедуру, возвращение на предыдущий уровень; *quit*, *done*, *stop* – выход из отладчика. Более подробная информация содержится в справочной системе.

Библиотеку пользователя легко создать, следуя приведенной инструкции, состоящей из следующих этапов:

1. Объявление таблицы – библиотеки пользовательских процедур (с любым именем, например, **mylib**, создается файл mylib.m):

>**mylib:=table();**

$$mylib := \mathrm{table}([])$$

2. Добавление в библиотеку элемента (процедуры) с именем **rk:**

> **mylib[rk]:=proc(r,k) r+k; end proc;**

$$mylib_{rk} := \mathbf{proc}(r, k)\ r + k\ \mathbf{end\ proc}$$

3. Просмотр (и присоединение) содержимого **mylib:**

> **with(mylib);**

$$[rk]$$

4. Добавление в библиотеку **mylib** элемента (процедуры) **f:**

> **mylib[f]:=proc(g,a,b,n)**
unapply(g,x);
[evalf(a+(b-a)/(n-1)*(i-1)),evalf(%(a+(b-a)/(n-1)*(i-1)))] $i=1..n;
plot([%]);
end proc;

> **with(mylib); # просмотр (и активизация) библиотеки mylib**

$$[f, rk]$$

5. Сохранение пользовательской библиотеки (лучше в своем каталоге, если в текущем – без указания пути)

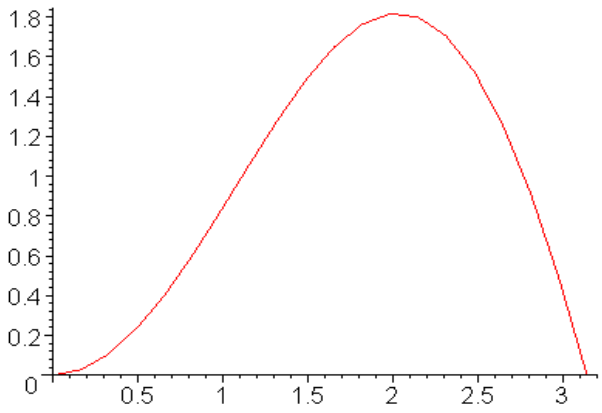
>**save mylib, или "c:\\users\\my_lib.m";**

> **restart;** # библиотека mylib недоступна

> **read "c:\\users\\my_lib.m";**# чтение файла my_lib.m библиотеки mylib

> **with(mylib);** # активизация библиотеки mylib

> **f(sin(x)*x,0,Pi,20);** # Пример использования процедуры f библиотеки mylib



mylib[f]:='mylib[f]'; # удаление процедуры f из библиотеки

makehelp позволяет добавить инструкцию (записанную в текстовом файле) использования процедуры или функции из f.e. пользовательской библиотеки

5 *Вопросы для самопроверки и задания*

1. **Очистка памяти (Workspace) осуществляется по команде**

   (a) clean

   (b) clear

   (c) restart

   (d) clc

2. **Внести комментарий в строку ввода позволяет знак**

(a) #

(b) c

(c) %

3. **Если строка ввода заканчивается одним из перечисленных символов, то подавляется вывод**

(a) ;

(b) :

(c) .

(d) !

4. **Продолжения строки ввода**

(a) ...

(b) *shift* & *enter*

(c) _

5. **С помощью какой команды присоединяется встроенная библиотека или библиотека пользователя**

(a) include

(b) add

(c) with

(d) readlib

6. **Результат предыдущей команды используется с помощью**

(a) %

(b) ans

(c) %%

7. **Функции локального меню становится доступным по нажатию**

   (a) левой кнопки мыши

   (b) правой кнопки мыши

   (c) обеих кнопок мыши

8. **Определите тип выражения:**

$$> v := [sin(x),\ 2,\ x^3]$$

   (a) list

   (b) vector

   (c) algebraic

9. **В результате выполнения команды**

$$> op([sin(x),\ 2,\ x^3])[-3]?$$

   получается

   (a) sin(x)

   (b) NaN

   (c) reset

10. **Результат выполнения команды**

$$> nops(map(evalf, \{sin(Pi/2 * i)\ \$i = 1..5\}));$$

   (a) 1

   (b) 2

(c) 3

(d) 4

11. **В теле процедуры переход к новой строке осуществляется клавишам**

(a) Alt & F4

(b) Shift & Reset

(c) Shift & Enter

12. **Формальные параметры отделяются от описателя типа**

(a) двойным двоеточием

(b) тире

(c) точкой с запятой

13. **В процедурах Maple предусмотрена**

(a) нестрогая типизация

(b) строгая типизация

(c) все данные одного типа

14. **Следующая процедура является**

$$
\begin{aligned}
&> f := proc(n) \\
&option\ remember; \\
&if\ n < 2\ then\ n\ else\ f(n-1) + f(n-2) \\
&fi \\
&end\ proc;
\end{aligned}
$$

(a) рекурсивной

(b) системной

(c) встроенной функцией

15. **В результате работы следующего оператора цикла**

$$
\begin{aligned}
&> for\ x\ in\ [1,\ 2,\ -2,\ 3]\ do\\
&if\ x > 0\ then\ next\ else\\
&f := y- > sin(Pi/y);\ print(evalf(f(x))):\\
&fi\\
&od;
\end{aligned}
$$

в строке вывода

(a) пустая строка

(b) $sin(-\pi/2)$

(c) -1

16. **Локальные переменные в процедуре описываются обязательно**

(a) да

(b) нет

(c) все переменные в процедуре - глобальные

17. **По какой из следующих команд пользователь инициирует создание библиотеки**

(a) mylib:=table();

(b) mylib:=array();

(c) mylib:=seq(namefun[i],i=1..5);

18. **Следующие коды создают библиотеку пользователя,**

$$> mylib := table();$$

$$> mylib[rk] := proc(r, k)r + k; endproc;$$

$$> mylib[f] := proc(g, a, b, n);$$

$$unapply(g, x);$$

$$x := evalf(a + (b - a)/(n - 1) * (i - 1));$$

$$[x,\ evalf(\%(x)]\ \$i = 1 \ldots n;$$

$$plot([\%]);$$

$$end\ proc;$$

$$> mylib[rk] := [\ ];$$

а последний оператор

(a) извлекает процедуру rk

(b) удаляет из библиотеки процедуру rk

(c) очищает библиотеку от процедур

Глава 2

Алгебраические преобразования. Элементы линейной алгебры

Ядро Maple и стандартная библиотека предоставляют широкие возможности при выполнении достаточно сложных задач. Команды и функции стандартной библиотеки и ядра автоматически подгружаются при запуске пакета. Надо сказать, что ядро Maple настолько хорошо разработано и "заточено" под символьные преобразования, что такие, например, известные пакеты, как MatLab, используют это ядро; его оказалось дешевле купить, чем создавать продукт такого же уровня и назначения.

1 Лекция 5. Аналитические преобразование алгебраических выражений. Функциональные операторы

Аналитические преобразования алгебраических, тригонометрических выражений сводится к раскрытию скобок, приведению подобных, группировке, подстановке параметров и т.п. Так раскрытие скобок выражения осуществляется командой **expand**:

$>$ **eq:=(x+1)*(x-1)*(x 2-x+1)*(x 2+x+1);**

$$eq := (x+1)(x-1)(x^2-x+1)(x^2+x+1)$$

> **expand(eq);**

$x^6 - 1$

Факторизация алгебраического выражения, представление в виде произведения выполняется по команде **factor(eq)**:

> **>p:=x^5-x^4-7*x^3+x^2+6*x;**

$p := x^5 - x^4 - 7x^3 + x^2 + 6x$

> **factor(p);**

$x(x-1)(x-3)(x+2)(x+1)$

Команда **expand** может иметь дополнительный параметр, позволяющий при раскрытии скобок оставлять определенное выражение без изменений. Например, пусть требуется каждое слагаемое выражения $\ln x + e^x - y^2$ умножить на выражение $(x + a)$. Тогда в командной строке следует написать:

> **expand((x+a)*(ln(x)+exp(x)-y 2), (x+a));**

$(x + a)\ln x + (x + a)e^x - (x + a)y^2$

Дробь приводится к нормальному виду с помощью команды **normal(eq)**. Например:

> **f:=(a^4-b^4)/((a^2+b^2)*a*b);**

$$f := \frac{a^4 - b^4}{(a^2 + b^2)ab}$$

> **normal(f);**

$$\frac{a^2 - b^2}{ab}$$

Упрощение выражений осуществляется с помощью **simplify**:

> **eq:=(cos(x)-sin(x))*(cos(x)+sin(x)):**

> **simplify(eq);**

$2\cos(x)^2 - 1$

Приведение подобных членов в выражении — командой **collect(exp,var),** здесь exp – выражение, var – имя переменной, относительно которой следует собирать подобные. В команде $simplify$ в качестве параметров можно указать, опции, например, $trig$, которые позволяют упростить **simplify(eq,trig)** с использованием тригонометрических соотношений. Предусмотрены следующие опции-параметры: $power$ – для степенных преобразований; $radical$ или $sqrt$ – для преобразований в радикалах; exp – в виде экспонент; ln – логарифмов. Использование параметров задает "траекторию вычислений" и увеличивает эффективность работы команды $simplify$.

Объединить показатели степенных функций или понизить степень тригонометрических функций можно при помощи команды **combine(eq,param)**, где eq – выражение, $param$ – параметры, указывающие, какой тип функций преобразовать, например, $trig$ – для тригонометрических, $power$ – для степенных. Пример:

$>$ **combine(4*sin(x)^3, trig);**

$$-\sin(3x) + 3\sin(x)$$

Для упрощения выражений, содержащих не только квадратные корни, но и корни других степеней, лучше использовать команду **radnormal(eq)**:

$>$ **sqrt(3+sqrt(3)+(10+6*sqrt(3)) (1/3))=**
radnormal(sqrt(3+sqrt(3)+(10+6*sqrt(3)) (1/3)));

$$\sqrt{3+\sqrt{3}+(10+6\sqrt{3})^{1/3}} = 1+\sqrt{3}$$

Так преобразование осуществляется и с помощью команды $convert(exp, param)$, здесь exp – выражение, которое будет преобразовано в тип опции $param$. В частности, можно преобразовать выражение, содержащее $\sin x$ и $\cos x$, в выражение, содержащее только $\mathrm{tg}\, x$, если указать

в качестве параметра *tan*, или, наоборот, tg$x$, ctg$x$ можно перевести в sin$x$ и cos$x$, если в параметрах указать *sincos*.

Синтаксис любой команды, можно найти в справке *Maple* по запросу из командной строки

>?AnyNameFunction или осуществить индексный поиск (по алфавиту) искомой функции.

В табл.7.1 приложения приводится соответствие стандартных математических функции идентификаторам функций в пакете, в том числе тригонометрических.

Функции. В *Maple* имеется несколько способов представления функции.

1. Самый простой – задание функции алгебраическим выражением с помощью оператора присваивания (**:=**):

```
> f:=sin(x)+cos(x);
```

$$f := \sin(x) + \cos(x)$$

Если задать конкретное значение переменной x, то результат будет равен значению функции f при текущем x. Продолжая предыдущий пример, вычислим значение f при $x = \pi/4$, для этого следует записать:

```
> x:=Pi/4;
```

$$x := \frac{\pi}{4}$$

```
> f;
```

$$\sqrt{2}$$

В результате при заданном x функции f присваивается значение $\sqrt{2}$.

2. Иначе работает команда подстановки, она не изменяет значения функции, а лишь позволяет воспользоваться ее "правилом" для вычисления значения при некоторых заданных аргументах.

Рассмотрим ее синтаксис: $subs(\{x1 = a1, x2 = a2, \ldots, \}, f)$, здесь в фигурных скобках указываются переменные $x1,\ x2$ и их новые значения $a1,\ a2$, которые следует подставить в функцию f. Например,

> **f:=x*exp(-t);**

$$f := xe^{(-t)}$$

> **subs({x=2,t=1},f);**

$$2e^{(-1)}$$

указанным способом либо с тем же результатом по команде $eval(f, [x = 2, t = 1])$. Все вычисления в *Maple* по умолчанию производятся символьно, то есть результат будет содержать в явном виде иррациональные константы, такие как, e, π и другие. Чтобы получить приближенное значение в виде числа с плавающей точкой, следует использовать команду $evalf(expr, t)$, где $expr$ – выражение, t – точность, равная количеству значащих цифр после запятой. С ее помощью результат предыдущего примера вычисляется приближенно

> **evalf(%);**

$$.7357588824$$

Здесь использован символ (**%**) для вызова предыдущей команды.

3. Определение функции с помощью функционального оператора, который ставит в соответствие набору переменных $(x1, x2, \ldots)$ одно или несколько выражений $(f1, f2, \ldots)$. Например, определение функции двух переменных с помощью функционального оператора выглядит следующим образом:

> **f:=(x,y)->sin(x+y);**

$$f := x \longrightarrow \sin(x + y)$$

Для вычисление значения функции следует в скобках вместо аргументов функции указать их значения, а именно:

> **f(Pi/2,0);**

1

4. С помощью команды $unapply(expr, x1, x2, \ldots)$, где $expr$ – выражение, $x1, x2, \ldots$ – набор переменных, от которых оно зависит, выражение $expr$ преобразовывается в функциональный оператор:

> **f:=unapply(x 2+y 2,x,y);**

$$f := (x, y) -> x^2 + y^2$$

> **f(-7,5);**

74

Эта команда создает процедуру из математических выражения с аргументами x, y.

Заметим, что команда >**apply** позволяет преобразовать оператор **f** в g – алгебраическое выражение, например,

> **g:=apply(f,x,y);**

5. В *Maple* имеется возможность определения кусочно-определенных функций вида

$$f(x) = \begin{cases} f_1(x), \ x < a_1 \\ f_2(x), \ a_1 < x < a_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ f_n(x), \ x > a_n \end{cases}$$

посредством команды

> **piecewise(cond_1,f1, cond_2, f2, . . .)**

В результате ее применения функция

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & 0 \le x < 1 \\ \sin x, & x \ge 1 \end{cases}$$

записывается следующим образом:

```
> f:=piecewise(x<0, 0, 0<=x and x<1, x, x>=1, sin(x));
```

$$f := \begin{cases} 0 & x < 0 \\ x & -x \le 0 \text{ and } x - 1 < 0 \\ \sin x & 1 \le x \end{cases}$$

Так особенности использования функций и алгебраических выражений, иллюстрируются предлагаемой таблицей.

Expression x^2+y^2	**Function (operator)** $g(x,y) = x^2 + y$
`f := x^2 + y^2;`	`g := (x,y) -> x^2+y^2;`
`eval(f, [x=1,y=2]);` produces 5	`g(1,2);` produces 5
`plot3d(f,x=0..1,y=0..1);`	`plot3d(g(x,y),x=0..1,y=0..1);`

`f2 := unapply(f,x,y);`
`f2(1,2);`
produces 5
	`g2 := g(x,1);`
	`g2 + z;`
	produces x^2+1+z

Рис. 2.1: Функции и выражения

2 Лекция 6. Операции линейной алгебры

Решение уравнений. Для решения уравнений, а также систем уравнений, в *Maple* предназначена команда **solve(eq,x)**, где **eq** – уравнение, **x** – переменная, относительно которой уравнение разрешается. Результатом выполнения команды является решение в случае одного корня или последовательность из корней.

Если уравнение имеет несколько решений, которые понадобятся для дальнейших расчетов, то команде **solve** следует присвоить какое-нибудь имя: **R:=solve(eq,x)**, тогда **k**–е решение данного уравнения выбирается следующим образом: **R[k]**.

При решении систем уравнений **R=solve({eq1,eq2,...},{x1,x2,...})** первый параметр команды – переменная, являющаяся множеством уравнений, второй параметр – множество неизвестных системы. Результат выполнения команды – множество решений, записанных в виде равенств, левая часть – идентификатор переменной, правая – ее числовое значение. Полученные числовые значения не присваиваются неизвестным, без принудительного присвоения. Такое присвоение выполняется по команде **assign(R)** В рассматриваемом случае предполагается, что существуют точные решения. В противном случае не избежать привлечения численных методов.

Для *численного решения уравнений*, в тех случаях, когда трансцендентные уравнения не имеют аналитических решений, используется специальная команда **fsolve**, с синтаксисом **solve**.

$>$ **x:=fsolve(cos(x)=x,x);**

$$x{:=}.7390851332$$

Для *решения рекуррентных уравнений*, предназначена команда **rsolve(eq,f)**, она позволяет решить рекуррентное уравнение *eq* для целой

функции f, как и дОлжно, задается некоторое начальное значение функции $f(n)$ для поиска решения данного рекуррентного уравнения, например:

> **eq:=2*f(n)=3*f(n-1)-f(n-2);**

$$eq := 2f(n) = 3f(n-1) - f(n-2)$$

> **rsolve({eq,f(1)=0,f(2)=1},f);**

$$2 - 4\left(\frac{1}{2}\right)^n$$

Команда **solve** позволяет *решать функциональные уравнения*, например,

> **F:=solve(f(x) 2-3*f(x)+2*x,f);**

$$F := \mathbf{proc}(x) \text{ RootOf}(\_Z \text{ } 2 - 3^*\_Z + 2^*x) \mathbf{end}$$

с результатом решения в неявном виде; *Maple* умеет работать с такими объектами. Неявное решение функционального уравнения можно попытаться преобразовать в элементарную функцию, определяемую опцией команды **convert**. Эта эвристическая затея иногда приводит положительному результату, так в рассматриваемом примере удалось выразить решение в радикалах

> **f:=convert(F(x),radical);**

$$f := \frac{3}{2} + \frac{1}{2}\sqrt{9 - 8x}$$

Команда **solve**, для *решения тригонометрических уравнений*, определяет только главные решения, то есть решения в интервале $[0,2\pi]$. В Maple используются переменные операционной среды (ОС), их имя должно начинаться с лексемы $\_Env$, за которой следует последовательность разрешенных символов.

Так для того, чтобы получить все решения уравнения, следует инициировать опцию поиска всех решений для переменной ОС **_EnvAllSolutions:=true.** Например:

```
> _EnvAllSolutions:=true:
> solve(sin(x)=cos(x),x);
```

$$\frac{1}{4}\pi + \pi\_Z \sim$$

В *Maple* символ $\_Z \sim$ обозначает константу целого типа, поэтому решение данного уравнения в привычной форме имеет вид $x := \pi/4 + \pi n$, где n – целые числа.

При *решении трансцендентных уравнений* для получения корней в явном виде выполнение команды **solve** следует предварить заданием параметра **_EnvExplicit** следующим образом **_EnvExplicit:=true**; предлагаемый код демонстрирует решение системы трансцендентных уравнений

```
> eq:={ 7*3 x-3*2 (z+y-x+2)=15, 2*3 (x+1)+
3*2 (z+y-x)=66, ln(x+y+z)-3*ln(x)-ln(y*z)=-ln(4) }:
> _EnvExplicit:=true:
> s:=solve(eq,{x,y,z}):
> simplify(s[1]);simplify(s[2]);
```

$$\{x{=}2,\, y{=}3,\, z{=}1\},\ \{x{=}2,\, y{=}1,\, z{=}3\}$$

и преобразование вида решений.

Линейная Алгебра Большая часть команд решения задач линейной алгебры содержится в библиотеке **linalg**. Для использования ее функций следует загрузить библиотеку **with(linalg)**.

Способы задания векторов. Для создания вектора в *Maple* используется команда **vector([x1,x2,. . . ,xn]),** здесь в квадратных скобках указываются координаты вектора, отделяемые запятой,

>x:=vector([1,0,0]): x[i] – выбор i–й координаты.

Вектор можно преобразовать в список и, наоборот, с помощью команды **convert(vector, list)** или **convert(list, vector).**

Операции над векторами.

Сложить два вектора $\boldsymbol{a}$ и $\boldsymbol{b}$ можно с помощью двух команд:

evalm(a+b): и **matadd(a,b):**

Команда **matadd** позволяет вычислять линейную комбинацию векторов $\boldsymbol{a}$ и $\boldsymbol{b}$: $\alpha a + \beta b$, где α, β – скалярные величины, если использовать формат: **matadd(a,b,alpha,beta)**.

Скалярное произведение двух векторов $(a, b) = \sum_{i=1}^{n} a_i b_i$ вычисляется командой **dotprod(a,b).**

Векторное произведение двух векторов $[a, b]$ вычисляется командой **crossprod(a,b).**

Угол между двумя векторами $\boldsymbol{a}$ и $\boldsymbol{b}$ вычисляется с помощью команды **angle(a,b)**.

Норма вектора $\mathbf{a} = (x_1, ..., x_n)$, равная $\|\mathbf{a}\| = \sqrt{x_1^2 + ... + x_n^2}$, вычисляется с помощью команды **norm(a,2)**.

Нормирование вектора $\boldsymbol{a}$ осуществляется по команде **normalize(a),** результат выполнения – вектор единичной длины $\dfrac{\mathbf{a}}{\|\mathbf{a}\|}$.

Нахождение базиса системы векторов. Ортогонализация системы векторов по процедуре Грамма-Шмидта.

Если имеется система n векторов $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\}$, то с помощью команды **basis([a1,a2,. . . ,an])** можно найти базис этой системы.

При помощи команды **GramSchmidt([a1,a2,. . . ,an])** можно ортогонализовать систему линейно-независимых векторов $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\}$.

Операции с матрицами

Для задания матрицы в *Maple* используется команда **matrix(n,**

m, [[a11,a12,. . . ,a1n], [a21,a22, . . . ,a2m],. . . , [an1,an2,. . . ,anm]]), здесь **n** – число строк, **m** – число столбцов в матрице. Можно непосредственно задать матрицу своими элементами построчно в квадратных скобках через запятую, например,

> **A:=matrix([[1,2,3],[-3,-2,-1]]);**

$$A := \begin{bmatrix} 1 & 2 & 3 \\ -3 & -2 & -1 \end{bmatrix}$$

Матрицы специального вида в *Maple* можно создавать с помощью специальных команд. В частности, диагональная матрица строится командой **diag(v)**, здесь v – вектор диагонали.

Матрицу также можно создать с помощью функции $f(i,j)$ от переменных i, j – индексов матрицы: **matrix(n, m, f),** здесь n – число строк, m – число столбцов; пример, приведенный далее

> **f:=(i, j)->x^i*y^j;**

$$f := (i,j) \to x^i y^j$$

демонстрирует текущий способ создания матрицы:

> **A:=matrix(2,3,f);**

$$A := \begin{bmatrix} xy & xy^2 & xy^3 \\ x^2y & x^2y^2 & x^2y^3 \end{bmatrix}$$

Размерность матрицы можно определить с помощью команд **rowdim(A)** – число строк; **coldim(A)**– число столбцов.

Арифметические операции с матрицами. Сложение двух матриц одинаковой размерности осуществляется теми же командами, что и сложение векторов: **evalm(A+B)** или **matadd(A,B).** Произведение двух матриц может быть найдено с помощью двух команд:

evalm(A&*B); и **multiply(A,B);**

В качестве второго аргумента в командах, вычисляющих произведение, должна быть матрица с согласующимися размерностями, в том числе и вектор.

Чтобы матрицу умножить на константу и прибавить к каждому элементу константу, достаточно выполнить

> **evalm(2+3*C);**

Определители, миноры и алгебраические дополнения. Ранг и след матрицы. Определитель матрицы A вычисляется командой **det(A).** Команда **minor(A,i,j)** возвращает матрицу, полученную из исходной матрицы вычеркиванием i-ой строки и j-ого столбца. Минор M_{ij} элемента a_{ij} матрицы A можно вычислить командой **det(minor(A,i,j)).** Ранг матрицы A вычисляется командой **rank(A).** След матрицы A, равный сумме ее диагональных элементов, вычисляется командой **trace(A).** Выполнение этих функций не составляет особого труда.

Обратную матрицу A^{-1} , такую что $A^{-1}A = AA^{-1} = E$, где E – единичная матрица, можно вычислить двумя способами:

evalm(1/A); или **inverse(A).**

Транспонированную матрицу A' матрицы можно вычислить командой **transpose(A).**

Положительная или отрицательная определенность матрицы определяется по команде **definite(A,param),** здесь **param** может принимать значения: матрица **'positive_def'** – положительно определенная; **'positive_semidef'** – неотрицательно определенная;**'negative_def'** – отрицательно определенная; **'negative_semidef'** – неположительно определенная. Результатом действия команды будет константа истинности **true** или ложности **false** сделанного предположения.

Ортогональность матрицы проверяется командой **orthog(A).**

Функции от матриц вычисляются с помощью команд Maple. Так

возведение матрицы A в степень n производится командой **evalm(A^n).** Вычисление матричной экспоненты e^A возможно с помощью команды **exponential(A).** Например:

> **T:=matrix([[5*a,2*b],[-2*b,5*a]]);**

$$T := \begin{bmatrix} 5a & 2b \\ -2b & 5a \end{bmatrix}$$

> **exponential(T);**

$$\begin{bmatrix} e^{(5a)}\cos(2b) & e^{(5a)}\sin(2b) \\ -e^{(5a)}\sin(2b) & e^{(5a)}\cos(2b) \end{bmatrix}$$

> **evalm(T^2);**

$$\begin{bmatrix} 25a^2 - 4b^2 & 20ab \\ -20ab & 25a^2 - 4b^2 \end{bmatrix}$$

Собственные числа и собственные векторы матрицы. Из курса линейной алгебры известно, что если $A\boldsymbol{x}=\lambda\boldsymbol{x}$, то вектор $\boldsymbol{x}$ называется собственным вектором матрицы A, а число λ – собственным числом, соответствующим данному собственному вектору. Совокупность всех собственных чисел матрицы называется спектром матрицы. Если в спектре матрицы одно и тоже собственное число встречается k раз, это означает, что оно имеет k–ю кратность.

Для нахождения собственных чисел матрицы A используется команда **eigenvalues(A).** Для нахождения собственных векторов матрицы A используется команда **eigenvectors(A).** В результате ее выполнения будут получены собственные числа, их кратность и соответствующие собственные векторы.

Рассмотрим результат выполнения команды **eigenvectors,** на следующем примере:

Пример

Пусть матрица $A = \begin{bmatrix} 3 & -1 & 1 \\ -1 & 5 & -1 \\ 1 & -1 & 3 \end{bmatrix}$ имеет 3 собственных вектора: вектор $\mathbf{a}_1 = (-1, 0, 1)$ отвечает собственному числу $\lambda_1 = 2$ имеет кратность 1, $\mathbf{a}_2 = (1, 1, 1)$, отвечает собственному числу $\lambda_2 = 3$ имеет единичную кратность, $\mathbf{a}_3 = (1, -2, 1)$, отвечает собственному числу $\lambda_3 = 6$ имеет единичную кратность.

Задача решается средствами *Maple* следующим образом:

> **A:=matrix([[3,-1,1],[-1,5,-1],[1,-1,3]]):**

> **v:=eigenvectors(A);**

[2,1,{[-1,0,1]}], [3,1,{[1,1,1]}], [6,1,{[1,-2,1]}]

В строке вывода в квадратных скобках перечислена последовательность списков; каждый из списков содержит подобные данные; так первый список содержит первое собственное число $v[1][1]$, его кратность $v[1][2]$ и соответствующий собственный вектор $v[1][3]$ в виде множества.

Для вычисления *характеристического многочлена* матрицы A $P_A(\lambda) = \det(\lambda E - A)$ используется команда **charpoly(A,lambda).**

Минимальный многочлен (делитель) матрицы A находится по команде **minpoly(A,lambda).**

Канонические и специальные матрицы. Матрицу A приводится к нормальной форме Жордана с помощью **jordan(A).**

К треугольному виду матрица приводится, как минимум, тремя способами:

1) команда **gausselim(A)** приводит матрицу A к треугольному виду методом Гаусса; 2) команда **ffgausselim(A)** приводит матрицу A

к треугольному виду методом Гаусса без деления. Эта команда предпочтительней для работы с символьными матрицами, так как не производит нормировку элементов и исключает возможные ошибки, связанные с делением на нуль; 3) команда **gaussjord(A)** приводит матрицу A к треугольному виду методом Гаусса-Жордана.

Характеристическую матрицу $F(A) = \lambda E - A$ можно вычислить командой **charmat(A,lambda).**

3 Лекция 7. Свойства матриц, матричные уравнения, билиотека LinearAlgebra.

Система линейных уравнений $A\mathbf{x} = \mathbf{b}$ может быть решена стандартной командой **solve**, она находит решение системы линейных уравнений, записанных в развернутом виде:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ ..... \\ a_{m1}x_m + a_{m2}x_2 + ... + a_{mn}x_n = b_m \end{cases} .$$

Команда **linsolve(A,b)** из пакета **linalg** ориентирована на решение линейных систем $A\mathbf{x} = \mathbf{b}$. Аргументы этой команды: A – матрица, $\boldsymbol{b}$ – вектор.

С ее помощью можно также найти решение матричного уравнения $AX = B$, если в качестве аргументов этой команды указать, соответственно, матрицы A и B.

Ядро матрицы A – это множество векторов $\boldsymbol{x}$ таких, что произведение их и матрицы A равно нулевому вектору, а именно, $A\mathbf{x} = 0$. Поиск ядра матрицы A эквивалентен решению системы линейных однородных уравнений. Чтобы найти ядро матрицы A следует воспользоваться командой **kernel(A).**

Библиотека LinearAlgebra. Наряду с использованием функций библиотеки *linalg* разработаны функции и процедуры библиотеки LinearAlgebra, которые предназначены для решения родственных задач линейной алгебры.

Конструкторы создания объектов матриц – Matrix, а векторов – Vector. Назначение процедур и функций библиотеки идентично соответствующим процедурам и функциям, которые содержатся в linalg.

Основные операции сложения, умножения и обращения выполняют следующие процедуры, соответственно:

MatrixAdd,
MatrixMatrixMultiply,
MatrixInverse,

Конструктор матриц содержит значительное число параметров, которые управляют ее свойствами, так следующий пример кода демонстрирует создание диагональной матрицы с диагональю V; заметим, что аргументом конструктора $Vector$ должен быть список.

```
> r:=i^2$i=1..4;
```

$$r := 1, 4, 9, 16$$

```
> V:=Vector([r]);
```

$$V := \begin{bmatrix} 1 \\ 4 \\ 9 \\ 16 \end{bmatrix}$$

```
> M:=Matrix(1..4,1..4,V,shape=diagonal);
```

$$M := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 16 \end{bmatrix}$$

При создании симметричных матриц, в соответствии с кодами рис.2.2, система осуществляет заполнение матрицы с учетом симметрии, а также контроль симметрии.

Опции, создаваемых конструктором Matrix типов матриц, приводятся на рис.2.3: это матрицы антиэрмитовы, диагональные, симметричные, антисимметричные, единичные, триангулируемые, ленточные, эрмитовы, нулевые, постоянные, скалярные, Хессенберга. Специальные матрицы нулевая, диагональная, единичная могут быть созданы с помощью

```
> R:=Matrix(1..3,1..3,shape=symmetric):
> R[1,3]:=evalf(sin(Pi/12.));
```

$$R_{1,3} := 0.2588190451$$

```
> R;
```

$$\begin{bmatrix} 0 & 0 & 0.2588190451 \\ 0 & 0 & 0 \\ 0.2588190451 & 0 & 0 \end{bmatrix}$$

Рис. 2.2: Пример создания симметричных матриц

antihermitian antisymmetric band constant
diagonal identity hermitian scalar
symmetric triangular zero Hessenberg

ZeroMatrix,
DiagonalMatrix,
IdentityMatrix,

Рис. 2.3: Опции свойств матриц

перечисленных ниже функций.Собственные значения и векторы определяются с использованием процедур EigenValues и EigenVectors, основным отличием является форма организации выходных данных. Так собственные векторы выдаются в виде столбцов результирующей матрицы. Решения линейной системы осуществляется с помощью LinearSolve здесь также приводятся методы декомпозиции матриц в виде произведения треугольных матриц (верхне- и нижне-) LUDecompositions, а также в виде произведения двух матриц QRDecompositions, в том числе R–ортогональной.

Связь элементов array, matrix, Matrix Очевидно, что любые действия с матрицами (линейной алгебры) сводятся к повторяющимся операциям, которые предполагают использование циклов. Для формализации таких процессов в Maple были написаны две различные библиотеки процедур: **linalg** и **LinearAlgebra.**

В каждой из них существуют более или менее удобные возможности мобильной работы с матрицами, однако следует помнить, что эти две библиотеки "научены" работать с объектами, созданными собственными средствами.

Массив – основной элемент Maple, который конвертируется в матрицу; массив создается с помощью конструктора

> A := array(indexfcn, bounds, lists):

в качестве первого параметра $indexfcn$ могут быть такие спецификации как $(anti)symmetric$ ((анти)симметричный массив)

$identity$ (массив, с ненулевыми только диагональными элементами, равными единице)

$sparse$ ("разреженный" массив, в списке задаются только ненулевые элементы, если не задан список элементов, то будет создан нулевой массив)

в качестве второго параметра *bounds* должен быть целочисленный список изменения размерностей массива (для массива индексация может начинаться с нуля, для матрицы - нет)

в качестве третьего параметра *lists* приводятся элементы массива, (если параметр, задающий элементы типа *listlist* , например, [[1, 2, 3],[1,0,1],[3,4,5]], и индексы изменяются от единицы, то тип созданного массива – матрица

>**a:=array(identity,1..3,1..3);** – двумерный массив с единичными диагональными элементами;

> **evalm(a);** – визуализация, вычисление элементов массива;

или

>a:=array(1..2,1..2,[[1, 2],[3, 4]]); – форма создания массива

здесь важно, чтобы тип при задании элементов был **listlist**

Приведем сводную таблицу, в которой сравниваются команды общего назначения, правила формирования входных–выходных параметров, особенности функционирования. К числу важных отнесены процедуры создания специальных матриц; определения размерностей; операции редактирования; матричные операции; в таблицу включен конструктор векторов.

```
> A := <<0,-2,1,3>|<1,3,0,1>|<1,1,0,1>|<-3,4,1,0>>;
```

$$A := \begin{bmatrix} 0 & 1 & 1 & -3 \\ -2 & 3 & 1 & 4 \\ 1 & 0 & 0 & 1 \\ 3 & 1 & 1 & 0 \end{bmatrix}$$

```
> (P, Lmatrix, Umatrix) := LUDecomposition(A);
```

$$P, Lmatrix, Umatrix := \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{1}{2} & \frac{3}{2} & 1 & 0 \\ -\frac{3}{2} & \frac{11}{2} & 3 & 1 \end{bmatrix}, \begin{bmatrix} -2 & 3 & 1 & 4 \\ 0 & 1 & 1 & -3 \\ 0 & 0 & -1 & \frac{15}{2} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

В таблице также приводятся команды, позволяющие определить свойства матриц, провести необходимую по логике задачи декомпозицию

Назначение процедуры	средства в библиотеке **linalg**	Средства в библиотеке **LinearAlgebra**
Создание матрицы	>**a:= diag(1,1,1);** с помощью диагональной	> **a:=** > **IdentityMatrix(3);** - единичная матрица >**a:=Matrix(1..2,1..2,[[1,2],[3,4]]);**
Нахождение собственных чисел и векторов	*eigenvals , eigenvalues , eigenvectors , eigenvects* > **v := [eigenvectors(A)];** $v := [[4, 1, \{[1, 1, 2]\}], [-2, 2, \{[1, 1, 0], [-1, 0, 1]\}]]$ здесь выходной параметр имеет тип *listlist* каждый его операнд – список , состоящий из собственного значения, кратности, множества собственных и присоединенных векторов	*Eigenvalues , Eigenvectors* > **(v, e) := Eigenvectors(A);** здесь первый выходной параметр – вектор из собственных значений, второй – матрица, столбцы которой – собственные векторы
Определение размерности	Размеры неизвестных матриц (A - matrix) определяются: > **rowdim(A)** – количество строк (размерность вдоль строк) > **coldim(A)** – количество столбцов (размерность вдоль столбцов)	> **rowdim := RowDimension(A);** > **coldim := ColumnDimension(A);** > **m,n := Dimension(A);** ***m***– количество строк и ***n***-столбцов (размерность вдоль столбцов)

матрицы, (пример LU–декомпозиции)

а также решить систему линейных алгебраических уравнений.

Полный перечень процедур и функций обеих библиотек, а также доступ к их использованию можно получить в результате их присоединения с помощью *with*, имена функций соответствуют их назначениям. Синтаксические детали применения команд, а также правила формирования выходных параметров описываются в справке пакета по и доступны по

?*NameFunction* из командной строки. Продолжение таблицы (см. стр. 69)

4 *Вопросы для самопроверки и задания*

1. **Определите a[1,1]**

$$> a := array(1..4, 1..4, `identity`);$$

$$> a[1, 1];$$

   (a) 1

   (b) 2

   (c) 3

2. **Команда** *expand (factor, simplify, normal)* **предназначена для**

   (a) раскрытия скобок при выполнении аналитических преобразований

   (b) приведения подобных

   (c) сведения к операциям первого порядка

   (d) нормализации выражений

3. **Какой из перечисленных команд преобразуется выражение** f

   в аналог процедуры-функции

   (a) unapply(f, arg)

   (b) apply(f, arg)

   (c) myproc(f, arg)
   arg – список аргументов

4. **Работа цикла равносильна**

$$> tot := 1;\ r := 1,\ x,\ y,\ q^2,\ 3$$

$$> for\ z\ in\ r\ do$$

$$tot := tot * z;$$

$$end\ do;$$

   (a) mul(r[k], k=0..5);

   (b) product(r[k], k=1..5);

   (c) Product(r[k], k=1..5);

5. **Чему равно c[2,1]**

   в результате выполнения следующих команд

$$> with(linalg);$$

$$> a := matrix(2, 4, [1, 2, 3, 4, 5, 6, 7, 8]);$$

$$> b := matrix(2, 4, 1);$$

$$> c := zip((x, y) - > x + y, a, b);$$

$$> c[2, 1];$$

   (a) 5

   (b) 6

   (c) 7

6. **Для заданной системы линейных алгебраических уравнений**

   определить результат выполнения команд

$$> solve(2 * x + y = 3, x - y = 0, x, y);$$

$$> 2 * x + y;$$

(a) 3

(b) 2x+y

(c) error ...

7. **Тип выходных параметров процедуры solve**

(a) list

(b) float

(c) set

(d) nothing

8. **Оператор assign позволяет**

(a) присвоить результаты решения системы ее неизвестным

(b) выполнить выражение, являющееся его аргументом

(c) очистить значение переменной, являющейся аргументом assign

9. **Что происходит по следующей команде**

$$> a := array(1..5, 1..5);$$

(a) создается массив с постоянными элементами

(b) резервируется пустой массив заданной размерности

(c) задается динамический массив

Операции с матрицами и векторами	> **u := vector([1,x,y]);** > **v := vector([1,0,0]);** > **dotprod(u, v);** – скалярное произведение (пр.) > **crossprod(u,v);** – векторное пр. > **multiply(A, B, C);** – пр. матриц > **mulrow(A, n, f);** – в матрице A n-я строка умножается на выражение f > **transpose(A);** транспонирование > **inverse(A);** – обращение квадратной матрицы *delcols , delrows* – удаление столбцов, строк, см.*help*	> **DotProduct(L, M);** – скалярное произведение > **CrossProduct(V1, V2);** > **MatrixMatrixMultiply(A,B);** > **MatrixScalarMultiply(A,f);** > **Transpose(A);** транспонирование > **MatrixInverse(A);** обращение квадратной матрицы *DeleteColumn , DeleteRow* – удаление столбцов, строк, см.*help*
Вычисление свойств матриц	> **det(A);** – определителя > **cond(A, 1);** – числа обусловленности >**log:=definite(A, kind),** A – квадратная симметричная матрица, kind – спецификация; 'positive_def', – положительно определенная, 'negative_def', or 'positive_semidef', 'negative_semidef' (it is clear!); log принимает значения false ∨ true	> **Determinant(H,method=float);** - определителя > **ConditionNumber(A);** – числа обусловленности (**Norm(MatrixInverse(A, infinity))**), характеризует устойчивость решения системы > **IsDefinite(C, 'query' = 'positive_definite');** смысл спецификаций аналогичен
Декомпозиция матриц и решение систем	см.*help* *LUdecomp* *linsolve*	> **(p, l, u) := LUDecomposition(A);** **p,** – матрица перестановок **l** - нижнетреугольная, **u** – верхнетреугольная > **X := LeastSquares(A, b);** – решение системы AX= b в смысле м.н.к. > **LinearSolve(M);** решение линейной системы, здесь M – расширенная матрица, последний столбец – столбец правой части

Глава 3

Графика

По мере развития Maple происходила и эволюция графики: от графических команд до графических объектов, реализованных как структуры. Помимо программного управления свойствами объекта появилась возможность интерактивного редактирования его свойств. В рассматриваемых лекциях текущей главы будут изучены возможности построения графических объектов, заданных явно, неявно, параметрически, а также примеры анимации. Графики, которые строятся пакетом, конвертируются как в растровые форматы (*.bmp, *.jpeg, *.wmf) так и векторные (*.ps, *.eps) такого рода независимость делает привлекательным Maple для визуализации данных и расчетов.

1 Лекция 8. Интерактивная графика. 2–D графика в Maple

Самым простым способом построения графиков функции является его построение в интерактивном режиме. Для этого следует, выделив выражение функции из строки вывода, вызвать функцию *plot* локального меню (вызываемого правой кнопкой мыши в режиме выделенного графического окна (см. рис. 3.1)), при этом генерируется соответствующая команда. Очевидно, что такое взаимодействие с системой имеет обучающее

значение и важно на начальном этапе освоения пакета. Там же в локальном меню предлагаются методы, которые поддерживаются для графического объекта, и позволяют редактировать вид графика, изменяя его текущие свойства. Настройка изображения также может осуществляться с помощью

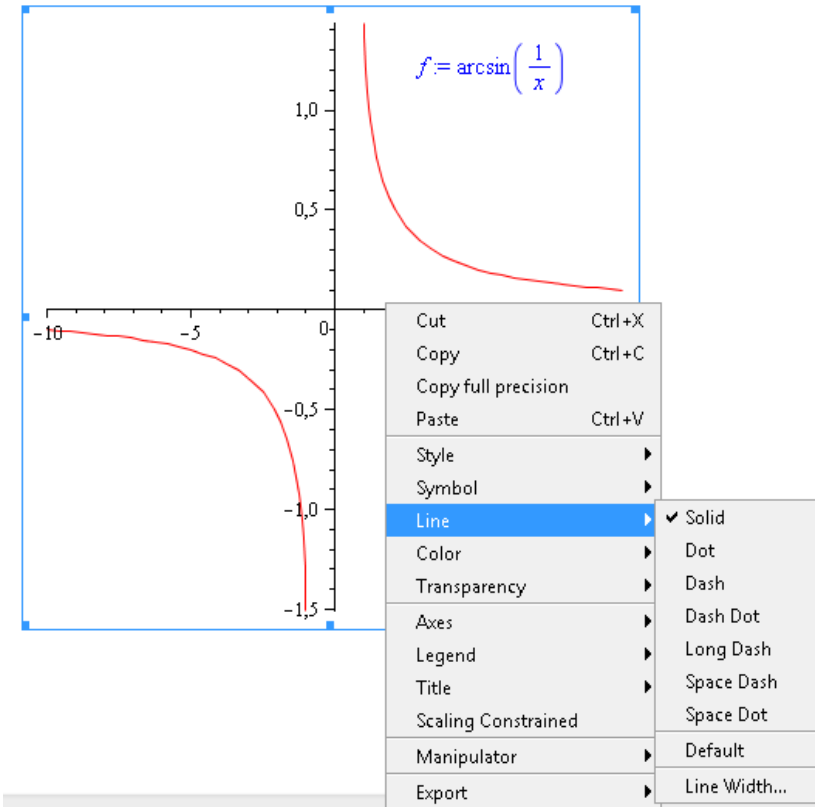


Рис. 3.1: Методы редактирования графики. Локальное меню

инструментария панели инструментов.

В пакете также предусмотрен интерфейс, позволяющий провести редактирование всех свойств графического объекта. На рисунке 3.2 приводится “алгоритм” действий, который приводит к осмысленному редактированию графики

Выбирая в локальном меню **1** – Plots; **2** – Plot Builder; получаем доступ **3** – изменить опции графического объекта; в появившейся панели свойств графика **4** – редактируем область изменения аргумента, идентификатор; **5** – задаем текст заголовка; **6** – изменяем свойства, связанные со стилем,

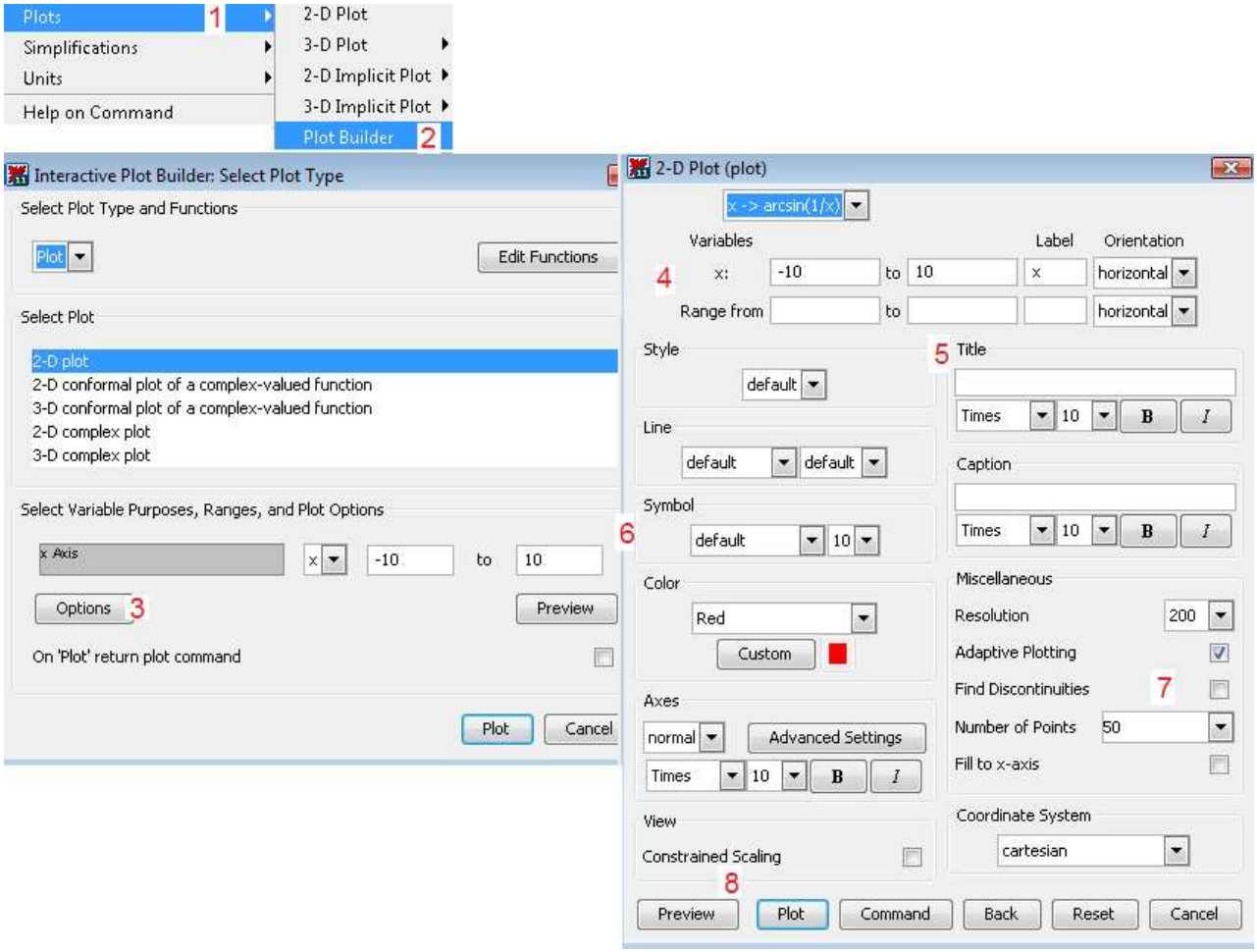


Рис. 3.2: Функции локального меню и управление свойствами графиков

толщиной линии, шрифтом, цветом, оформлением осей; **7** – выбираем опции, регулирующие разрешение, количество точек, поиск разрывов, выбор координат.

Метод “проб и ошибок” позволит освоить синтаксис графических команд и возможности визуализации.

Построение плоских фигур и графиков

Конструктором построения графиков функции $y = f(x)$ одной переменной, $a \le x \le b$, $c \le y \le d$, является команда

plot(f(x), x=a..b, y=c..d, parameters), здесь *parameters* – параметры управления изображением. В случае использования команды

без параметров будет получено изображение со свойствами по умолчанию.

Рассмотрим основные параметры команды *plot*:

1) $title = text$, где $text$ – заголовок рисунка;

2) $coords = polar$ – установка полярных координат (по умолчанию установлены декартовы);

3) $axes$ – установка типа координатных осей: $axes = normal$ – обычные оси; $axes = boxed$ – график в рамке со шкалой; $axes = frame$ – оси с центром в левом нижнем углу рисунка; $axes = none$ – без осей;

4) $scaling$ – установка масштаба рисунка: $scaling = constrained$ – одинаковый масштаб по осям; $scaling = unconstrained$ – график масштабируется по размерам окна;

5) $style = line[point]$ – вывод линиями (или точками);

6) $numpoints = n$ – число вычисляемых точек графика (по умолчанию $n = 50$);

7) $color$ – установка цвета линии: название цвета по-английски, например, $green$ – зеленый;

8) $xtickmarks = nx$ и $ytickmarks = ny$ – число меток по оси Ox и оси Oy, соответственно;

9) $thickness = n$, здесь $n = 1, 2, 3 \ldots$ – толщина линии (по умолчанию $n = 1$);

10) $linestyle = n$ – тип линии: непрерывная, пунктирная и т.д. ($n = 1$ – непрерывная, установлено по умолчанию);

11) $symbol = s$ – тип маркера точки: $box\, cross$, $circle$, $point$, $diamond$;

12) $font = [f, style, size]$ – установка типа шрифта для вывода текста: f задает название шрифта:$times$, $courier$, $helvetica$, $symbol$; $style$ задает стиль шрифта: $bold, italic, underline$; $size$ – размер шрифта в pt;

13) $labels = [tx, ty]$ – метки–названия координатных осей: tx – оси x и ty – оси y;

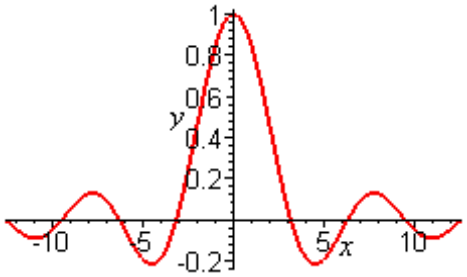
14) $discont = true$ – опция, указывающая режим истинного поведения

функции в окрестности разрывов в случае особенностей; $false$ – по умолчанию отражает качественное поведение.

С помощью команды *plot* можно строить помимо графиков функций $y = f(x)$, заданных явно, также *графики функций, заданные параметрически* $y = y(t), x = x(t)$, для этого следует указать опцию *parameters* в команде **plot([y=y(t), x=x(t), t=a..b], parameters)**.

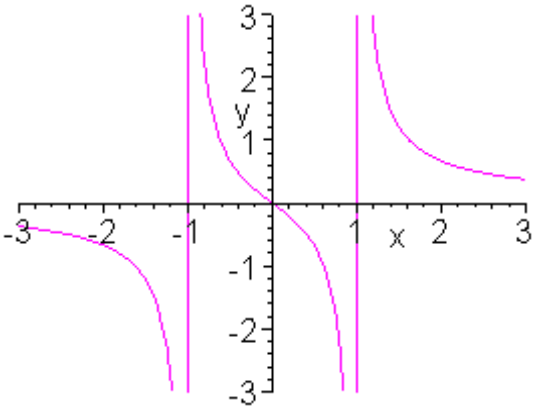
Так, чтобы построить график $y = \dfrac{\sin x}{x}$ жирной линией в интервале (-4π,4π), необходимо изменить параметры следующим образом:

> **plot(sin(x)/x, x=-4*Pi..4*Pi, labels=[x,y], labelfont=[times,italic,12], thickness=2);**



А *график разрывной функции* $y = \dfrac{x}{x^2 - 1}$

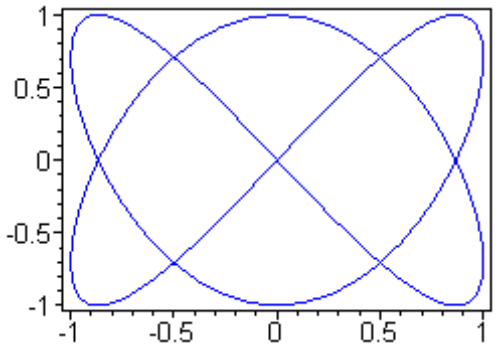
> **plot(x/(x 2-1),x=-3..3,y=-3..3,color=magenta);**



строится с автоматическим нанесением вертикальных асимптот.

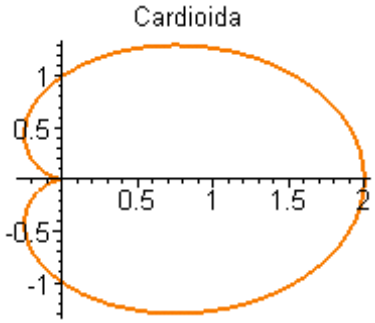
Построение *кривой, заданной параметрически*, $y = \sin 2t$, $x = \cos 3t$, $0 \le t \le 2\pi$, ограниченной рамкой может быть достигнуто по команде:

> **plot([sin(2*t),cos(3*t),t=0..2*Pi], axes=boxed, color=blue);**



Пользователем регулируется выбор системы координат, так *в полярных координатах* график кардиоиды $\rho = 1 + \cos\varphi$ с названием "*Cardioida*" будет построен по команде:

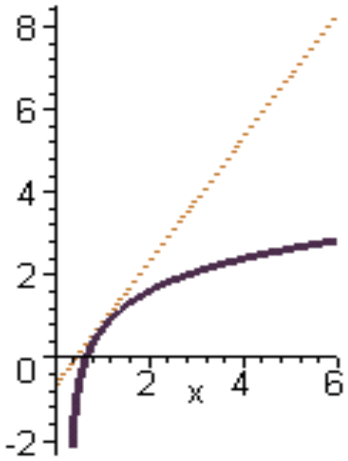
> **plot(1+cos(x), x=0..2*Pi,**
> **title="Cardioida", coords=polar, color=coral, thickness=2);**



Во всех предыдущих случаях на осях было построено по одной зависимости; *построение нескольких графиков* этой же командой будет выполнено, если в качестве первого аргумента команды будет задан список, содержащий алгебраические выражения зависимостей, подлежащих визуализации.

Так построить два графика функции $y = \ln(3x - 1)$ и касательную к нему $y = \frac{3}{2}x - \ln 2$ на одном рисунке получится по команде с указанными параметрами

> **plot([ln(3*x-1), 3*x/2-ln(2)], x=0..6,**
> **scaling=constrained, color=[violet,gold],**



linestyle=[1,2], thickness=[3,2]);

Чтобы построить функцию $F(x, y) = 0$, *заданную неявно*, необходимо воспользоваться командой **implicitplot** из графического пакета **plots: implicitplot(F(x,y)=0, x=x1..x2, y=y1..y2).**

2 Лекция 9. Структуры. Серия графических объектов

Помимо команд высокого уровня для построения графических объектов используются конструкторы $CURVES$ – ломаных; $POINTS$ – точечных объектов; $POLYGONS$ – многоугольников, все они используются в качестве первого аргумента универсальной команды $PLOT$, с их помощью формируются структуры, значения полей которых задают их свойства. Конструктор имеет следующую форму $CURVES([[x11, y11], ...[xmn, ymn]])$, здесь аргументами также могут быть последовательности из ломаных (кривых) с различным количеством сегментов.

Аналогично для точечных графических объектов

$POINTS([x1, y1], [x2, y2], ...[xn, yn])$, множество точек которых расположено в одной плоскости; данный конструктор применяется и для 3d-графического объекта, в этом случае совокупность точек (список списков) трехкомпонентная.

Конструктор $POLYGONS([[x11, y11], ...[xmn, ymn]])$ – позволяет строить закрашенные многоугольники; а

$TEXT([x, y], STRING, HORIZONTAL, VERTICAL)$ – текстовые объекты с текстом $STRING$, вертикальной или горизонтальной ориентацией текста; данная команда также используется в качестве первого аргумента команды $PLOT$. Заметим, что в пакете *plots* реализована процедура *textplot*, которая также предназначена для вывода текстовых объектов (комментариев, надписей, и т.п.) и выполняется следующим образом

$textplot([xo, yo, text], options)$, здесь xo, yo – координаты точки, начиная с которой размещается текст $text$. Для рассматриваемых структур синтаксис задания опций предусматривает режим прописных букв.

Рассмотрим пример использования структур.

На рисунке 3.3 параметры по умолчанию были изменены, так с помощью

```
> PLOT(CURVES([[0, 0], [.5, 1], [1, 1], [.75, .5], [.28,
  .5], [0.5e-1, 0]]), THICKNESS(2), FONT(symbol, 16),
  AXESSTYLE(BOX), AXESTICKS(DEFAULT, [0 = `0`, .5 =
  `1/2`, 1 = `1`]), _AXIS[1](_GRIDLINES(10, 10)), _AXIS
  [2](_GRIDLINES(10, 10)), COLOR(HUE, .6));
```

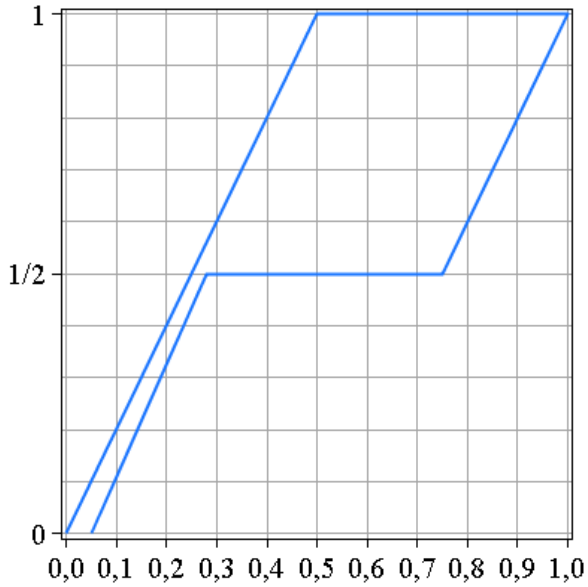


Рис. 3.3: Пример создания графического объекта как структуры

$THICKNESS(2)$ – увеличена толщина линии; $FONT(symbol, 16)$ – выбрано имя шрифта и размер; $AXESSTYLE(BOX)$ – графический объект был заключен в рамку; $AXESTICKS(DEFAULT, [0 = `0`, .5 = `1/2`, 1 = `1`])$ – была задана разметка осей, первый параметр – по умолчанию совпадает с количеством узлов сетки вдоль оси, в данном случае Ox; $\_AXIS[1](\_GRIDLINES(1, 5))$, $\_GRIDLINES$ – вторым параметром задано количество узлов сетки, нанесенной на первую ось (Ox); $COLOR(HUE, .6))$ – выбран цвет сетки.

Все возможные свойства приводятся в справке пакета, из командной строки $?plot$ и далее по ссылке справки $plot/details \rightarrow plot/structure$.

Мы рассмотрели случаи визуализации отдельных графических объектов или нескольких графиков, заданных списком. Для размещения в одном графическом окне нескольких графических объектов, в том числе

текстовых, можно воспользовавшись командой *display* пакета *plots*.

В этом случае требуется каждому создаваемому графическому объекту присваивать имя; в результате этого создается дескриптор–описатель всех свойств объекта, но сам объект при этом не отображается.

> **p:=plot(x*sin(1/x)): t:=textplot([1,1,function]):**

> **with(plots): display([p,t], options)**.

Для того, чтобы построить двумерную область, заданную системой неравенств $f_1(x,y) > c_1, f_2(x,y) > c_2, ..., f_n(x,y) > c_n$, необходимо использовать команду *inequal* из пакета *plots*. В команде **inequals({f1(x,y)>c1,. . . x=x1. . . x2, y=y1..y2, options)** в фигурных скобках указывается система неравенств, определяющих область, затем размеры координатных осей и параметры. Параметры регулируют цвета открытых и закрытых границ, цвета внешней и внутренней областей, а также толщину линий границ:

по команде $optionsfeasible = (color = red)$ – происходит установка цвета внутренней области;

$optionsexcluded = (color = yellow)$ – установка цвета внешней области;

$optionsopen(color = blue, thickness = 2)$ – установка цвета и толщины линии открытой границы;

$optionsclosed(color = green, thickness = 3)$ – установка цвета и толщины линии закрытой границы.

Обобщая возможности визуализации пакета, рассмотрим примеры

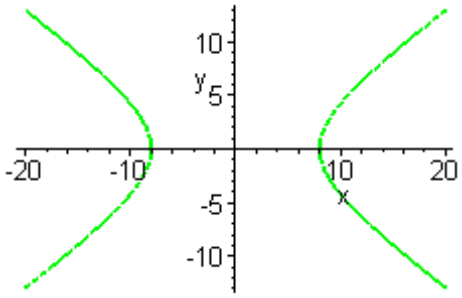
Пример

Построение графика неявной функции (гиперболы) $\frac{x^2}{4} - \frac{y^2}{2} = 16$ осуществляется в результате выполнения команд:

> **with(plots):**

> **implicitplot(x 2/4-y 2/2=16, x=-20..20, y=-16..16,**

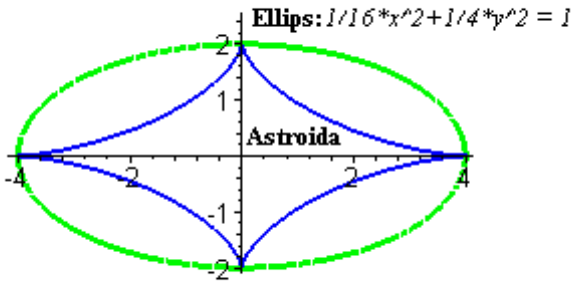
color=green, thickness=2);



Пример

Построение на одном рисунке графика астроиды $x = 4\cos^3 t$, $x = 2\sin^3 t$ $(0 \le t \le 2\pi)$ вписанного в эллипс $\dfrac{x^2}{16} + \dfrac{y^2}{4} = 1$ с названием линий Astroida и Ellips (жирным шрифтом), а также выражения уравнения, набранного курсивом достигается следующими командами

```
> with(plots):
> eq:=x 2/16+y 2/4=1:
> el:=implicitplot(eq, x=-4..4, y=-2..2, scaling=CONSTRAINED,
color=green, thickness=3):
```



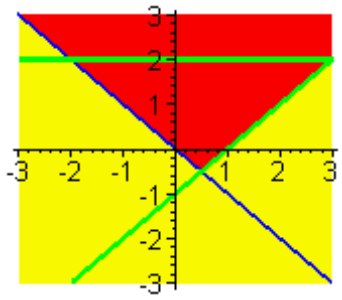

```
> as:=plot([4*cos(t) 3,2*sin(t) 3, t=0..2*Pi],
color=blue, scaling=CONSTRAINED, thickness=2):
> eq1:=convert(eq,string):
> t1:=textplot([1.5,2.5,eq1], font=[TIMES,
ITALIC, 10], align=RIGHT):
```

```
> t2:=textplot([0.2,2.5,"Ellips:"], font=[TIMES,
BOLD,10], align=RIGHT):
> t3:=textplot([1.8,0.4,Astroida], font=[TIMES,
BOLD,10], align=LEFT):
> display([as,el,t1,t2,t3]);
```

Пример

Построение области, ограниченной линиями $x+y>0$, $x-y\le 1$, $y=2$, осуществляется:

```
> with(plots):
```




```
> inequal({x+y>0, x-y<=1, y=2}, x=-3..3, y=-3..3,
optionsfeasible=(color=red),
optionsopen=(color=blue,thickness=2),
optionsclosed=(color=green, thickness=3),
optionsexcluded=(color=yellow) ;
```

здесь открытые-закрытые границы различаются цветом.

3 Лекция 10. Трехмерные графики. Анимация

Поверхность, выражаемую функцией $z = f(x, y)$, графически можно представить с помощью команды

plot3d(f(x,y), x=x1...x2, y=y1...y2,options).

Ее параметры частично совпадают с параметрами команды plot. К часто используемым параметрам команды $plot3d$ относится $light = [angl1, angl2, c1, c2, c3]$ – задание подсветки поверхности, создаваемой источником света из точки со сферическими координатами $angl1$, $angl2$. Цвет определяется долями красного $c1$, зеленого $c2$ и синего $c3$ цветов, которые находятся в интервале [0,1]. Параметр $style = opt$ задает стиль рисунка: $POINT$ – точки, $LINE$ – линии, $HIDDEN$ – сетка с удалением невидимых линий, $PATCH$ – заполнитель (установлен по умолчанию), $WIREFRAME$ – сетка с выводом невидимых линий, $CONTOUR$ – линии уровня, $PATCHCONTOUR$ – заполнитель и линии уровня. Параметр $shading = opt$ задает функцию интенсивности заполнителя, его значение равно xyz – по умолчанию, $NONE$ – без раскраски.

Чтобы построить поверхность, *заданную параметрически* $x = x(u,v)$, $y = y(u,v)$, $z = z(u,v)$, необходимо первым аргументом команды задать список, содержащий параметрические представления координат,

plot3d([x(u,v), y(u,v), z(u,v)], u=u1..u2, v=v1..v2)

и задать интервалы изменения параметров. Трехмерный график *поверхности, заданной неявно* уравнением $F(x, y, z) = c$, строится с помощью команды пакета

plots: implicitplot3d(F(x,y,z)=c, x=x1..x2, y=y1..y2, z=z1..z2),

где указывается уравнение поверхности $F(x, y, z) = c$ и размеры рисунка по координатным осям.

График пространственных кривых. В пакете $plots$ имеется команда $spacecurve$ для построения пространственной кривой, заданной параметрически:

$x = x(t), y = y(t), z = z(t)$. Параметры команды:

> **spacecurve([x(t),y(t),z(t)],t=t1..t2)**,

здесь переменная t изменяется от $t1$ до $t2$.

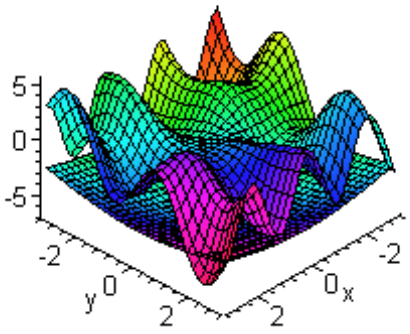
Анимация. Maple позволяет выводить на экран движущиеся изображения с помощью команд *animate* (двумерные) и *animate3d* (трехмерные) из пакета *plots*. Среди параметров команды *animate3d* обязательный параметр $frames$ – число кадров анимации (по умолчанию $frames = 8$).

Трехмерные изображения удобнее настраивать не при помощи опций команды *plot3d*, а используя контекстное меню программы. Для этого следует щелкнуть правой кнопкой мыши по изображению. Тогда появится контекстное меню настройки изображения. Команды этого меню позволяют изменять цвет изображения, режимы подсветки, устанавливать нужный тип осей, тип линий и управлять движущимся изображением.

Пример

Построения двух поверхностей $z = x \sin 2y + y \cos 3x$ и $z = \sqrt{x^2 + y^2} - 7$ в пределах $(x, y) \in [-\pi, \pi]$. Цвет поверхностей пропорционален $x + y$.

> **plot3d({x*sin(2*y)+y*cos(3*x), sqrt(x 2+y 2)-7}, x=-Pi..Pi, y=-Pi..Pi, grid=[30,30], axes=FRAMED, color=x+y);**

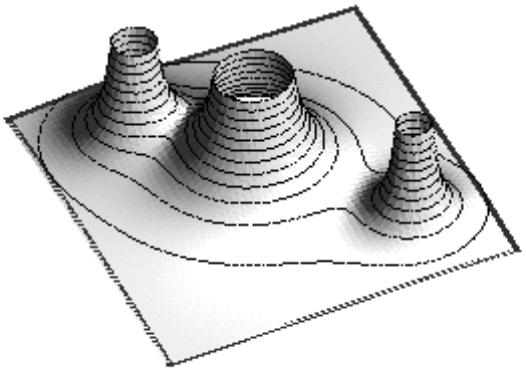


Пример

Построения двух поверхности $z = \dfrac{1}{x^2 + y^2} + \dfrac{0,2}{(x + 1,2)^2 + (y - 1,5)^2} +$

$\frac{0,3}{(x-0,9)^2+(y+1,1)^2}$ вместе с линиями уровня:

> **plot3d(1/(x 2+y 2)+0.2/((x+1.2) 2+(y-1.5) 2)+ 0.3/((x-0.9) 2+(y+1.1) 2), x=-2..2, y=-2..2.5, view=[-2..2, -2..2.5, 0..6], grid=[60,60], shading=NONE, light=[100,30,1,1,1], axes=NONE, orientation=[65,20], style=PATCHCONTOUR);**



Пример

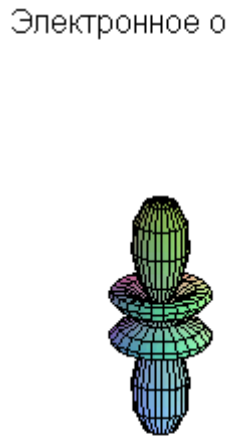
Построения примерной формы электронного облака атома. Форма электронного облака определяется двумя квантовыми числами: число l – определяет тип орбитали, число m – определяет магнитный момент электрона. При m=0 форма электронного облака задается полиномами Лежандра первого рода: $P(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n$. Следует построить параметрически заданную поверхность: $x(\theta,\varphi) = Y(\varphi)\sin\varphi\cos\theta$, $y(\theta,\varphi) = Y(\varphi)\sin\varphi\sin\theta$, $z(\theta,\varphi) = Y(\varphi)\cos\varphi$, где $Y(\varphi) = \left|\sqrt{\frac{2l+1}{4\pi}}P(\cos\varphi)\right|$; (положить l=3), и тогда в результате

> **l:=3:**

> **P:=(x,n)->1/(2 n*n!)*diff((x 2-1) n,x$n);**

```
>Y:=(phi)->abs(sqrt((2*l+1)/(4*Pi))*
subs(x=cos(phi),P(x,l)));
> X0:=Y(phi)*sin(phi)*cos(theta);
> Y0:=Y(phi)*sin(phi)*sin(theta);
> Z0:=Y(phi)*cos(phi);
> plot3d([X0,Y0,Z0],phi=0..Pi,theta=0..2*Pi,
scaling=CONSTRAINED, title="Электронное облако");
```

получим графический аналог команд.



Пример

Движущегося объекта.

```
> animate3d(cos(t*x)*sin(t*y), x=-Pi..Pi,
y=-Pi..Pi, t=1..2);
```

Щелкните по появившемуся изображению правой кнопкой мыши. В

появившемся контекстном меню выполните команду Animation→Continuous. Затем снова вызовите контекстное меню и выполните команду Animation→Play. Для того, чтобы остановить движение, выполните команду Animation→Stop. Затем с помощью мыши поверните рисунок под другим углом и сделайте его вновь движущимся.

4 *Вопросы для самопроверки и задания*

1. **В результате выполнения следующих кодов**

$$> with(plots) :$$
$$> X := seq(0.25 * i, i = 1..16);$$
$$> Y := seq(.25 * X[i]^2, i = 1..16)$$
$$> XY := seq([X[i], Y[i]], i = 1..16);$$
$$> r := pointplot([XY]);$$

   происходит

   (a) визуализация точечного графика;

   (b) создание графической структуры с визуализацией графика;

   (c) создание графической структуры без отображения графика в графическом окне;

2. $f1, f2, f3$ **- имена графических объектов** $\rightarrow display(f1, f2, f3)$

   (a) формирует дескриптор "ансамбля" графических объектов

   (b) выводит на экран их графический аналог

   (c) создает анимацию из объектов $f1, f2, f3$

3. **Команда** *implicitplot*

(a) предназначена для построения функций, заданных параметрически;

(b) предназначена для построения функций, заданных неявно;

(c) предназначена для построения функций, заданных явно;

Глава 4

Обыкновенные дифференциальные уравнения

Инструментарий решения обыкновенных дифференциальных уравнений и их систем востребован при выполнении учебно-научных, а также инженерных расчетов. В Maple для решения таких задач используются не только функции и процедуры стандартной библиотеки, но и модулей специально предназначенных для этого.

В предлагаемом разделе будут рассмотрены возможности решения задачи Коши, краевых задач, решения дифференциальных уравнений, имеющих и не имеющих аналитического решения. Будут рассмотрены функции графического исследования поведения решения и его устойчивости.

1 Лекция 11. Решение дифференциальных уравнений

Для нахождения аналитических решений дифференциальных уравнений в *Maple* применяется команда

dsolve(eq,var,options), где eq – дифференциальное уравнение, var – неизвестные функции, $options$ – параметры. Параметры призваны выбирать метод решения задачи, например, по умолчанию ищется точное аналитическое решение: $type = exact$.

При конструировании дифференциальных уравнений в синтаксисе Maple для обозначения производной применяется команда $diff$, а именно, дифференциальное уравнение $y'' + y = x$ записывается в виде: $diff(y(x), x\$2) + y(x) = x$.

Общее решение дифференциального уравнения зависит от произвольных постоянных, число которых равно порядку дифференциального уравнения. В *Maple* такие постоянные, как правило, обозначаются как $\_C1$, $\_C2$, и т.д.

Общее решение неоднородного линейного дифференциального уравнения всегда выводится так, чтобы была четко видна, структура этого решения. Как известно, общее решение неоднородного линейного дифференциального уравнения равно сумме общего решения соответствующего однородного дифференциального уравнения и частного решения этого же неоднородного дифференциального уравнения. Поэтому в строке вывода решение неоднородного линейного дифференциального уравнения состоит из слагаемых, которые содержат произвольные постоянные (это общее решения соответствующего однородного дифференциального уравнения), и слагаемых без произвольных постоянных (это частное решения этого же неоднородного дифференциального уравнения).

Команда *dsolve* выдает решение дифференциального уравнения в невычисляемом формате. Для того, чтобы с решением можно было работать (например, построить график решения) следует отделить правую часть полученного решения командой $rhs(\%)$.

Пример.

Требуется найти общее решение дифференциального уравнения второго порядка $y"+k^2y=\sin(qx)$ в случае $q \neq k$ и резонанса – $q = k$. Общее решение получается в соответствии со следующими кодами:

> **restart; de:=diff(y(x),x$2)+k^2*y(x)=sin(q*x);**

$$de := \left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right) + k^2\mathrm{y}(x) = \sin(qx)$$

> **dsolve(deq,y(x));**

$$\mathrm{y}(x) = \frac{\left(-\frac{1}{2}\frac{\cos((k+q)x)}{k+q} + \frac{1}{2}\frac{\cos((k-q)x)}{k-q}\right)\sin(kx)}{k} - \frac{\left(\frac{1}{2}\frac{\sin((k-q)x)}{k-q} - \frac{1}{2}\frac{\sin((k+q)x)}{k+q}\right)\cos(kx)}{k} + \\ + \_C1\sin(kx) + \_C2\cos(kx)$$

Отыщем найдем решение в случае резонанса. Для этого перед вызовом команды *dsolve* следует приравнять $q = k$.

> **q:=k: dsolve(de,y(x));**

$$\mathrm{y}(x) = -\frac{1}{2}\frac{\cos(kx)^2\sin(kx)}{k^2} - \frac{\left(-\frac{1}{2}\cos(kx)\sin(kx) + \frac{1}{2}kx\right)\cos(kx)}{k^2} + \\ \_C1\sin(kx) + \_C2\cos(kx)$$

Заметим, что в обоих случаях частное решение неоднородного уравнения и общее решение, содержащее произвольные постоянные, выводятся отдельными слагаемыми.

Фундаментальная (базисная) система решений отыскивается этой же командой *dsolve*. Для того, чтобы отыскать фундаментальную систему решений (базисные функции) дифференциального уравнения следует указать опцию $output = basis$, определяющую режим поиска ФСР.

Рассмотрим

Пример.

Отыскания ФСР для дифференциального уравнения:

$y^{(4)}+2y''+y=0$.

> **de:=diff(y(x),x$4)+2*diff(y(x),x$2)+y(x)=0;**

$$de := \left(\frac{\partial^4}{\partial x^4}\mathrm{y}(x)\right) + 2\left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right) + y(x) = 0$$

> **dsolve(de, y(x), output=basis);**

$$[\cos(x), \sin(x), x\cos(x), x\sin(x)]$$

Задача Коши и краевые задачи

Команда *dsolve* может найти решение задачи Коши или краевой задачи, если помимо дифференциального уравнения задать начальные или краевые условия для неизвестной функции. Для обозначения производных в начальных или краевых условиях используется дифференциальный оператор D, например, условие y''(0)=2 следует записать в виде $(D@@2)(y)(0) = 2$, или условие $y'(1)=0$: $D(y)(1) = 0$. Напомним, что производная n-го порядка записывается как композиция n–кратного дифференцирования в виде $(D@@n)(y)$.

Пример.

Найти решение задачи Коши: д.у. $y^{(4)} + y''=2\cos x$, с начальными условиями $y(0)=-2$, $y'(0)=1$, $y''(0)=0$, $y'''(0)=0$. Дифференциальное уравнение в Maple имеет вид

> **de:=diff(y(x),x$4)+diff(y(x),x$2)=2*cos(x);**

$$de := \left(\frac{\partial^4}{\partial x^4}\mathrm{y}(x)\right) + \left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right) = 2\cos(x)$$

и начальные условия.

> **cond:=y(0)=-2, D(y)(0)=1, (D@@2)(y)(0)=0, (D@@3)(y)(0)=0;**

$$cond:=\mathrm{y}(0)=-2,\ \mathrm{D}(\mathrm{y})(0)=1,\ (\mathrm{D}^{(2)})(\mathrm{y})(0)=0,\ (\mathrm{D}^{(3)})(\mathrm{y})(0)=0$$

> **dsolve({de,cond},y(x));**

$$\mathrm{y}(x)=-2\cos(x)-x\sin(x)+x$$

Моделирование в Maple решения краевой задачи прояснит следующий пример:

Пример.

Найти решение краевой задачи: $y''+y=2x-\pi$, с граничными условиями $y(0)=0$, $y\left(\frac{\pi}{2}\right)=0$. Построить график решения.

> **restart; de:=diff(y(x),x$2)+y(x)=2*x-Pi;**

$$de:=\left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right)+\mathrm{y}(x)=2x-\pi$$

> **cond:=y(0)=0,y(Pi/2)=0;**

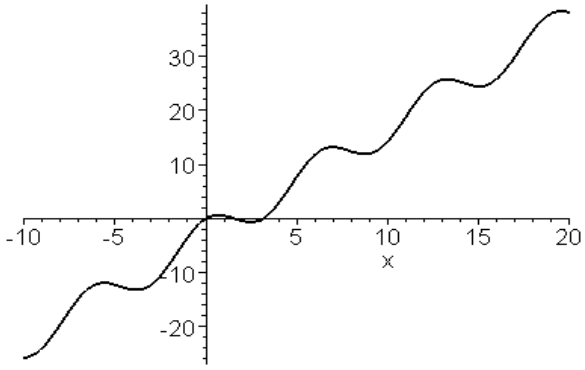
$$cond:=y(0)=0,\quad y\left(\frac{\pi}{2}\right)=0$$

> **dsolve({de,cond},y(x));**

$$\mathrm{y}(x)=2x-\pi+\pi\cos(x)$$

Замечание: один из приемов построения графика решения ДУ — отделить правую часть полученного выражения.

> **y1:=rhs(%):plot(y1,x=-10..20,thickness=2);**



Для решения *системы дифференциальных уравнений* используется та же команда *dsolve* в виде:

dsolve({sys},{x(t),y(t),...}),

здесь sys – система дифференциальных уравнений, $\{x(t), y(t), \ldots\}$ – набор неизвестных функций. Нахождение решения системы дифференциальных уравнений предлагает следующий пример.

Пример. Пусть задана система ОДУ

$$\begin{cases} x' = -4x - 2y + \dfrac{2}{e^t - 1}, \\ y' = 6x + 3y - \dfrac{3}{e^t - 1} \end{cases}$$

в Maple она задается следующим образом:

> **sys:=diff(x(t),t)=-4*x(t)-2*y(t)+2/(exp(t)-1),**
diff(y(t),t)=6*x(t)+3*y(t)-3/(exp(t)-1):

и решается

> **dsolve({sys},{x(t),y(t)});**

$$\{\mathrm{x}(t) = -3\_C1 + 4C1\_e^{(-t)} - 2C2\_ + 2C2\_e^{(-t)} + 2e^{(-t)}\ln(e^t - 1),$$

$$\mathrm{y}(t) = 6\_C1 - 6\_C1e^{-t} - 3\_C2e^{(-t)} + 4\_C2 - 3e^{(-t)}\ln(e^{(t)} - 1)\}$$

В результате найдены две функции $x(t)$ и $y(t)$, которые зависят от двух произвольных постоянных $\_C1$ и $\_C2$.

2 Лекция 12. Приближенное и численное решение дифференциальных уравнений

Для многих типов дифференциальных уравнений не может быть найдено точное аналитическое решение. В этом случае дифференциальное уравнение можно решить с помощью приближенных методов, и, в частности, с помощью разложения в степенной ряд неизвестной функции.

Чтобы найти приближенное решение дифференциального уравнения в виде степенного ряда, в команде *dsolve* следует задать параметр $type = series$ (или просто *series*). Для того, чтобы указать порядок (степень) разложения n следует перед командой *dsolve* задать определение порядка с помощью команды $Order := n$.

Если ищется общее решение дифференциального уравнения в виде разложения в степенной ряд, то коэффициенты при степенях найденного разложения будут содержать неизвестные значения функции в нуле $y(0)$ и ее производных $D(y)(0), (D@@2)(y)(0)$ и т.д. Полученное в строке вывода выражение будет иметь вид, похожий на разложение искомого решения в ряд Маклорена, но с другими коэффициентами при степенях x. Для выделения частного решения следует задать начальные условия $y(0) = 1, D(y)(0) = 2, (D@@2)(y)(0) = 3$ и т.д., причем количество этих начальных условий должно совпадать с порядком соответствующего дифференциального уравнения.

Разложение в степенной ряд имеет тип *series*, поэтому для дальнейшей работы с этим рядом его следует преобразовать в полином с помощью команды

convert(%,polynom),

а затем выделить правую часть полученного выражения командой $rhs(\%)$.

Пример. Требуется найти решение задачи Коши:

$y' = y + xe^y$, $y(0) = 0$

в виде степенного ряда с точностью до 5-го порядка. Для этого сначала следует задать порядок, а затем воспользоваться *dsolve*

> **restart; Order:=5:**

> **dsolve({diff(y(x),x)=y(x)+x*exp(y(x)),**

y(0)=0}, y(x), type=series);

$$\mathrm{y}(x) = \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{6}x^4 + \mathrm{O}(x^5)$$

В полученном решении слагаемое $\mathrm{O}(x^5)$ означает, что остальные слагаемые ряда не превосходят x^5.

Пример. Требуется отыскать общее решение дифференциального уравнения $y"(x) - y^3(x) = e^{-x}\cos x$, в виде разложения в степенной ряд до 4-го порядка, если выполняются условия: $y(0)=1$, $y'(0)=0$.

> **restart; Order:=4: de:=diff(y(x),x$2)-**

y(x) 3=exp(-x)*cos(x):

> **f:=dsolve(de,y(x),series);**

$$f := \mathrm{y}(x) = \mathrm{y}(0) + \mathrm{D}(y)(0)x + \left(\tfrac{1}{2}\mathrm{y}(0)^3 + \tfrac{1}{2}\right)x^2 +$$
$$\left(\tfrac{1}{2}\mathrm{y}(0)^2\mathrm{D}(y)(0) - \tfrac{1}{6}\right)x^3 + \mathrm{O}(x^4)$$

Замечание: в полученном разложении запись $D(y)(0)$ обозначает производную в нуле: $y'(0)$. Для нахождения частого решения осталось задать начальные условия:

> **y(0):=1: D(y)(0):=0:f;**

$$\mathrm{y}(x) = 1 + x^2 - \frac{1}{6}x^3 + \mathrm{O}(x^4)$$

Пример. Требуется найти приближенное решение в виде степенного ряда до 6-го порядка и точное решение задачи Коши: $y''' - y' = 3(2 - x^2)\sin x$,

$y(0) = 1, y'(0) = 1, y''(0) = 1$, а также построить на одном рисунке графики точного и приближенного решений.

Предлагаемые коды достигают поставленной цели.

```
> restart; Order:=6:
> de:=diff(y(x),x$3)-diff(y(x),x)=
3*(2-x 2)*sin(x);
```

$$de := \left(\frac{\partial^3}{\partial x^3}\mathrm{y}(x)\right) - \left(\frac{\partial}{\partial x}\mathrm{y}(x)\right) = 3(2 - x^2)\sin(x)$$

```
> cond:=y(0)=1, D(y)(0)=1, (D@@2)(y)(0)=1;
```

$$cond := \mathrm{y}(0)=1,\ \mathrm{D}(\mathrm{y})(0)=1,\ \mathrm{D}^{(2)}(\mathrm{y})(0)=1$$

```
> dsolve({de,cond},y(x));
```

$$\mathrm{y}(x) = \frac{21}{2}\cos(x) - \frac{3}{2}x^2\cos(x) + 6x\sin(x) - 12 + \frac{7}{4}\mathrm{e}^{x} + \frac{3}{4}\mathrm{e}^{(-x)}$$

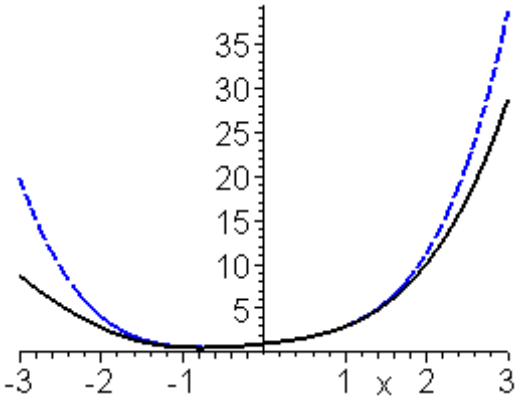
```
> y1:=rhs(%):
> dsolve({de,cond},y(x), series);
```

$$\mathrm{y}(x) = 1 + x + \tfrac{1}{2}x^2 + \tfrac{1}{6}x^3 + \tfrac{7}{24}x^4 + \tfrac{1}{120}x^5 + \mathrm{O}(x^6)$$

Замечание: выбор метода определения решения дифференциального уравнения в виде ряда определяется *series*, поэтому для дальнейшего использования такого решения (вычислений или построения графика) его следует преобразовать в полином с помощью команды *convert*

```
> convert(%,polynom): y2:=rhs(%):
> p1:=plot(y1,x=-3..3,thickness=2,color=black):
> p2:=plot(y2,x=-3..3, linestyle=3,thickness=2,
color=blue):
> with(plots): display(p1,p2);
```

На этом рисунке видно, что наилучшее приближение точного решения степенным рядом достигается на интервале -1<x<1.



Численное решение дифференциальных уравнений находится с помощью той же команды *dsolve*. Однако построения графиков решений дифференциальных уравнений можно достичь не только рассмотренным ранее способом, но и с помощью команды *odeplot*.

Для того, чтобы найти численное решение дифференциального уравнения (задачи Коши или краевой задачи) в команде *dsolve* следует указать параметр $type = numeric$ (или просто $numeric$). Тогда команда решения дифференциального уравнения будет иметь вид

dsolve(eq, vars, type=numeric, options), здесь eq – уравнения; $vars$ – список неизвестных функций; $options$ – параметры-опции выбора метода численного интегрирования дифференциального уравнения. В *Maple* реализованы такие методы: $method = rkf45$ – метод Рунге-Кутта-Фельберга 4-5-ого порядка (установлен по умолчанию); $method = dverk78$ – метод Рунге-Кутта 7-8 порядка; $method = classical$ – классический метод Рунге-Кутта 3-его порядка; $method = gear$ и $method = mgear$ – одношаговый и многошаговый методы Гира.

График численного решения дифференциального уравнения можно построить с помощью команды

odeplot(dd, [x,y(x)], x=x1..x2), где в качестве функции используется команда

dd:=dsolve({eq,cond}, y(x), numeric) – численного решения, далее в *odeplot* указывают список из переменной и неизвестной функции $[x, y(x)]$, а также интервал $x = x1..x2$, на котором строится график.

Пример.

Требуется найти численное и приближенное решение задачи Коши в виде степенного ряда до 6-ого порядка: $y'' - x\sin(y) = \sin 2x$, $y(0) = 0$, $y'(0) = 1$.

Сначала найдем численное решение задачи Коши и построим его график.

> **restart; Order=6:**

> **eq:=diff(y(x),x$2)-x*sin(y(x))=sin(2*x):**

> **cond:=y(0)=0, D(y)(0)=1:**

> **de:=dsolve({eq,cond},y(x),numeric);**

$$de{:=}\mathbf{proc}(rkf45\_x)...\mathbf{end}$$

Замечание: в строке вывода появляется сообщение о том, что при решении использован метод *rkf45*.

Во избежание вывода промежуточных результатов в отлаженных кодах (в том числе процедурах пользователя) рекомендуется подавлять вывод двоеточием.

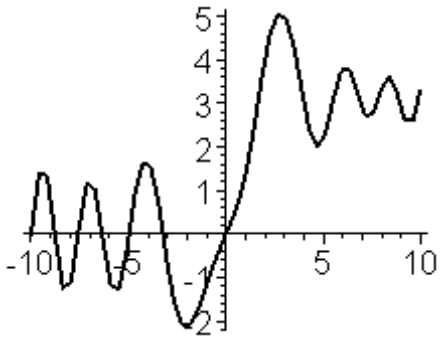
Если необходимо получить решения при каком-то фиксированном значении переменной x (заодно будет выведено значение производной решения в этой точке), например, при =0.5, то следует набрать:

> **de(0.5);**

$$\left[x = .5,\ \mathrm{y}(x) = .5449261153862630,\ \frac{\partial}{\partial \mathrm{x}}\,\mathrm{y}(x) = 1.272503082225380\right]$$

> **with(plots):**

> **odeplot(de,[x,y(x)],-10..10,thickness=2);**



Теперь найдем приближенное решение задачи Коши в виде степенного ряда и построим графики численного решения, а также полученного численного решения (в рядах) в интервале их наилучшего совпадения.

> **dsolve({eq, cond}, y(x), series)**

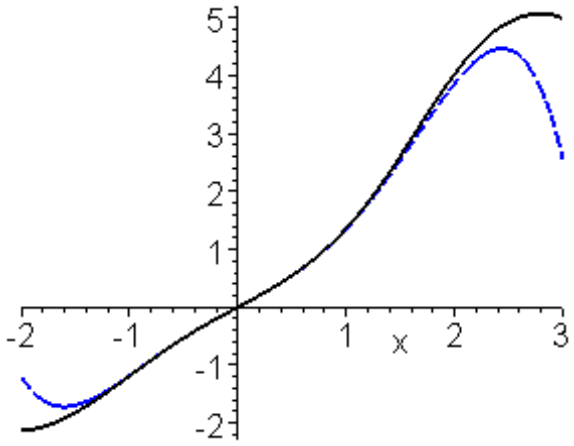
$$\mathrm{y}(x) = x + \frac{1}{3}x^3 + \frac{1}{12}x^4 - \frac{1}{15}x^5 + \mathrm{O}(x^6)$$

> **convert(%, polynom):p:=rhs(%):**

> **p1:=odeplot(de,[x,y(x)],-2..3, thickness=2,**
color=black):

> **p2:=plot(p,x=-2..3,thickness=2,linestyle=3,**
color=blue):

> **display(p1,p2);**



Интервал наилучшего приближение решения степенным рядом легко идентифицировать из рисунка.

Пример.

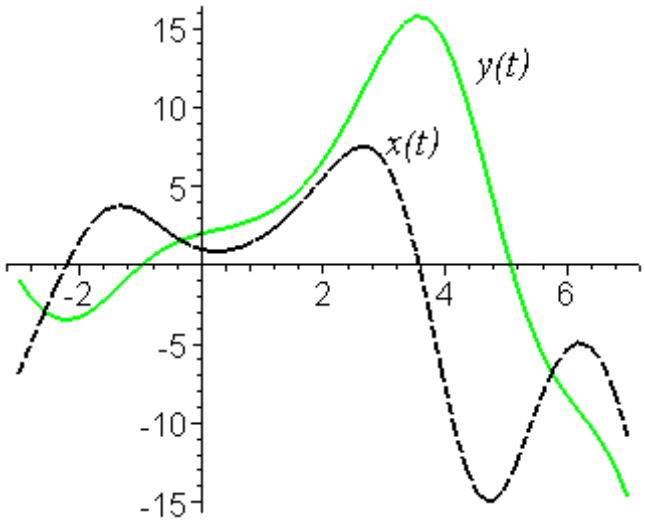
Построить графики решений задачи Коши системы дифференциальных уравнений:

$x'(t)=2y(t)\sin(t) - x(t) - t$,

$y'(t) = x(t)$,

$x(0)=1$, $y(0)=2$.

Графическое решение



получено в результате выполнения следующих кодов:

```
> restart; cond:=x(0)=1,y(0)=2:
> sys:=diff(x(t),t)=2*y(t)*sin(t)-x(t)-t,
diff(y(t),t)=x(t):
> F:=dsolve({sys,cond},[x(t),y(t)],numeric):
> with(plots):
> p1:=odeplot(F,[t,x(t)],-3..7, color=black,
thickness=2,linestyle=3):
> p2:=odeplot(F,[t,y(t)],-3..7,color=green,
thickness=2):
> p3:=textplot([3.5,8,"x(t)"], font=[TIMES,
```

ITALIC, 12]):

> **p4:=textplot([5,13,"y(t)"], font=[TIMES,**

ITALIC, 12]):

В результате созданы структуры графических объектов, в том числе текстовых; они описывают их свойства, некоторые из них (цвет, размер, имя шрифта) изменены, остальные заданы по умолчанию.

Следующая команда отобразит графические объекты, заданные списком.

> **display(p1,p2,p3,p4);**

3 Лекция 13. Пакет графического представления решений Detools

Для численного решения задачи Коши, построения графиков решения и фазовых портретов в *Maple* имеется специальный пакет *DEtools*.

Команда **DEplot** из пакета *DEtools* строит графики решения или фазовые портреты. Она аналогична команде *odeplot*, но более функциональна. Главное отличие *odeplot* заключается в том, что она находит численное решение дифференциального уравнения и строит графики. Основные параметры *DEplot* совпадают параметрами *odeplot*:

DEplot(de, vars, range, x=x1..x2, y=y1..y2, cond, options), здесь de – дифференциальное уравнение или система дифференциальных уравнений; $vars$ – список неизвестных функций; $range$ – диапазон измерения независимой переменной; $cond$ – начальные условия; $x = 1..2$ и $y = 1..2$ – диапазоны изменения функций; $options$ – дополнительные параметры.

Приведем примеры варьирования параметров: $linecolor$=цвет линии; $scene = [x, y]$ – определяет переменные, подлежащие визуализации; **iterations**=число итераций, необходимое для повышения точности вычислений (по умолчанию это число равно 1); **stepsize=number** – определяет величину щага между точками графика, по умолчанию оно равно ($x2 - x1)/20$; $obsrange=true/false$ – прерывать или нет вычисления, если график решения выходит за область визуализации.

Для решения дифференциального уравнения n-ого порядка начальные условия можно задавать в более компактной форме:

$[x_0, y_0, y_0', y_0'', \ldots]$, где x_0, y_0 – начальные условия, y_0', y_0'', ... – значения всех производных в точке x_0 вплоть до $(n-1)$-ого порядка. Для освоения этих команд рассмотрим

Пример.

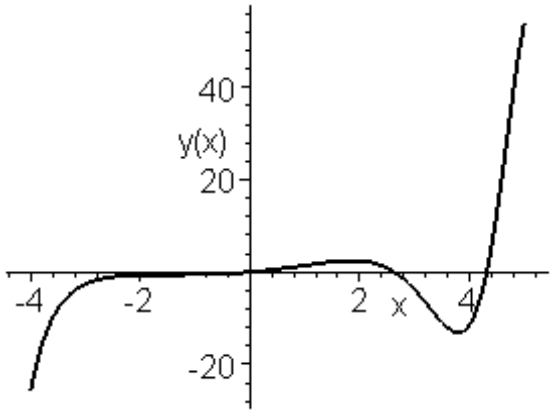
Построения графика решения дифференциального уравнения:

$$y''' + x\sqrt{|y'|} + x^2 y = 0, \quad y(0) = 0, \quad y'(0) = 1, \quad y''(0) = 1$$

для $x \in [-4, 5]$.

с помощью кодов:

```
> restart; with(DEtools):
> DEplot(diff(y(x),x$3)+x*sqrt(abs(diff(y(x),x)))
+x 2*y(x)=0, {y(x)}, =-4..5, [[y(0)=0,D(y)(0)=1,
(D@@2)(y)(0)=1]], stepsize=.1, linecolor=black,
thickness=2);
```



Здесь представлен график решения, точки выбирались с шагом 0.1; линии задана двойная толщина.

Построение фазовых портретов уравнений высоких порядков и систем дифференциальных уравнений *DEplot* осуществляется командой *DEplot*.

Так с помощью команды *DEplot* можно построить фазовый портрет в плоскости (x, y), для системы двух дифференциальных уравнений:

$\frac{dx}{dt} = f(x, y, t), \quad \frac{dy}{dt} = g(x, y, t)$, если также в параметрах данной команды указать $scene = [x, y]$.

Для автономной системы дифференциальных уравнений на фазовом

портрете будет построено поле скоростей в виде векторов. Их размер регулируется параметром $arrows$=$SMALL$, $MEDIUM$, $LARGE$, $LINE$ или $NONE$.

Чтобы отобразить весь фазовый портрет, необходимо для каждой фазовой траектории указывать начальные условия: например, для системы двух дифференциальных уравнений первого порядка несколько начальных условий в команде $DEplots$ после задания диапазона изменения независимой переменной **t** указываются следующие параметры:

$[[x(0) = x_1, y(0) = y_1], [x(0) = x_2, y(0) = y_2], \ldots, [x(0) = x_n, y(0) = y_n]]$.

Начальные условия можно задавать в более компактной форме: $[t_0, x_0, y_0]$, где t_0 – точка, в которой задаются начальные условия, x_0 и y_0 – значения искомых функций в точке t_0.

Фазовый портрет системы двух дифференциальных уравнений первого порядка можно также построить с помощью команды:

phaseportrait(sys, [x,y],x1..x2,[[cond]]), здесь sys – система двух дифференциальных уравнений первого порядка; $[x, y]$ – имена искомых функций; $x1..x2$ – интервал, на котором следует построить фазовый портрет, а в скобках указываются начальные условия. Эта команда находится в пакете $DEtools$, поэтому данный пакет должен быть предварительно загружен. Последовательность команд построения фазового портрета демонстрирует

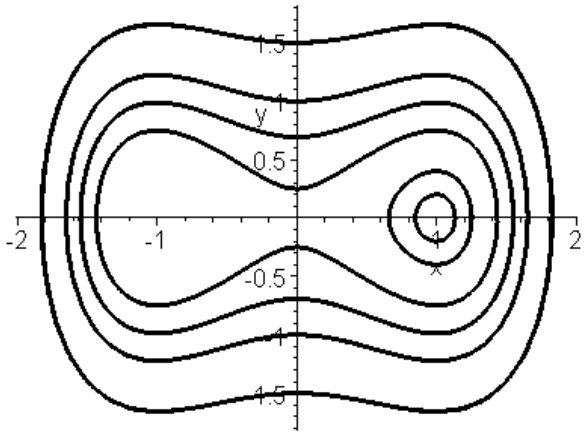
Пример.

Пусть задана система дифференциальных уравнений

$$\begin{cases} x' = y \\ y' = x - x^3 \end{cases}$$

для нескольких наборов начальных условий: $x(0)$=1, $y(0)$=0.2; $x(0)$=0, $y(0)$=1; $x(0)$=1, $y(0)$=0.4; $x(0)$=1, $y(0)$=0.75; $x(0)$=0, $y(0)$=1.5; $x(0)$=-0.1, $y(0)$=0.7.

```
> restart; with(DEtools):
> DEplot({diff(x(t),t)=y, diff(y(t),t)=x-x^3},
[x(t),y(t)], t=0..20, [[0,1,0.2], [0,0,1],
[0,1,0.4], [0,1,0.75], [0,0,1.5], [0,-0.1,0.7]],
stepsize=0.1, arrows=none, linecolor=black);
```



Пример.

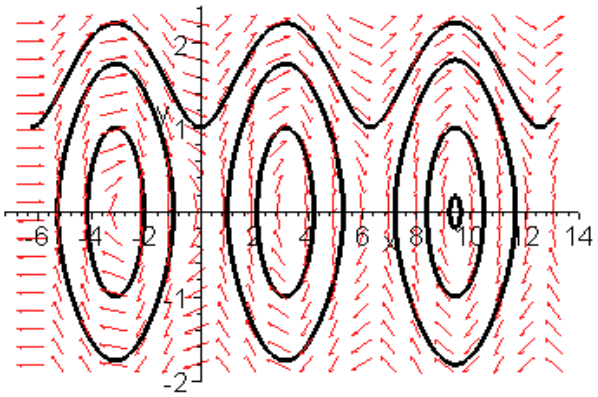
Построения фазового портрета и поля скоростей автономной системы

$$\begin{cases} x' = y \\ y' = \sin x \end{cases}$$

для различных начальных условий $x(0)$=1, $y(0)$=0; $x(0)$=-1, $y(0)$=0; $x(0)$=π, $y(0)$=1; $x(0)$=-π, $y(0)$=1; $x(0)$=3π, $y(0)$=0.2; $x(0)$=3π, $y(0)$=1; $x(0)$=3π, $y(0)$=1.8; $x(0)$=-2π, $y(0)$=1.

```
> restart; with(DEtools):
> sys:=diff(x(t),t)=y, diff(y(t),t)=sin(x):
> DEplot({sys},[x(t),y(t)], t=0..4*Pi, [[0,1,0],
[0,-1,0], [0,Pi,1], [0,-Pi,1], [0,3*Pi,0.2],
[0,3*Pi,1], [0,3*Pi,1.8], [0,-2*Pi,1]],
stepsize=0.1, linecolor=black);
```

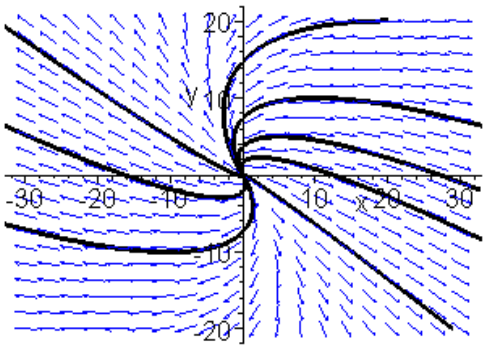
Пример.



Построения фазового портрета системы дифференциальных уравнений:

$$\begin{cases} x' = 3x + y \\ y' = y - x \end{cases}$$

Начальные условия, диапазон изменения переменной и размеры координатных осей подбираются самостоятельно из соображений наглядности фазового портрета.




```
> restart; with(DEtools):
> sys:=diff(x(t),t)=3*x+y, diff(y(t),t)=-x+y:
> phaseportrait([sys],[x(t),y(t)],t=-10..10,
[[0,1,-2], [0,-3,-3], [0,-2,4], [0,5,5], [0,5,-3],
[0,-5,2], [0,5,2], [0,-1,2]], x=-30..30,y=-20..20,
stepsize=.1, colour=blue,linecolor=black);
```

Таким образом с помощью команд пакета пользователь может решать достаточно сложные задачи теории дифференциальных уравнений.

4 *Вопросы для самопроверки и задания*

1. **Определить результат выполнения следующих команд:**

$$> x := 1;$$

$$> diff(x^2, x);$$

   (a) 2

   (b) error

   (c) 2x

2. **n-я производная функции f(x),**

   заданной алгебраическим выражением, вычисляется

   (a) Diff(f, x$n);

   (b) diff(f, x$n);

   (c) diff(f, x^ n);

3. **Команда отложенного интегрирования**

   (a) Int

   (b) int

   (c) INT

4. **График решения задачи Коши, реализован ниже**

$$> de := diff(y(x), x) = x + sin(y(x)/7);$$

$$> ic := y(0.5) = 0.6;$$

$$> ds := dsolve(\{de, ic\},\ numeric,\ range = 0.5 \ldots 1.5,$$

$$output = listprocedure);$$

$$> yresult := rhs(ds[2]); \#\text{или } yresult := eval(y(x), ds);$$

$$> with(plots); plot(yresult, 0.5 \ldots 1.5);$$

его можно также получить с помощью

(a) smartplot(ds)

(b) odeplot(ds);

(c) plot(ds)

5. **Чему равно значение последней команды кода?**

$> de := diff(y(x), x) = x + sin(y(x)/7);$

$> ic := y(0.5) = 0.6;$

$> ds := dsolve(\{de, ic\},\ numeric,\ range = 0.5 \ldots 1.5,$

$output = listprocedure);$

$> yresult := rhs(ds[2]);$

$> yresult(0.5);$

(a) ∞

(b) 0.6

(c) 0

6. **Определите результат выполнения последней команды**

$> ode := diff(y(x), x) = F((y(x) - x * ln(x))/x) + ln(x);$

$> type(type(odetest(dsolve(ode, implicit), ode), `numeric`), `boolean`);$

(a) true

(b) false

(c) reset

Глава 5

Использование Maple в теории функций многих переменных

Для исследования функций многих переменных требуется использование операций дифференциального и интегрального исчисления. Для этого пользуются теми же командами, что и для функций одной переменной, только с указанием дополнительных параметров.

1 Лекция 14. Анализ многомерных функций

Для вычисления *частных производных* функции $f(x_1,\ldots,x_m)$ в команде $diff$ используется формат:

diff(f,x1$n1,x2$n2,. . . , xm$nm), здесь $x1,\ldots,xm$ – переменные, по которым производится дифференцирование, а после знака $ указаны соответствующие порядки дифференцирования; так частная производная $\dfrac{\partial^2 f}{\partial x \partial y}$ записывается в виде: $diff(f,x,y)$. Далее приведен

Пример

поиска частных производных $\dfrac{\partial f}{\partial x}$ и $\dfrac{\partial f}{\partial y}$ функции $f(x,y) = \operatorname{arctg}\dfrac{x}{y}$.

> **f:=arctan(x/y):**

D> iff(f,x)=simplify(diff(f,x));

$$\frac{\partial}{\partial x}\arctan\frac{x}{y} = \frac{y}{x^2+y^2}$$

> Diff(f,y)=simplify(diff(f,y));

$$\frac{\partial}{\partial y}\arctan\frac{x}{y} = -\frac{x}{x^2+y^2}.$$

результат достигается приведенными кодами.

Пример

Определения частных производных 2–го порядка функции $f(x,y) = \dfrac{x-y}{x+y}$.

> restart; f:=(x-y)/(x+y):

> Diff(f,x$2)=simplify(diff(f,x$2));

$$\frac{\partial^2}{\partial x^2}\frac{x-y}{x+y} = -4\frac{y}{(x+y)^3}$$

> Diff(f,y$2)=simplify(diff(f,y$2));

$$\frac{\partial^2}{\partial y^2}\frac{x-y}{x+y} = 4\frac{x}{(x+y)^3}$$

> Diff(f,x,y)=diff(f,x,y);

$$\frac{\partial^2}{\partial x\partial y}\frac{x-y}{x+y} = 2\frac{x-y}{(x+y)^3}.$$

Для нахождения *локальных и условных экстремумов* функции многих переменных используется команда из стандартной библиотеки

extrema(f,{cond},{x,y,...},'s'), где *cond* – ограничения для поиска условного экстремума, которые записываются в виде равенств. Далее в фигурных скобках указываются все переменные, от которых зависит функция f, а затем s – имя переменной, которой будут присвоены

координаты точек экстремума. При отсутствии ограничений будет производиться поиск локального экстремума.

Команда *extrema* выдает все критические точки, в том числе и те, в которых экстремума нет. Для выявления экстремального поведения требуется дополнительный анализ.

Так же, как и для функции одной переменной, наибольшее и наименьшее значения функции многих переменных вычисляются командами $maximize(f, \{x1, ..$ и $minimize(f, \{x1, \ldots, xn\}, range)$, где f – идентификатор или само выражение; в фигурных скобках список всех аргументов; интервалы изменения каждого аргумента.

Если требуется найти переменные, при которых линейная функция многих переменных имеет максимум (или минимум) при выполнении определенных ограничений, заданных в виде линейных равенств или неравенств, следует использовать *симплекс-метод*. Для этого необходимо загрузить пакет *simplex*, а затем воспользоваться командой *maximize* (или *minimize*), где теперь в качестве *range* можно указывать в фигурных скобках ограничительную систему неравенств. Пакет предназначен для решения задач линейной оптимизации, команды *maximize* и *minimize* меняют свой "характер", в результате выдаются координаты точек, при которых заданная линейная функция имеет максимум или минимум. При этом допускается дополнительная опция, например, для поиска решений $NONNEGATIVE$ в области неотрицательных величин.

Рассмотрим пример

Пример.

Найти экстремумы функции $f(x, y) = 2x^4 + y^4 - x^2 - 2y^2$.

> **restart:**

> **f:=2*x^4 +y^4-x^2-2*y^2:**

> **extrema(f,{},{x,y},'s'):s;**

{{x=0, y=0}, {$x = \frac{1}{2}$ y=0}, {$x = \frac{-1}{2}$, y=0}, {x=0, y=1}, {x=0, y=-1}, {$x = \frac{1}{2}$, y=1}, {$x = \frac{1}{2}$, y=-1}, {$x = \frac{-1}{2}$, y=1}, {$x = \frac{-1}{2}$, y=-1}}

Здесь s – множество множеств критических точек, записанных в виде равенств. Получилось всего два экстремума, поэтому очевидно, что f_{max}=0 и $f_{min} = -9/8$, причем максимум достигается в точке (0,0). Остальные критические точки следует проверить. В силу четности функции по обеим переменным, можно ограничиться проверкой критических точек только с положительными координатами.

> **subs([x=1/2,y=1],f):**

> **subs([x=1/2,y=0],f):**

> **subs([x=0,y=1],f):**

Таким образом, функция имеет следующие локальные экстремумы: $f_{max} = f(0,0)$=0 и $f_{min} = f\left(\pm\frac{1}{2}, \pm 1\right) = f\left(\pm\frac{1}{2}, \mp 1\right) = -9/8$.

Чтобы найти наибольшее и наименьшее значения функции $f(x, y) = x^2 + 2xy - 4x + 8y$ в прямоугольнике x=0, y=0, x=1, y=2, необходимо область представить в виде неравенств: 0<x<1, 0<y<2.

> **restart: readlib(maximize):readlib(minimize):**

> **f:=x^2+2*x*y-4*x+8*y:**

> **maximize(f,{x,y},{x=0..1,y=0..2}):**

> **minimize(f,{x,y},{x=0..1,y=0..2}):**

Таким образом, функция имеет наибольшее значение f_{max}=17 и наименьшее значение $f_{min} = -4$.

Чтобы найти условные экстремумы функции $f(x,y)$=xy+yz при $x^2 + y^2$=2, $y + z$=2, x>0, y>0, z>0, необходимо выполнить следующие команды

> **restart: readlib(extrema): f:=x*y+y*z:**

> **assume(x>0);assume(y>0);assume(z>0);**

> **simplify(extrema(f,{x^2+y^2=2,y+z=2},{x,y,z}, 's'));**

{min($\frac{3}{2}$RootOf(_Z^2+4_Z+1)+$\frac{1}{2}$, 0), max($\frac{3}{2}$RootOf(_Z^2+4_Z+1)+$\frac{1}{2}$, 2)}

Чтобы воспользоваться *maximize*, *extrema* процедуры предварительно можно не загружать из стандартной библиотеки. Команда *simplify* не привела к удовлетворительному результату, тогда как в результате использования *convert* получили оценку экстремумов.

> **convert(%,radical);**

$$\{\min\left(-\tfrac{5}{2}+\tfrac{3}{2}\sqrt{3},\,0\right),\,\max\left(-\tfrac{5}{2}+\tfrac{3}{2}\sqrt{3},\,2\right)\}$$

> **convert(s,radical);**

$$\{\{x\sim=1,z\sim=1,y\sim=1\},\{x\sim=\text{-}1,z\sim=1,y\sim=1\},$$
$$\{x\sim=-\tfrac{1}{2}-\tfrac{1}{2}\sqrt{3},y\sim=-\tfrac{1}{2}(-2+\sqrt{3})(1+\sqrt{3}),\,z\sim=\tfrac{5}{2}-\tfrac{1}{2}\sqrt{3}\}\}$$

В этом случае команда *extrema* определила характер экстремумов, а величина экстремумов может быть вычислена подстановкой:

> **subs(s[1],f):**

> **subs(s[2],f);**

> **subs(s[3],f):convert(%,radical):simplify(%):**

$$-\frac{5}{2}+\frac{3}{2}\sqrt{3}$$

в результате функция имеет следующие локальные экстремумы: $f_{max} = f(1,1,1)=2$ и $f_{min} = f(-1,1,1)=0$; третья критическая точка является седловой.

Пример.

Чтобы определить, при каких значениях переменных функция $f(x,y,z)=\text{-}x+2y+3z$ имеет максимум, для $x+2y-3z \leq 4$, $5x-6y+7z \leq 8$, $9x+10z \leq 11$, $x, y, y > 0$, следует выполнить последовательность команд:

> **restart: with(simplex):**

> **f:=-x+2*y+3*z:**

> **cond:={x+2*y-3*z<=4, 5*x-6*y+7*z<=8, 9*x+10*z<=11}:**

> **maximize(f,cond,NONNEGATIVE);**

$$\{x{=}0,\, y = \tfrac{73}{20},\, z = \tfrac{11}{10}\}$$

2 Лекция 15. Интегрование функций многих переменных

В *Maple* имеются две специальные команды для вычисления двойных и тройных интегралов, содержащиеся в библиотеке *student*.

Для вычисления двойных интегралов $\iint\limits_D f(x,y)dxdy$ используется команда

Doubleint(f(x, y), D), где D – область интегрирования, записываемая в одном из следующих форматов:

$x = x1..2, y = y1..y2$, где числа $1, 2, y1, y2$ задают прямоугольную область интегрирования;

$x = f1(y)..f2(y), y = y1..y2$, где $f1(y), f2(y)$ – линии, ограничивающие область интегрирования слева и справа на интервале от $y1$ до $y2$;

$x = x1..x2, y = g1(x)..g2(x)$, где $g1(y), g2(y)$ – линии, ограничивающие область интегрирования снизу и сверху на интервале от $x1$ до $x2$.

Для вычисления тройных интегралов $\iiint\limits_V f(x,y,z)dxdydz$ используется команда

Tripleint(f(x, y, z),x, y, z, V), где V – область интегрирования.

Обе эти команды являются командами отложенного действия. Чтобы получить значение интеграла, следует использовать команду $value(\%)$.

Кратные интегралы можно вычислять с помощью повторения команды int, например, интеграл $\int\limits_0^2 dy \int\limits_0^1 x^2y^3dx$ вычисляется командой

> **int(int(x 2*y 3, x=0..1), y=0..2):**

рассмотрим

Пример

Вычисления кратного интеграла $\int\limits_2^4 dy \int\limits_0^y \frac{y^3}{x^2+y^2}dx$

> **Int(Int(y^3 (x^2 +y^3),x=0..y),y=2..4)=**

int(int(y^3(x^2+y^2), x=0..y),y=2..4);

$$\int_2^4 dy \int_0^y \frac{y^3}{x^2+y^2} dx = \frac{14}{3}\pi$$

Здесь Int – команда отложенного действия.

Пример

Вычисления двойного интеграла $\iint\limits_D \sin(x+2y)dxdy$ по области, ограниченной линиями $y=0,\ y=x,\ x+y=\frac{\pi}{2}$.

Для этого сначала следует описать область интегрирования D в виде неравенств: $D=\{(x,y): y\le x\le \frac{\pi}{2}-y,\ 0\le y\le \frac{\pi}{2}\}$

> **restart: with(student):**

> **J:=Doubleint(sin(x+2*y), x=y..Pi/2-y, y=0..Pi/2);**

$$J := \int_0^{\frac{1}{2}\pi} \int_y^{\frac{1}{2}\pi - y} \sin(x+2y)dxdy$$

> **J:=value(%);**

$$J := \frac{2}{3}$$

Пример

Вычисления тройного интеграла $\int\limits_{-1}^{1} dx \int\limits_{x^2}^{1} dy \int\limits_{0}^{2} (4+z)dz$.

Следует указать порядок интегрирования в соответствии с выражением подынтегральной функции.

> **J:=Tripleint(4+z, y=x 2..1,x=-1..1, z=0..2);**

$$J := \int_0^2 \int_{-1}^{1} \int_{x^2}^{1} (4+z)dydxdz$$

> **J:=value(%);**

$$J := \frac{40}{3}$$

3 Лекция 16. Векторный анализ

Рассмотрим основные дифференциальные операции векторного анализа и команды *Maple* для их вычисления, эти процедуры содержатся в библиотеке $linalg$.

Градиент скалярной функции $f(x,\ y,\ z)$ – вектор, координатами которого являются частные производные по соответствующим переменным: $\mathrm{grad}\, f(x,y,z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right)$. В *Maple* градиент вычисляется одноименной командой **grad(f,[x,y,z],c)**, здесь и в дальнейшем f – функция, $[x, y, z]$ – набор переменных, от которых она зависит.

Параметр c позволяет вычислять данную дифференциальную операцию в различных криволинейных координатах (по умолчанию используется прямоугольная декартова система координат). Этот параметр может указываться во всех имеющихся в *Maple* дифференциальных операциях, так для цилиндрических координат следует выбрать $coords = cylindrical$, сферических – $coords = spherical$.

Лапласиан скалярной функции $f(x,\ y,\ z)$ – оператор, действующий на функцию $f(x,\ y,\ z)$ по правилу: $\Delta f = \dfrac{\partial^2 f}{\partial x^2} + \dfrac{\partial^2 f}{\partial y^2} + \dfrac{\partial^2 f}{\partial z^2}$ и вычисляется командой

laplacian(f,[x,y,z],c).

Дивергенцией вектор-функции $\mathbf{F}(x, y, z)$ является функция (скалярная), вычисляемая по правилу: $\mathrm{div}\mathbf{F}(x,y,z) = \dfrac{\partial F_x}{\partial x} + \dfrac{\partial F_y}{\partial y} + \dfrac{\partial F_z}{\partial z}$, а в *Maple* – командой **diverge(F,[x,y,z],c)**, **F** – вектор-функция, **[x,y,z]** – набор переменных, от которых она зависит. Ротором вектор-функции $\boldsymbol{F}(x,\ y,\ z)$ является вектор с координатами:

$$\mathrm{rot}\mathbf{F} = \left[\left(\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z}\right), \left(\frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x}\right), \left(\frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y}\right)\right];$$

он вычисляется командой

curl(F,[x,y,z],c).

Для вектор-функции $\mathbf{F}(x,\, y,\, z)$ матрица Якоби вычисляется

$$J = \begin{bmatrix} \frac{\partial F_x}{\partial x} & \frac{\partial F_y}{\partial x} & \frac{\partial F_z}{\partial x} \\ \frac{\partial F_x}{\partial y} & \frac{\partial F_y}{\partial y} & \frac{\partial F_z}{\partial y} \\ \frac{\partial F_x}{\partial z} & \frac{\partial F_y}{\partial z} & \frac{\partial F_z}{\partial z} \end{bmatrix}$$

с помощью команды **jacobian(F,[x,y,z]).**

Пример использования перечисленных команд.

Пусть дана функция $u(x,y) = \operatorname{arctg}\dfrac{y}{x}$. Найти $\operatorname{grad} u(x,y)$, углы $\operatorname{grad} u$ с осями координат, производную функции $u(x,y)$ по направлению вектора $\boldsymbol{q}$=[1,1].

> **restart: with(linalg):**

> **u:=arctan(y/x): g:=simplify(grad(u, [x, y]));**

$$g := \left[-\frac{y}{x^2+y^2},\ \frac{x}{x^2+y^2} \right]$$

> **alpha:=simplify(angle(g, [1, 0]));**

$$\alpha := \pi - \arccos\left(\frac{y}{(x^2+y^2)\sqrt{\dfrac{1}{x^2+y^2}}} \right)$$

> **beta:=simplify(angle(g, [0, 1]));**

$$\beta := \arccos\left(\frac{x}{(x^2+y^2)\sqrt{\frac{1}{x^2+y^2}}} \right)$$

Косинусы этих углов являются направляющими косинусами $\operatorname{grad} u(x,y)$, убедитесь, что сумма их квадратов равна единице.

> **simplify(cos(alpha) 2+cos(beta) 2):**

Производная функции u по направлению $\boldsymbol{q}$ равна скалярному произведению градиента этой функции и нормированного вектора $\boldsymbol{q}$: $\dfrac{\partial u}{\partial \mathbf{q}} =$ (grad$u$, $\mathbf{e}$), где $\mathbf{e} = \dfrac{\mathbf{q}}{\|\mathbf{q}\|}$ – нормированный вектор $\boldsymbol{q}$.

> **q:=vector([1,1]); e:=normalize(q);**

$$q{:=}[1,\ 1]$$

$$:=\left[\tfrac{1}{2}\sqrt{2},\ \tfrac{1}{2}\sqrt{2}\right]$$

> **udq:=simplify(dotprod(g,e));**

$$udq{:=}\frac{1}{2}\frac{\sqrt{2}(-y+x)}{x^2+y^2}$$

Пример

Пусть дана вектор-функция $\boldsymbol{F}(x,\ y,\ z) = [x^2yz,\ xy^2z,\ xyz^2]$. Найти div $\mathbf{F}$ и rot$\mathbf{F}$.

> **F:=vector([x^2*y*z, x*y^2*z, x*y*z^2]);**

> **divF:=diverge(F, [x, y, z]);**

$$divF{:=}6xyz$$

> **rotF:=curl(F, [x, y, z]);**

$$rotF := [xz^2 - xy^2, x^2x - xz^2, y^2z - x^2z]$$

При каком значении параметра функция $u = x^3{+}axy^2$ удовлетворяет уравнению Лапласа $\Delta u{=}0$?

> **u:=x^3+a*x*y^2:**

> **Delta(u):=laplacian(u, [x,y]);**

$\Delta(\mathrm{x}^3{+}\mathrm{axy}^2){:=}6x{+}2ax$

> **a=solve(%=0,a);**

a=-3

Докажем, что функция $u = \dfrac{e^{-kr} + e^{kr}}{r}$, где $r = \sqrt{x^2 + y^2 + z^2}$ удовлетворяет дифференциальному уравнению $\Delta u - k^2 u = 0$, k – постоянная.

> **u:=(exp(-k*r)+exp(k*r))/r:**

> **Delta(u):=simplify(laplacian(u, [r, theta, phi], coords=spherical));**

$$\Delta\left(\frac{\mathrm{e}^{(-kr)} + \mathrm{e}^{(kr)}}{r}\right) := \frac{k^2\left(\mathrm{e}^{(-2kr)} + 1\right)\mathrm{e}^{(kr)}}{r}$$

> **simplify(%-k^2*u);**

$$0$$

Найдем матрицу Якоби и ее определитель вектор-функция $\boldsymbol{v}$=$[x, y/x]$.

> **v:=vector([x, y/x]): jacobian(v, [x, y]);**

$$\begin{bmatrix} 1 & 0 \\ -\dfrac{y}{x^2} & \dfrac{1}{x} \end{bmatrix}$$

> **det(%):**

В результате использования возможностей, предусмотренных в пакете Maple, традиционно трудоемкий анализ функций многих переменных не требует значительных “вычислительных усилий”.

4 Лекция 17. Ряды. Интегральные преобразования

Конечные и бесконечные суммы $\sum_{n=a}^{b} S(n)$ вычисляются командой прямого исполнения sum и отложенного исполнения Sum, они имеют одинаковый синтаксис: **sum(expr, n=a..b),** здесь $expr$ – выражение, зависящее от индекса суммирования, $a..b$ – пределы индекса суммирования, для бесконечного ряда верхний предел равен $infinity$.

Аналогично вычисляются произведения $\prod_{n=a}^{b} P(n)$ командами прямого действия **product(P(n),n=a..b)** и отложенного — $ProductP(n), n = a..b)$. Следующие примеры иллюстрируют особенности их использования:

Пример

Найти полную и N-частичную суммы ряда, общий член которого равен: $a_n = \dfrac{1}{(3n-2)(3n+1)}$.

> **restart: a[n]:=1/((3*n-2)*(3*n+1));**

$$a_n := \frac{1}{(3n-2)(3n+1)}$$

> **S[N]:=Sum(a[n], n=1..N)=sum(a[n], n=1..N);**

$$S_N := \sum_{n=1}^{N} \frac{1}{(3n-2)(3n+1)} = -\frac{1}{3}\frac{1}{3N+1} + \frac{1}{3}$$

> **S:=limit(rhs(S[N]), N=+infinity);**

$$S := \frac{1}{3}$$

Пример

Найти сумму биномиального ряда $\sum_{n=1}^{\infty} C_n^4 (1-x)^n$.

> **Sum(binomial(n,4)*(1-x)^n, n=1..infinity)=**
sum(binomial(n,4)*(1-x)^n, n=1..infinity);

$$\sum_{n=1}^{\infty} \text{binomial}(n,4)(1-x)^n = \frac{(1-x)^4}{x^5}$$

Разложение функции в степенной ряд и ряд Тейлора легко осуществляется средствами пакета. Так, чтобы разложить функцию $f(x)$ в степенной ряд в окрестности точки $x = a$

$$f(x) = C_0 + C_1(x-a) + C_2(x-a)^n + ... + O(x^n)$$

следует воспользоваться командой **series(f(x), x=a, n),** здесь – центр окрестности разложения, n – заданное число членов ряда; разложение в ряд Тейлора производится по команде **taylor(f(x), x=a, n)**.

В обоих случаях тип результата *series*, он преобразуется, например, в полином конвертированием $convert(\%, polynom)$.

Функция многих переменных $f(x_1,\ldots,x_n)$ раскладывается в ряд Тейлора по набору переменных $(x_1,\ldots,x_n)$ в окрестности точки $(a_1,\ldots,a_n)$ до порядка n по команде **mtaylor(f(x), [x1,...,xn], n).** Эта команда находится в стандартной библиотеке.

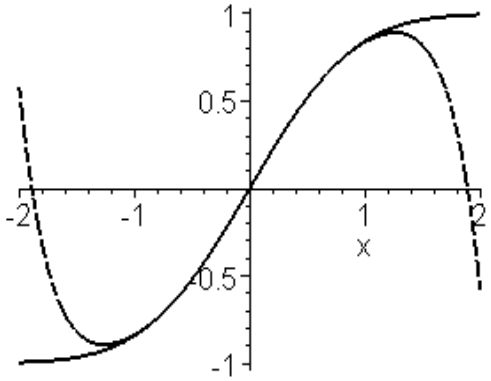
Пример

Построить графики интеграла ошибок $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int\limits_0^x e^{-t^2} dt$ и его разложения в ряд Тейлора в окрестности нуля.

> **taylor(erf(x),x,8): p:=convert(%,polynom);**

$$p := 2\frac{x}{\sqrt{\pi}} - \frac{2}{3}\frac{x^3}{\sqrt{\pi}} + \frac{1}{5}\frac{x^5}{\sqrt{\pi}} - \frac{1}{21}\frac{x^7}{\sqrt{\pi}}$$

> **plot({erf(x),p},x=-2..2,thickness=[2,2],**
linestyle=[1,3], color=[red,green]);



Пунктирной линей изображен график функции, а сплошной – приближения рядом Тейлора.

Разложение функции в ряд Фурье. В *Maple* нет команды, позволяющей производить разложение функции в тригонометрический ряд Фурье. Однако можно создать собственную процедуру разложения ряд Фурье. Так, если требуется разложить $2l$-периодическую функцию $f(x)$ на интервале $[x_1,x_2]$ в ряд Фурье

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos\frac{k\pi x}{l} + \sum_{k=1}^{\infty} b_k \sin\frac{k\pi x}{l},$$

где l=$(x_2 - x_1)/2$;

$$a_0 = \frac{1}{l}\int\limits_{x_1}^{x_2} f(x)dx; \quad a_k = \frac{1}{l}\int\limits_{x_1}^{x_2} f(x)\cos\frac{k\pi x}{l}dx; \quad b_k = \frac{1}{l}\int\limits_{x_1}^{x_2} f(x)\sin\frac{k\pi x}{l}dx.$$

то получить первые n членов ряда можно с помощью следующей процедуры:

```
> fourierseries:=proc(f,x,x1,x2,n)
> local k, l, a, b, s;
> l:=(x2-x1)/2;
> a[0]:=int(f,x=x1..x2)/l;
> a[k]:=int(f*cos(k*Pi*x/l),x=x1..x2)/l;
> b[k]:=int(f*sin(k*Pi*x/l),x=x1..x2)/l;
```

```
> s:=a[0]/2+sum(a[k]*cos(k*Pi*x/l)+
b[k]*sin(k*Pi*x/l), k=1..n);
> end;
```

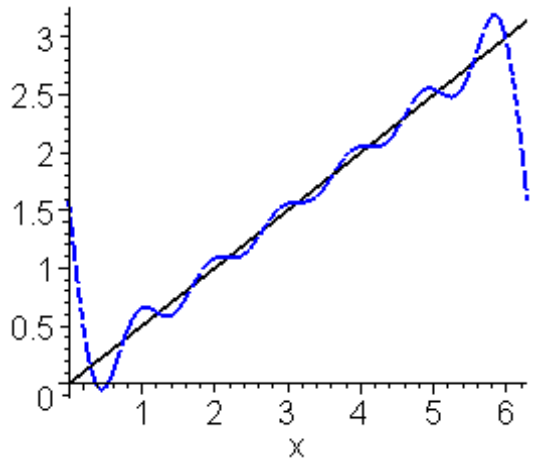
Порядок параметров при обращении к этой процедуре следующий: **fourierseries(f,x,x1,x2,n)**, здесь f – имя – имя независимой переменной; $1, x2$ – интервал разложения; n – число членов ряда.

Воспользуемся предлагаемой процедурой и разложим 2π–периодическую функцию $f(x) = x/2$на интервале $[0;\, 2\pi]$ в ряд Фурье, удерживая 6 членов ряда. Построим графики функции и ее n-частичной суммы ряда Фурье.

```
> f:=x/2:x1:=0:x2:=2*Pi:
> fr:=fourierseries(f,x,x1,x2,6);
```

$$fr := \frac{1}{2}\pi - \sin(x) - \frac{1}{2}\sin(2x) - \frac{1}{3}\sin(3x) - \frac{1}{4}\sin(4x) - \frac{1}{5}\sin(5x) - \frac{1}{6}\sin(6x)$$

```
> plot({fr,f}, x=x1..x2, color=[blue,black],
thickness=2, linestyle=[3,1]]);
```



Пунктирной линией изображен график n-частичной суммы ряда Фурье, а сплошной – самой функции. По виду n-частичной суммы ряда Фурье в данном примере легко установить общий вид этого ряда:

$$f(x) = \frac{\pi}{2} - \sum_{k=1}^{\infty} \frac{\sin kx}{k}.$$

Разложим далее ряд Фурье функцию $f(x) = e^{-x}$ с периодом 2π на интервале $[\pi;\text{-}\pi]$, удерживая 2, 4 и 8 членов ряда. Построим графики функции и ее n-частичных сумм ряда Фурье.

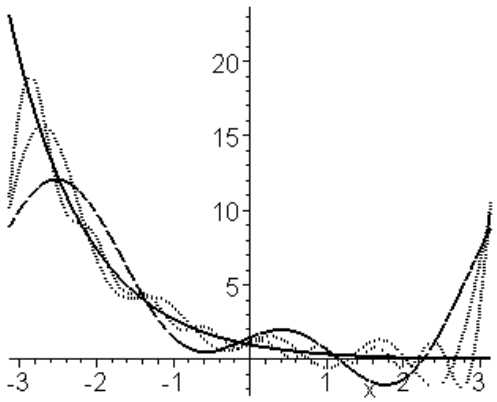
> **f:=exp(-x);x1:=-Pi;x2:=Pi:**

> **fr1:=fourierseries(f,x,x1,x2,2):**

> **fr2:=fourierseries(f,x,x1,x2,4):**

> **fr3:=fourierseries(f,x,x1,x2,8):**

> **plot({f,fr1,fr2,fr3},x=x1..x2,color=[black, blue, green, red], thickness=2, linestyle= [1,3,2,2]);**



Сплошной линией изображен график функции, пунктирными – графики n-частичных сумм ряда Фурье. Очевидно точность разложения растет с ростом n. Поскольку, как уже упоминалось, Maple не имеет функций разложения в ряд Фурье, следующая процедура может быть добавлена в библиотеку пользователя.

Интегральные преобразования В *Maple* имеется пакет **inttrans**, который предназначен для выполнения различных интегральных преобразований.

Прямое преобразование Фурье функции $f(x)$ вычисляется по формуле

$$F(k) = \int_{-\infty}^{\infty} f(x)e^{-ikx}dx.$$

В *Maple* оно может быть найдено командой **fourier(f(x),x,k)**, здесь x – переменная, по которой производится преобразование, k – имя переменной, которое следует присвоить параметру преобразования.

Обратное преобразование Фурье задается формулой

$$f(x) = \frac{1}{2\pi} \int\limits_{-\infty}^{\infty} F(k) e^{ikx} dk$$

и вычисляется командой **invfourier(F(k),k,x).**

Описанное выше прямое и обратное преобразования Фурье называются комплексными и применяются в тех случаях, когда функция $f(x)$ задана на всей оси. Если функция $f(x)$ задана только при >0, то рекомендуется использовать синус- и косинус- преобразования Фурье.

Прямое и обратное синус-преобразования Фурье функции $f(x)$ определяются формулами

$$F(k) = \sqrt{\frac{2}{\pi}} \int\limits_{0}^{\infty} f(x) \sin kx dx \quad \text{и} \quad f(x) = \sqrt{\frac{2}{\pi}} \int\limits_{-\infty}^{\infty} F(k) \sin kx dk.$$

Поскольку формулы синус-преобразования Фурье симметричны относительно замены x на k, то в *Maple* эти преобразования вычисляются одной командой, но с различным порядком указания параметров: **fourier-sin(f(x),x,k)** – вычисляет прямое синус-преобразование Фурье; **fourier-sin(F(k),k,x)** – вычисляет обратное синус-преобразование Фурье.

Аналогично, прямое и обратное косинус-преобразования Фурье функции $f(x)$ определяются формулами

$$F(k) = \sqrt{\frac{2}{\pi}} \int\limits_{0}^{\infty} f(x) \cos kx dx \text{и} \quad f(x) = \sqrt{\frac{2}{\pi}} \int\limits_{-\infty}^{\infty} F(k) \cos kx dk.$$

В *Maple* эти преобразования вычисляются одной командой, но с различным порядком указания параметров: **fouriercos(f(x),x,k)** –

вычисляет прямое косинус-преобразование Фурье; **fouriercos(F(k),k,x)** – вычисляет обратное косинус-преобразование Фурье. рассмотрим примеры их использования.

Пример

Для функции $f(x) = e^{-a|x|}$, $a > 0$ найти преобразование Фурье.

> **restart:with(inttrans): assume(a>0):**

> **fourier(exp(-a*abs(x)),x,k);**

$$2\frac{a\sim}{k^2 + a\sim^2}$$

Пример

Для функции $F(k) = \frac{1}{k^2-a^2}$, $a>0$ найти обратное преобразование Фурье.

> **invfourier(1/(k 2-a 2),k,x);**

$$-\frac{1}{4}\frac{I(\text{Heaviside}(x) - \text{Heaviside}(-x))(-\text{e}^{(Ia\sim x)} + \text{e}^{(-Ia\sim x)})}{a\sim}$$

После обратного преобразования Фурье результат часто содержит специальные функции (см. приложение). В данном примере в строке вывода появилась функция Хевисайда:

$$\text{Heaviside}(x) = \begin{cases} 1,\ x > 0 \\ 0,\ x < 0 \end{cases}$$

Результат выполнения обратного преобразования Фурье может иметь более компактный вид после применения команды **convert(%,trig).**

> **convert(%,trig);**

$$-\frac{1}{2}\frac{(\text{Heaviside}(x) - \text{Heaviside}(-x))\sin(a\sim x)}{a\sim}$$

Пример

Для функции $f(x) = e^{-ax}\sin bx$, $a>0$ найти синус- и косинус-преобразования Фурье.

> **f:=exp(-a*x)*sin(b*x):**

> **fouriercos(f,x,k);**

$$\frac{\sqrt{2}\left(\frac{1}{2}\frac{k+b}{a\sim^2+(k+b)^2}+\frac{1}{2}\frac{b-k}{a\sim^2+(b-k)^2}\right)}{\sqrt{\pi}}$$

> **fouriercos(f,x,k);**

$$\frac{1}{2}\frac{\sqrt{2}a\sim\left(\frac{1}{a\sim^2+(b-k)^2}-\frac{1}{a\sim^2+(k+b)^2}\right)}{\sqrt{\pi}}$$

Преобразование Лапласа функции $f(x)$ (если оно существует) вычисляется по формуле:

$$F(p)=\int\limits_0^\infty f(x)e^{-px}dx.$$

Получаемая функция $F(p)$ называется изображением.

В *Maple* это преобразование вычисляется командой **laplace(f(x),x,p)**, здесь x – переменная, по которой производится преобразование; p – имя переменной, которое следует присвоить параметру преобразования.

Обратное преобразование Лапласа (называется оригиналом) вычисляется по формуле:

$$f(x)=\frac{1}{2\pi i}\int\limits_{a-i\infty}^{a+i\infty}F(p)e^{px}dp.$$

Оригинал $f(x)$ (если он существует) может быть найден по изображению $F(p)$ командой **invlaplace(F(p),p,x).** Рассмотрим примеры их применения.

Пример

Найти изображение функции $f(x)=\cos(ax)\mathrm{sh}(bx)$.

> **restart:with(inttrans):**

> **F(p)=laplace(cos(a*x)*sinh(b*x), x, p);**

$$F(p) = \frac{1}{2}\frac{p-b}{(p-b)^2+a^2} - \frac{1}{2}\frac{p+b}{(p+b)^2+a^2}.$$

Пример

Найти оригинал Лапласа функции $F(p) = \frac{1}{p^2+2ap}$, $a>0$.

> **assume(a>0): invlaplace(1/(p 2+2*a*p),p,x):**

> **combine(%,trig);**

$$\frac{1}{2}\frac{1-\mathrm{e}^{(-2a\sim x)}}{a\sim}.$$

5 *Вопросы для самопроверки и задания.*

1. **Какое значение принимает t?**

$$> f := sin(x);$$

$$> r := taylor(f,\ x = 0,\ 6);$$

$$> p = convert(r,\ `polynom`);$$

$$> t := type(p,\ algebraic);$$

   (a) 0

   (b) 1

   (c) 2

2. **Какое значение принимает t?**

$$> f := sin(x);$$

$$> r := taylor(f,\ x = 0,\ 6);$$

$$> p = convert(r,\ `polynom`);$$

$$> pp := convert(p,\ `list`);$$

$$> t := nops(pp);$$

(a) 0

(b) 1

(c) 2

(d) 3

3. **Какая из перечисленных команд** *linalg* **позволяет определить собственные значения, кратность и собственные векторы?**

(a) ownvectors

(b) eigenvectors

(c) eigenvalues

4. **Какой командой из linalg определяется число строк матрицы**

(a) rowdim

(b) coldim

(c) vectdim

5. **Матрица** c **в результате выполнения следующих команд является**

$> with(linalg);$

$> c := matrix(3,3,[\,]);\ a := randmatrix(3,3);\ b := transpose(a);$

$> for\ i\ from\ 1\ to\ rowdim(a)\ do$

$for\ j\ from\ 1\ to\ coldim(a)\ do$

$c[i,j] := a[i,j] + b[i,j]\ od\ od;\#\ v := [map(evalf, eigenvals(c))];$

(a) нулевой

(b) единичной

(c) симметричной

6. **Выберите правильную характеристику вектора** v **в результате выполнения следующих команд**

$$> with(linalg);$$

$$> a := randmatrix(3,3);$$

$$> r = det(a); \#v := [r, det(inverse(a))];$$

(a) состоит из обратных элементов

(b) переменная v не определена

(c) empty

Список литературы

1. *Чарльз Генри Эдвардс, Дэвид Э. Пенни.* Дифференциальные уравнения и краевые задачи: моделирование и вычисление с помощью Mathematica, Maple и MATLAB. Киев: Диалектика-Вильямс, 2007.

2. *Е. Р. Алексеев, О. В. Чеснокова.* Решение задач вычислительной математики в пакетах Mathcad 12, MATLAB 7, Maple 9. М: НТ Пресс, 2006, 496с. ISBN: 5-477-00208-5

3. *Дьяконов В.П.* Maple 9 в математике, физике и образовании (МФО). М.: СОЛОН-Пресс, 2004.

4. *Дьяконов В.П.* Maple 8 в МФО. М.: СОЛОН-Пресс, 2003

5. *Манзон Б.М.* Maple V Power Edition. М.: Филинъ, 1998.

6. *Говорухин В.Н., Цибулин В.Г.* Компьютер в математическом исследовании: Maple, MATLAB, LaTeX. Питер. 2001.

7. *Прохоров Г.В., Леденев М.А., Колбеев В.В.* Пакет символьных вычислений Maple V. М.: Петит, 1997.

8. *Бугров Я.С., Никольский С.М.* Б Элементы линейной алгебры и аналитической геометрии. М.: Наука. 1989.

9. *Бугров Я.С., Никольский С.М.* Дифференциальное и интегральное исчисление. М.: Наука. 1989.

10. *Бугров Я.С., Никольский С.М.* Дифференциальные уравнения. Кратные интегралы. Ряды. Функции комплексного переменного. М.: Н. 1989.

11. *Ильин В.А., Позняк Э.Г.* Аналитическая геометрия. М.: Наука. 1970.

12. *Ильин В.А., Позняк Э.Г.* Линейная алгебра. М.: Наука. 1970.

13. *Никольский С.М.* Курс математического анализа М.: Наука. 1991.

14. *Эльсгольц Л.Э.* Дифференциальные уравнения и вариационное исчисление. М.: Эдиториал, 2000.

Глава 6

Календарно-тематический план

Наименование тем, их содержание, объем в часах

Maple-интерфейс. Среда символьных вычислений (8 ч.)

Maple-интерфейс. Форсированное освоение пакета

Конструирование выражений. "Динамический" инструментарий

Базовые типы данных. Контроль типов

Конструкции языка. Процедуры. Библиотеки

Алгебраические преобразования. Элементы линейной алгебры (6 ч.)

Аналитические преобразование алгебраических выражений. Функциональные операторы

Операции линейной алгебры

Свойства матриц, матричные уравнения, библиотека LinearAlgebra

Графика (6 ч.)

Интерактивная графика. 2-D графика в Maple

Структуры. Серия графических объектов

Трехмерные графики. Анимация

Обыкновенные дифференциальные уравнения (6 ч.)

Решение дифференциальных уравнений

Приближенное и численное решение дифференциальных уравнений

Пакет графического представления решений Detools

номер	Наименование лекции	Количество часов по плану		
		лекций	Практ.	лаб. самост.
1.	*Maple-интерфейс. Форсированное освоение пакета*	2	1	2
2.	*Конструирование выражений. "Динамический" инструментарий*	2	1	2
3.	*Базовые типы данных. Контроль типов*	2	1	2
4.	*Конструкции языка. Процедуры. Библиотеки*	2	1	2
5	*Аналитические преобразование алгебраических выражений. Функциональные операторы*	2	1	2
6	*Операции линейной алгебры*	2	1	2
7	*Свойства матриц, матричные уравнения, библиотека LinearAlgebra*	2	1	2
8	*Интерактивная графика. 2-D графика в Maple*	2	1	2
9	*Структуры. Серия графических объектов*	2	1	2
10	*Трехмерные графики. Анимация*	2	1	2
11	*Решение дифференциальных уравнений*	2	1	2
12	*Приближенное и численное решение дифференциальных уравнений*	2	1	2
13	*Пакет графического представления решений Detools*	2	1	2
14	*Анализ функций и дифференциальные операторы*	2	1	2
15	*Интегрирование функций многих переменных*	2	1	2
16	*Векторный анализ*	2	1	2
17	*Ряды. Интегральные преобразования*	2	1	2

Анализ функций и дифференциальные операторы

Использование Maple в теории функций многих переменных (8 ч.)

Интегрирование функций многих переменных

Векторный анализ

Ряды. Интегральные преобразования

Глава 7

Проектные задания к курсу

Проектное задание 1.

Найти сумму бесконечного степенного ряда от симметричной матрицы A, воспользовавшись процедурами Maple, а также алгоритмом, реализованным в Maple, вычисления рядов и функций от матриц:

1. проверить условие сходимости матричного ряда (максимальное собственное значение матрицы меньше единицы)
2. разложить матрицу на произведение диагональной D и ортогональной матриц Q (A=QDQ^T), Q - ортогональная матрица, столбцы которой – собственные векторы; D - диагональная, с собственными числами на диагонали
3. упростить A^k с учетом разложения
4. вычислить сумму ряда, используя значения суммы ряда для бесконечной геометрической прогрессии.

Аналогичное задание можно сформулировать для вычисления тригонометрических функций от матриц.

Проектное задание 2.

Используя метод Гюна и Эйлера, решить задачу Коши; построить графики решения в одном окне, подписать решения и выделить стилем линий. Создать процедуры решения задачи Коши для обоих методов.

Воспользоваться процедурами для решения задачи Коши вида:

$f(x,y) = x + \sin(\frac{y}{3}), \quad y_0 = 4.6, \quad x \in [1.6, 2.6]$, где $y' = f(x,y);\ f(x_0) = y_0,\ x \in [x_0, x_k]$;

Схема построения численного решения задачи Коши методом Эйлера:

$$y' = f(t,y), \quad y(t_0) = y_0. \tag{7.0.1}$$

$f(t,y)$ – непрерывна на $\Omega = [a \equiv t_0, b] \times [c,d]$ и удолетворяет условию Липши

1. Разобьем $[a,b]$ на отрезки точками $t_k = a + kh$, где $k = 0, 1, \ldots, m$ и $h = \frac{b-a}{m}$.

2. В окрестности t_0 представим функцию $y(t)$ по ф-ле Тейлора:

$$y(t) = y(t_0) + y'(t_0)(t - t_0) + o(t - t_0). \tag{7.0.2}$$

3. На каждом отрезке осуществим рекуррентный переход в (7.0.2), с помощью которого легко получается приближенное численное решение:

$$y(t_{k+1}) \approx y(t_k) + y'(t_k)h. \tag{7.0.3}$$

Схема построения численного решения задачи Коши методом Гюна:

Проинтегрируем (7.0.1)

$$\int_{t_0}^{b} y'(t)dt = \int_{t_0}^{b} f(t,y)dt \tag{7.0.4}$$

Интеграл в правой части можно приближенно посчитать по формуле трапеций, и тогда (7.0.4) преобразуется к виду:

$$y(b) - y(t_0) \approx h(f(t_0, y(t_0)) + f(b, y(b)))/2. \tag{7.0.5}$$

Аналогичным образом, разобьем $[a, b]$ на отрезки точками $t_k = a + kh$, и получим согласно (7.0.5), рекуррентное соотношение:

$$y(t_{k+1}) \approx y(t_k) + h(f(t_k, y(t_k)) + f(t_{k+1}, y(t_{k+1})))/2. \tag{7.0.6}$$

При замене в правой части (7.0.6) $y(t_{k+1})$ по ф-ле (7.0.3) , получаем окончательное представление для численного решения методом Гюна.

Проектное задание 3.

Построить численное решение дифференциального уравнения второго порядка в частных производных для трех типов уравнений. Построить поверхность натянутую на матрицу узловых решений, полученных с помощью сеточных схем (см. разностные схемы). Использовать команду matrixplot *библиотеки* plots. *для построения поверхности.*

1

$$u_t = u_{xx},$$

$$(u(x,t), \quad x \in [0,1], \quad t \in [0, 0.1]);$$

$$u|_{x=0} = t^2; \quad u|_{x=1} = e^t; \quad u|_{t=0} = \sin(\pi x) + \sin(2\pi x);$$

2

$$u_t = u_{xx} + \sin x,$$

$$(u(x,t), \quad x \in [0,1], \quad t \in [0, 0.1]);$$

$$u|_{x=1} = 0; \quad u|_{x=0} = 0; \quad u|_{t=0} = \sin(\pi x) + \sin(3\pi x);$$

3

$$u_{xx} + u_{yy} = -4u$$

$$(u(x,y), \quad (x,y) \in \Omega = [0,1] \times [0,1];$$

$$u|_{x=\partial\Omega} = \cos(2\pi x) + \sin(2\pi y);$$

4

$$u_{tt} = 4u_{xx},$$

$$(u(x,t), \quad (x,t) \in \Omega = [0,1] \times [0,0.5];$$

$$u|_{x=0,x=1} = 0; \quad \left.\frac{\partial u}{\partial t}\right|_{t=0} = 0;$$

Разностные схемы

решения дифференциальных уравнений в частных производных

В прямоугольной области введем сетку, h_x, h_y – шаги сетки по x, y соответственно;

$$u(x_i, y_i) = u_{i,j}, \; u(x_i \pm h_x, y_i \pm h_y) = u_{i+1,j+1} \tag{7.0.7}$$

– значения неизвестной функции в узлах сетки (сеточный аналог функции).

Для функции $u(x,y)$ справедливо разложение по формуле Тейлора в h_x – окрестности узла (x_i, y_i):

$$u(x_{i\pm1}, y_i) = u(x_i, y_i) \pm u'_x(x_i, y_i)h_x + \frac{1}{2}u''_{xx}(x_i, y_i)h_x^2 + O(h^3), \tag{7.0.8}$$

В соответствии с обозначениями (7.0.7) из представлений (7.0.8) легко получаются выражения частных производных первого и второго порядка

$$\begin{gathered}\left.\frac{\partial u}{\partial x}\right|_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2h_x}, \quad \left.\frac{\partial u}{\partial y}\right|_{i,j} = \frac{u_{i,j+1} - u_{i,j-1}}{2h_y}, \\ \left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j} = \frac{u_{i-1,j} - 2\,u_{i,j} + u_{i+1,j}}{h_x^2}, \quad \left.\frac{\partial^2 u}{\partial y^2}\right|_{i,j} = \frac{u_{i,j-1} - 2\,u_{i,j} + u_{i,j+1}}{h_y^2}\end{gathered} \tag{7.0.9}$$

1. Уравнение Лапласа (эллиптическое)

$$\begin{aligned} &\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0, \quad x\in[0,a], \quad y\in[0,b], \\ &u|_{y=0}=f_1(x), \quad u|_{y=b}=f_2(x), \quad u|_{x=0}=g_1(y), \quad u|_{x=a}=g_2(y), \end{aligned} \tag{7.0.10}$$

В результате подстановки разностных аналогов второй производной (7.0.9) в выражения (7.0.10) и простейших преобразований, получаем сеточный аналог исходного уравнения:

$$pu_{i,j-1}+u_{i-1,j}-2(1+p)u_{i,j}+u_{i+1,j}+pu_{i,j+1}=0, \quad p=(h_x/h_y)^2. \tag{7.0.11}$$

Пусть количество ячеек сетки по x и y равно n $(i,j=1,\ldots,n)$, очевидно, что количество уравнений вида (7.0.11) соответствует количеству внутренних узлов сетки области, и они образуют систему линейных алгебраических уравнений, порядка $(n-1)^2$, при этом значения функций с нулевыми индексами заменяются своими граничными узловыми аналогами и являются причиной правой части неоднородной системы. Решением системы и будет искомая функция, представляемая значениями в узлах области (сеточная функция).

Данная схема решения уравнений в частных производных пригодна для уравнений Пуассона, Гельмгольца, а также для уравнений гиперболического типа, например, волнового уравнения.

Уравнение теплопроводности (параболическое)

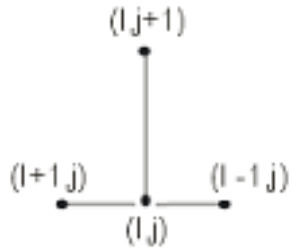


Рис. 7.1:

$$\begin{aligned} &\frac{\partial u}{\partial t} = c^2 \frac{\partial^2 u}{\partial x^2}, \quad x \in [0,a], \quad t \in [0,b] \ , \\ &u|_{t=0} = f(x), \\ &u|_{x=0} = g_1(t), \ u|_{x=a} = g_2(t). \end{aligned} \tag{7.0.12}$$

Четырехточечная схема

Для дискретизации данной задачи, в силу того, что уравнение содержит частные производные первого и второго порядков, выбирается схема замены производных, указанная на рисунке.7.1.

Из нее следует, что для замены второй производной по x в соответствии с центральными разностями (7.0.9) потребуются значения функции в трех узлах (вдоль x), тогда как в случае производной по t выбираются односторонние разности с участием соседних узлов

$$\left.\frac{\partial u}{\partial t}\right|_{i,j} = \frac{u_{i,j+1} - u_{i,j}}{h_t}.$$

В результате такой замены получается разностное уравнение

$$u_{i,j+1} = (1-2r)\, u_{i,j} + r\, u_{i+1,j} + r\, u_{i-1,j}, \quad r = \left(c^2 h_t / h_x^2\right), \tag{7.0.13}$$

позволяющее построить численное решение; очевидно, что узлы сетки каждого последующего $j+1$ – го слоя, полностью определяются

значениями предыдущего слоя и граничными значениями (функций $g_1,\ g_2$).

Для устойчивости метода следует выбирать такое разбиение, чтобы выполнялось условие $r << 1$.

Шеститочечная схема

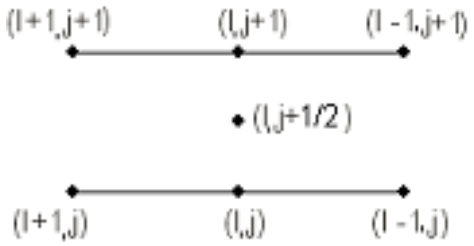


Рис. 7.2:

Все предшествующие подходы основывались на том, что производные, участвующие в уравнениях вычисляются в текущем узле (i, j).

В излагаемом подходе вводится узел $(i, j + 1/2)$, не принадлежащий сетке. В этом узле и вычисляются производные; первого порядка – по формулам центральных разностей, а второго – как полусумма производных соседних слоев:

$$\begin{aligned} \left.\frac{\partial u}{\partial t}\right|_{i,j+1/2} &= \frac{u_{i,j+1} - u_{i,j}}{2(h_t/2)}, \\ 2\left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j+1/2} &= \frac{u_{i-1,j} - 2\,u_{i,j} + u_{i+1,j}}{h_x^2} + \frac{u_{i-1,j+1} - 2\,u_{i,j+1} + u_{i+1,j+1}}{h_x^2} \end{aligned} \tag{7.0.14}$$

Подставляя (7.0.14) в (7.0.12), получаем уравнение

$$\begin{aligned} &-(r/2)\,u_{i-1,j+1} + (1+r)\,u_{i,j+1} - (r/2)\,u_{i+1,j+1} \\ &\quad (r/2)\,u_{i-1,j} + (1-r)\,u_{i,j} + (r/2)\,u_{i+1,j} \end{aligned} \tag{7.0.15}$$

для фиксированного j и изменяющегося$i (i = 1, \ldots, n)$ получается трехдиагональная система уравнений порядка $n - 1$,равного числу внутренних узлов слоя.

Таким образом, чтобы построить полное решение уравнения следует строить и решать систему на каждом слое, т.е. m– раз.

Учет граничных условий Неймана

Пусть задано условие $\left.\dfrac{\partial u}{\partial x}\right|_{x=a} = 0$; в представлении первой производной

$$\left.\frac{\partial u}{\partial x}\right|_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2h_x} = 0,\ i = n, \text{и} \quad \left.\frac{\partial u}{\partial x}\right|_{i=n,j} = \frac{u_{n+1,j} - u_{n-1,j}}{2h_x} = 0.$$

Несмотря на то, что в области не существует $(n + 1)$–го узла, можно получит приближенное значение $u_{n+1,j} = u_{n-1,j}$ и редуцировать (исключить) его из системы в случае $i = n$.

Аналогичный подход справедлив и для нетривиального условия Неймана.

1 Приложение.

Встроенные функции

e^x	**exp(x)**
$\ln x$, $\lg x$	**ln(x)**, **log10(x)**
$\log_a x$	**log[a](x)**
$\sqrt{x}$	**sqrt(x)**
$\|x\|$	**abs(x)**
$\sin x$ $\cos x$,	**sin(x)**, **cos(x)**
$\mathrm{tg}x$	**tan(x)**
$\mathrm{ctg}x$	**cot(x)**
$\sec x$, $\cos \mathrm{ec}x$	**sec(x)**, **csc(x)**
$\arcsin x$	**arcsin(x)**
$\arccos x$	**arccos(x)**
$\mathrm{arctg}x$	**arctan(x)**
$\mathrm{arcctg}x$	**arccot(x)**
$\mathrm{sh}x$	**sinh(x)**
$\mathrm{ch}x$	**cosh(x)**
$\mathrm{th}x$	**tanh(x)**
$\mathrm{cth}x$	**coth(x)**
$\delta(x)$ – функцияДирака	**Dirac(x)**
$\theta(x)$ – ф-я Хевисайда	**Heaviside(x)**

Таблица 7.1: Специальные функции

2 Список пакетов Maple

DEtools - дополнительных средств для дифференциальных уравнений

Domains - для разработки кодов сложных алгоритмов

GF - "поля Галуа"

GaussInt - Гауссовых целых чисел

LREtools - для проведения расчетов с рекуррентными соотношениями

combinat - комбинаторики; combstruct - комбинаторных структур

difforms - дифференциальных форм

finance - финансовой математики

genfunc - для проведения расчетов с производящими функциями

geometry - геометрический

grobner - содержит набор процедур для нахождения базиса Гробнера

group - групп перестановок и конечно-представимых групп

inttrans - интегральных преобразований

liesymm - симметрий Ли

linalg - пакет линейной алгебры

logic - математической логики

Networks - теории графов

numapprox - численной аппроксимации функций

numtheory - теории чисел

orthopoly - ортогональных полиномов

padic - для оперирования p-адическими числами

plots - команд графики и анимации

plottools - вспомогательных инструментариев графики

powseries - генерации и преобразования степенных рядов

simplex - линейной оптимизации

stats - статистики

student - для изучения математики и программирования

sumtools - для вычисления конечных и бесконечных сумм

tensor - тензорной алгебры

totorder - полного упорядочения имен

Библиотека совместного пользования (share-библиотека)

Методы, реализованные в пакетах доступны по команде $with(NamePackage)$.

3 Типы объектов в Maple

!	*	+	.
::	<	<=	<>
=	@	@@	abstract_rootof
algebraic	algext	algfun	algnum
algnumext	And	and	anyfunc
anyindex	anything	appliable	applied
arctrig	Array	array	assignable
atomic	attributed	boolean	BooleanOpt
builtin	character	ClosedIdeal	CommAlgebra
complex	complexcons	composition	const
constant	copyrighted	cubic	cx_infinity
cx_zero	dependent	dictionary	dimension
disjcyc	embedded_axis	embedded_imaginary	embedded_real
equation	even	evenfunc	expanded
extended_numeric	extended_rational	facint	filedesc
filename	finite	float	float[8]
float[]	form	fraction	freeof
function	global	hfarray	hfloat
identical	imaginary	implies	in
indexable	indexed	indexedfun	infinity
integer	intersect	last_name_eval	laurent
linear	list	listlist	literal
local	logical	mathfunc	Matrix
matrix	minus	module	moduledefinition
monomial	MonomialOrder	MVIndex	name
negative	negint	negzero	neg_infinity
Non	NONNEGATIVE	nonnegative	nonnegint
nonposint	nonpositive	nonreal	Not
not	nothing	numeric	odd

Описание типов, представленных в данном окружении, доступно по инициализации справки.

oddfunc	operator	Or	or
OreAlgebra	package	partition	patfunc
patindex	patlist	permlist	Point
point	polynom	posint	positive
poszero	pos_infinity	prime	procedure
property	protected	quadratic	quartic
Queue	radalgfun	radalgnum	radext
radfun	radfunext	radical	radnum
radnumext	Range	range	rational
ratpoly	ratseq	realcons	real_infinity
relation	RootOf	rtable	satisfies
scalar	SDMPolynom	sequential	SERIES
series	set	sfloat	ShortMonomialOrder
SkewAlgebra	SkewParamAlgebra	SkewPolynomial	specfunc
specified_rootof	specindex	sqrt	Stack
stack	std	stdlib	string
subset	suffixed	symbol	SymbolicInfinity
symmfunc	table	tabular	taylor
TEXT	trig	trigh	truefalse
type	typefunc	typeindex	undefined
uneval	union	unit	unit_name
Vector	vector	verification	verify
with_unit	xor	zppoly	^