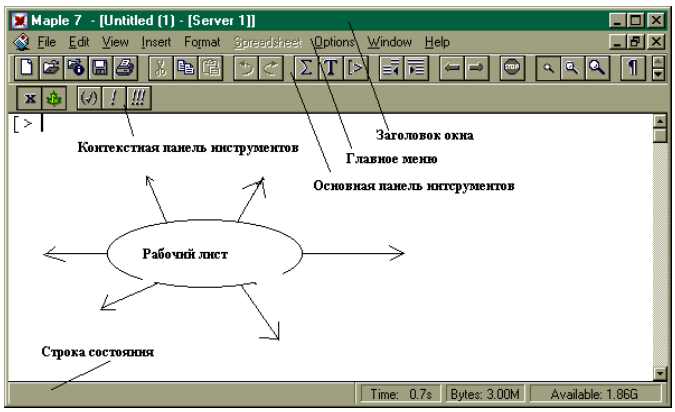
Лекция 1.
Основы работы в системе MAPLE V.

1.1 Начальные навыки.

Графический интерфейс пользователя.

Приложение Maple имеет стандартный графический интерфейс пользователя для программ, работающих под управлением операционных систем семейства Windows. При загрузке программы на экране монитора отображается окно, в котором можно выделить несколько основных частей.



Вид основного меню (его команд) меняется в зависимости от контекста: если на рабочем листе выделен графический объект, то меню содержит команды для работы с графикой. Всего может быть отображено пять видов основного меню:

- Стандартное меню рабочего листа
- Меню электронной таблицы
- Меню двумерной графики
- Меню трехмерной графики
- Меню справочной системы

Ниже главного меню расположена **основная панель инструментов** с рядом кнопок, дублирующих наиболее часто используемые команды главного меню. Непосредственно под основной панелью инструментов расположена **контекстная панель инструментов**, вид которой зависит от того, в какой области рабочего листа расположен курсор и что в этой области отображается. Существует пять видов контекстных панелей инструментов:

- для области ввода
- для области вывода
- для выделенного двумерного графика
- для выделенного трехмерного графика

- для выделенной анимационной графики

Большую часть окна интерфейса занимает **рабочая область**, в которой располагаются рабочие листы, где вводятся команды и отображаются результаты их выполнения. В нижней части окна расположена **строка состояния**, в которой отображаются некоторые параметры работы системы Maple, а также краткая информация относительно выбранной команды меню или кнопки панели инструментов.

Работа в диалоговом режиме.

Диалог с системой Maple происходит на языке задания системе вопросов: пользователь вводит на рабочем листе команды и нажатием клавиши **[Enter]** передает их на исполнение ядру системы Maple.

Рабочий лист состоит из **области ввода** и **области вывода.** В первой пользователь вводит команды Maple, а в области вывода отображаются результаты выполнения команд и операторов, а также двумерная и трехмерная графика (если задан режим вставки графики в рабочий лист, а не отображения ее в отдельном окне).

Содержимое областей ввода и вывода образуют отдельный блок - **Группу вычислений**, которая на рабочем листе отмечается слева квадратной скобкой. Основное свойство группы вычислений заключается в том, что все ее операторы и команды выполняются при однократном нажатии клавиши **[Enter]**.

Для начала работы нам понадобится знание некоторых одиночных и составных знаков элементов синтаксиса Maple:	
>	Знак приглашения к заданию вопроса.
|	Мигающая вертикальная черта - маркер ввода.
;	Знак фиксации конца выражения с выводом результата вычислений на экран.
:	Знак фиксации конца выражения, предотвращающий вывод результата на экран.
:=	Оператор присваивания (например, x:=4).
=	Оператор равенства для задания равенств и логических условий (например, a=b), указания областей изменения переменных (например, i=1..5) и значений параметров - опций в функциях и командах (например, color=black).

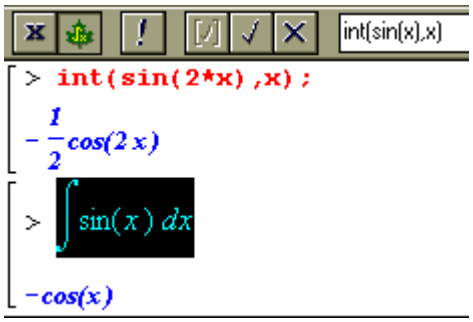
Рабочий лист. Область ввода

Область ввода - это область рабочего листа, в которой пользователь вводит информацию. Эта информация может быть двоякого рода:

- команды и операторы Maple
- текстовые комментарии.

По умолчанию при создании нового рабочего листа устанавливается режим ввода команд и операторов.Указанием на это является приглашение ввода на рабочем листе - символ >. Операция **Edit →Entry Mode** или клавиша **F5** служит для включения/выключения режима ввода текста. Если режим текстового ввода включен, приглашение в виде знака > исчезает, и имеется возможность ввода текстовых комменариев с применением панели форматирования для задания параметров текста. Если режим отменен, то можно вводить математические выражения (Maple-текст).

Режим ввода команд.Команды можно отображать либо в форме синтаксиса языка Maple, либо в виде привычной математической записи. Переключаться между режимами ввода можно с помощью кнопки  на контекстной панели инструментов или используя команды главного меню.

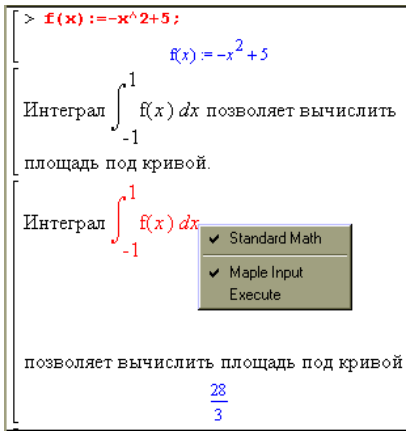


Для ввода результата в стандартной математической форме надо до начала набора команд выполнить команду **Insert → Standart Math Input**, результатом выполнения которой будет смена курсора в области ввода на вопросительный знак. Также появится контекстная панель инструментов ввода команд Maple с полем ввода.

При этом в поле ввода будет отображаться команда в форме синтаксиса Maple, а в области ввода эта же команда будет отображениа в математической нотации (если команда имеет соответствующую математическую запись) по завершении ввода нажатием кнопки  или клавиши **[Enter]**.

Режим ввода текстовых комментариев - это такой режим, при котором любая математическая информация вопринимается как текст (невыполняемый). Для встаки текстового комментария следует выполнить команду **Insert → Text** или нажать кнопку  на основной панели инструментов.

В текстовый комментарий можно вставлять формулы, причем в зависимости от способа встаки эти формулы могут воспринимаются как текст или как вычисляемые выражения.
Для вставки формул-текста достаточно выполнить команду **Insert → Standart Math** или нажать кнопку  на основной панели инструментов. Технология ввода формул аналогична вводу команд Maple в математической нотации (см. выше).



Для вставки в комментарии выполняемых команд необходимо выполнить команду **Insert → Maple Input** (для вставки в форме Maple-команд) или **Insert → Standart Math Input** (для вставки в стандартной математической форме). Введенные таким образом формулы можно в любой момент вычислить, выполнив команду **Execute** контекстного меню

для вставленной формулы или нажатием кнопки на контекстной панели инструментов.

В Maple на контекстрой панели инструментов введена новая кнопка  , которая дублирует команду основногоменю **Edit → Execute → Worksheet (Выполнить всю страницу)**

Переключение между исполняемым выражением или неисполняемым (текстом) можно осуществить с помощью кнопки  контекстной панели инструментов.

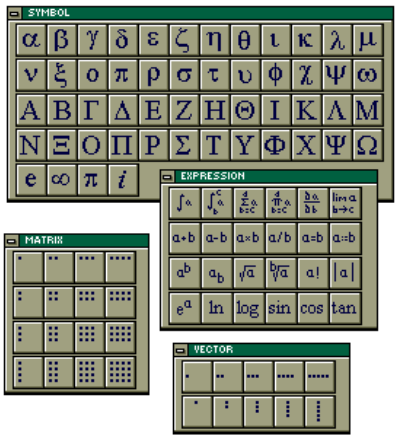
Вернемся к процедуре ввода команд. Если команда достаточно длинная, то Maple автоматически переходит на следующую строку (если команда представлена не как поток символов, а разбита пробелами на слова). Расположенная в нескольких строках команда является одним целым.

В одной строке можно вводить несколько операторов, разделенных точкой с запятой (;) или двоеточием (:). Но иногда удобно задать несколько операторов по одному на строке в области ввода. Перенос команды на новую строку (без выполнения) осуществляется нажатием комбинации клавиш **[Shift+Enter]**. Все введенные таким способом операторы образуют одну группу и последовательно выполняются однократным нажатием клавиши **[Enter]**.

Для того, чтобы ПОСЛЕ выплнения очередной команды, автоматически появлялась новая ячеейка с приглашением Maple для ввода команд, необходимо выполнить команду **Option → Insert Mode**.

Если режим автоматической вставки новой ячейки не включен, вы можете воспользоваться кнопкой  стандартной панели инструментов (ячека появится ниже той, в которой находится курсор) или воспользоваться командой меню **Insert → Execution Group**: для вставки новой ячеки перед той, в которой находится курсор - **Before Cursor** ([Ctrl+K]), и, соответственно, **After Cursor** ([Ctrl+J]) для вставки новой ячейки после курсора.

Пользователь вводит команды Maple в строке ввода, используя клавиатуру. При этом ему приходится запоминать, особенно при первом ознакомлении с системой, достаточно большое их количество. В интерфейс пользователя в версии Maple R5 были введены палитры - небольшие окна с набором шаблонов для ввода определенных команд и объектов. Всего Maple предлагает четыре вида палитр:



- Палитра для ввода символов (команда **View→Palettes→Symbol Palette**) - для ввода греческих букв и некоторых символов.
- Палитра с шаблонами для ввода выражений Maple (команда **View→Palettes→Expression Palette**)
- Палитра с шаблонами для ввода матриц размерности не более (4 x 4) (команда **View→Palettes→Matrix Palette**)
- Палитра с шаблонами для ввода векторов **View→Palettes→Vector Palette**(Появилась в Maple)

В Maple 7 в подменю **View→Paletts** добавлены еще две команды: отобразить все палитры **Show All Palettes** и скрыть все палитры **Hide All Palettes** .

Размеры наборных панелей устанавливаются командами подменю **Options → Palette Size**, при выборе которых они отображаются малого размера **(Small)**, большого размера **(Large)** или размеры выбираются приложением Maple в зависимости от размеров экрана монитора компьютерам **(Best Choise)**.

Рабочий лист. Область вывода.

Результаты выполнения операторов, введенных в область ввода, отображаются в области вывода. Для неграфического вывода Maple позволяет задать три формата:

>f(x):=sin(x);

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

- в виде строковой записи - аналогично формату ввода Maple-команд
- в виде символьной записи - имитирует запись формул в математической нотации
- в стандартном математическом виде, причем в этом случае возможны два варианта вывода: редактируемый (можно выделять отдельные элементы выводящегося выражения, копировать их) и не редактируемый (при выделении области вывода выделяется все выражение целиком).

>Int(f(x),x)=int(f(x),x);

Int(sin(x),x) = -cos(x)

>
Int(f(x),x)=int(f(x),x);

```
  /
 |
 | sin(x) dx = -cos(x)
 |
/
```

>
Int(f(x),x)=int(f(x),x);

$$\int \sin(x)\,dx = -\cos(x)$$

>
Int(f(x),x)=int(f(x),x);

Для установки одного из возможных вариантов отображения информации в области вывода необходимо выполнить команду **Option → Output Display**. В таблице приведены соответствующие подкоманды:

Maple Notation	Строковый вывод результата
Character Notation	Вывод в форме символьной записи
Typeset Notation	не редактируемый вывод в математической форме
Standart Math Nonanion	Редактируемая математическая нотация (установлена по умолчанию)

$$\int \sin(x)\,dx = -\cos(x)$$

Простейшие графики.

Графика Maple реализует все мыслимые (и даже "немыслимые") варианты математических графиков - от построения графиков простых функций в Декартовой и в полярной системах координат до создания реалистических образов сложных пересекающихся в пространстве фигур с их функциональной окраской. Возможны наглядные графические иллюстрации решений самых разнообразных уравнений, включая системы дифференциальных уравнений.

В само ядро Maple встроено ограниченное число функций графики. Это прежде всего функция для построения двумерных графиков (2D-типа) - **plot( )** и функция для построения трехмерных графиков (3D-типа) - **plot3d( )**. Они позволяют строить графики наиболее распространенных типов. Для построения графиков специального типа (например, в виде векторных полей градиентов, решения дифференциальных уравнений, построения фазовых портретов и т.д.) в пакеты расширения системы Maple V включено большое число различных графических функций. Для их вызова необходимы соответствующие указания.

Для построения двумерных графиков служит функция **plot( ).** Она задается в виде:
>plot(f, h, v)
или
>plot(f, h, v, o)
где

f - функция (или функции), чей (чьи) график(и) строятся,
h - переменная с указанием области ее изменения по горизонтали,
v - (не обязательный параметр) заданная опционально переменная с указанием области ее изменения по вертикали,
o - не обязательный параметр) опция или набор опций, задающих стиль построения графика (толщину и цвет кривых, тип кривых, метки на них и т.д.).

Самыми простыми формами задания этой функции служат: **>plot(f, xmin..xmax)** - построение графика функции f, заданной только именем; **>plot(f(x), x=xmin..xmax)** - построение графика функции f(x).

Диапазон изменений независимой переменной **x** задается как **xmin..xmax**, где **xmin** и **xmax** - минимальное и максимальное значение ***x***, **..** (две точки) - составной символ, указывающий на изменение независимой переменной. Разумеется, имя ***x*** здесь дано условно - независимая переменная может иметь любое допустимое имя.

Для построения графиков трехмерных поверхностей Maple имеет встроенную в ядро функцию **plot3d**. Простейшие форматы ее представления:
>plot3d(expr1, x=a..b, y=c..d, p)
>plot3d(f, a..b, c..d, p)

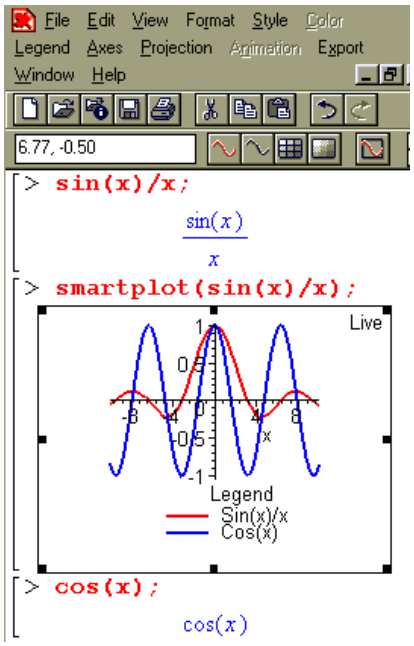
где

f - функция
expr1 - выражение, отражающее зависимость от *x* и *y*,
a и **b** - числовые константы действительного типа,
c и **d** - числовые константы или выражения действительного типа
p - параметры-опции.

Вывод графики в Maple можно осуществлять непосредственно в рабочий лист (режим по умолчанию) или в отдельное окно. Задание соответствующих режимов осуществляется командами **Options →Plot Display → Inline** и **Options →Plot Display → Window**.

Вывод графики в отдельном окне полезен для отображения промежуточных результатов расчетов.

В системе Maple предусмотрено быстрое построение двумерных графиков функций с помощью команды контекстного меню, которое появляется при щелчке ПРАВОЙ кнопкой мыши на выражении в области вывода. (Для этого сначала в области ввода необходимо набрать команду, задающую функцию, и выполнить ее, т.е. нажать клавишу **Enter**). В контекстном меню следует выбрать команду **Plots → 2-D Plot**, после чего Maple с помощью специальной команды **smartplot()** построит график функции, содержащейся в области вывода.



Команда **smartplot()** использует при построении графика функции значения параметров по умолчанию. Но их можно изменить, теперь уже с помощью контекстного меню полученного графика. Обратите внимание на то, что содержимое строки главного меню и контекстной панели инструментов изменилось.

На график, построенный с помощью функции **smartplot()** можно добавить графики других функций простым перетаскиванием с помощью мыши их определения из области вывода в область графика. При этом, если необходимо оставить определения функции в области вывода, перетаскивание осуществляется с нажатой клавишей **Ctrl**. В противном случае перетаскиваемая выделенная часть будет удалена из области вывода.

Аналогичным перетаскиванием можно удалить какую-либо функцию из области графика. Для этого следует подвести указательмыши на линию графика функции и переместитьего за пределы рисунка. При этом в том месте рабочего листа, куда он будет перемещен, появится команда, задающая удаленную функцию.

Основы редактирования.

Операции редактирования содержатся в позиции главного меню **Edit**.

Некоторые операции редактирования:	
Undo Delete Ctrl+Z	Отменить последнюю операцию редактирования.
Cut Ctrl+X	Поместить выделенное в буфер.
Copy Ctrl+C	Скопировать выделенное в буфер.
Copy As Maple Text	Копирование выделения в буфер в формате Maple-текста.
Paste Ctrl+V	Вставить выделенное из буфера в документ.
Paste Maple Text	Вставить данные из буфера в формате Maple-текста.
Delete Paragraph Ctrl+Del	Стирание выделенных объектов.
Select All Ctrl+A	Выделение всех объектов документа.
Find... Ctrl+F5	Найти заданную текстовую или математическую строку.
Input Mode F5	Включение/выключение ввода текстов.
Split or Join	Разделение или объединение объектов.
Execute	Исполнение выделенных или всех строк.
Remove Output	Удаление вывода для выделенных или всех строк.

 **Сохранение результатов работы.**

Рабочие листы Maple сохраняются в файлах с расширением **msw.** Существует возможность сохранить рабочий лист в файлах других форматов, указав в диалоге команде **Save As** соответствующий формат, либо воспользоваться командами подменю **Export As**:

Команда	Расширение	Описание формата
Plain Text	.txt	Обычный текстовый файл.
Maple Text	.txt	Текстовый формат Maple.
Latex	.tex	Формат издательской системы LaTeX.
HTML	.html	Формат HTML.
Maple Explorer	.tex	Формат для обозревателя Maple (позволяет просматривать готовые файлы Maple)
RTF	.RTF	Формат RTF (позволяет вставлять рабочие листы в документы MS Word с сохранением цветовой палитры)

1.2 Пакеты расширений MAPLE .

Некоторые функции системы MAPLE помимо ядра могут находиться в стандартной библиотеке и в пакетах расширения, входящих в базовую поставку системы.

Пакеты в Maple используются для удобства организации работы пользователя. Пакет представляет собой набор команд для решения задач, относящихся к определенным разделам математики, или решения определенных задач графического представления информации, например, пакет **finance** служит для решения задач финансовой математики, в пакете **stats** собраны команды для статистической обработки результатов и т.д.

Перед использованием функций пакетов их надо загрузить отдельно или целым пакетом. Список пакетов расширения можно получить используя команду

>?index[package]

С помощью команды

>?name_package
можно получить информацию о любом пакете расширения и найти список входящих в него функций.

Для обращения к функциям того или иного пакета используется его полная загрузка командой

>with(пакет)
где в качестве параметра указывается имя соответствующего пакета. Может оказаться, что подключаемый пакет содержит команду с таким же именем, что и в ранее подключенном пакете. В этом случае Maple выдает сообщение о переопределении соответствующей команды. Подключив пакет, в дальнейшем пользователь может вызывать все его команды, просто набирая их имя и требуемые для выполнения параметры прямо в области ввода.

Если необходимы несколько конкретных функций пакета, то вместо подключения всего пакета целиком можно подключить только эти требуемые функции, используя команду

>with(пакет,f1,f2,...)
или
>with(пакет,[f1,f2,...])

В дальнейшем загруженные команды также могут быть использованы ссылкой только на их имена, без указания пакета, в котором они находятся.

Если пользователь не желает, чтобы команда постоянно находилась в памяти, ее можно загрузить только на время ее выполнения, после чего она будет выгружена из памяти. Для этого следует указать ее полное имя, состоящее из имени пакета и имени самой команды, в следующем виде:
>имя пакета[имя команды](...);
При следующем обращении к команде ее необходимо снова загрузить одним из трех указанных способов.

Если пользователь не желает, чтобы при загрузке всего пакета (первый способ подключения) отображались все команды подключаемого пакета (список команд некоторых пакетов может занимать целую страницу), то команду **with()** следует завершить двоеточием **(:)**, а не точкой с запятой.

Перечень пакетов расширений и их назначение можно просмотреть **здесь**

1.3 Работа со справочной системой Maple .

Система Maple имеет мощную и подробную справочную базу данных, рассчитанную на удовлетворение всех запросов пользователя на справочную информацию.

В реализации Maple R7 справочная база данных имеет очень удобную древообразную структуру контекстно-зависимого поиска. При этом в ходе поиска информации вначале указываются ее общие признаки, затем более частные - и так до пяти ступеней.

Основные операции по работе со справочной системой Maple R7 сосредоточены в позиции главного меню **Help** (Справка).

Она содержит следующие команды и операции:	
Contexts	Включение раздела справки по контексту.
Help on Context Ctrl+F1	Оперативная справка по контексту.
Topic Search...	Предметный поиск по заданному образцу.
Full Text Search...	Предметный поиск с полным обзором текста справки.
History...	Вывод истории справочного поиска.
Save to Database...	Формирование базы данных.
Remove Topic...	Восстановление базы данных предметного поиска.
Using Help...	Справка по использованию справочной системы.
Ballon Help	Включение всплывающих "шпаргалок".
About Help	Вывод окна с информацией о системе.

Другая система помощи онлайновая (online) позволяет вызвать нужную справку в командном режиме. Для этого достаточно набрать в командной строке слово **help** или **?** с последующим указанием искомого слова, например:

>help sin

или
>? Sin

В данном случае после нажатия клавиши **Enter** будет получена справка по функции **sin**. Такое обращение применяется для библиотечных функций и зарезервированных слов. Недостатком является то, что нужно знать имя оператора или функции, по которой ожидается получение справки.

При этом на экран дисплея будет выведено подробное сообщение (справка) на английском языке о назначении и правилах записи (синтаксисе) соответствующей функции, оператора или пакета применений.

Достаточно полезным средством является выделение и выбор примеров из справок. Для этого используется клавиша **Ins**, которой намечают начало и конец выделений. Можно также выделить пример, перемещая мышь с нажатой левой клавишей. Наконец, используя команду **Copy Examples** в позиции **Edit** главного меню окна справки, можно скопировать в буфер все примеры.

После этого, закрыв окно справки, можно с помощью команды **Edit →Paste** перенести в него имеющиеся в буфере примеры.

Таким образом, можно легко проверить работу примеров по любой функции системы. Как правило, дается несколько примеров на применение каждой функции так что общее число примеров трудно представить, поскольку число функций достигает двух тысяч восьмисот.

Помимо встроенных в ядро функций в систему входят дополнительные функции и пакеты прикладных программ. Чтобы получить помощь по ним, нужно ввести их имя по-

сле вопросительного знака. Например, чтобы получить помощь по функциям пакета расширения системы **plots**, достаточно ввести в командной строке:
> ?plots

А чтобы найти полный перечень пакетов расширения, можно ввести команду:

>?index[package]

Обратите внимание на то, что после этих обращений к справочной системе никакие знаки (двоеточие или точка с запятой) не ставятся. В конце каждой справки приводятся имена связанных с данным объектом слов, а также другие возможные формы получения справок. Используя их, можно более подробно познакомиться с возможностями системы.

Удобно в описываемой версии Maple решен и вопрос о предоставлении оперативной контекстно-зависимой справки по любой функции. Достаточно указать ее маркером и нажать клавиши**Ctrl + F1** (или, что не документировано, просто **F1**). Тут же будет выведено окно справки по данной функции. Можно также выделить интересующее вас слово и получить справку по нему.

1.4 Алфавит Maple-языка и его синтаксис.

Maple-язык является входным языком общения с системой Maple и одновременно языком ее программирования.

Входящие в него средства (прежде всего операторы и функции) подобраны настолько полно и удачно, что при решении подавляющего большинства типовых математических задач от пользователя не требуется знаний даже основ программирования. Достаточно составить алгоритм решения нужной задачи и подобрать набор нужных для его реализации функций и иных средств Maple-языка.

Maple-язык один из самых мощных языков программирования математических задач, намного превосходящий по своим возможностям такие известные языки программирования, как Fortran, Algol, PL. Basic, Pascal и др. Это связано прежде всего с возможностью использования в его программных модулях почти 2800 операторов, команд и функций, входящих в ядро, основную библиотеку и пакеты расширения системы Maple .
В тоже время относящаяся к традиционному программированию часть Maple-языка реализована с помощью довольно скромного набора специальных знаков и зарезервированных слов.

Maple-язык является как бы саморасширяющимся и легко адаптируемым к решению конкретных задач любого пользователя. Свыше 90% всех определений системы Maple , в частности все пакеты расширений и огромная SHARE-библиотека пользователей, написаны на этом языке. Поэтому знание этого языка является определяющим в серьезном изучении системы Maple . Ниже Maple-язык описывается как типичный язык программирования.

Алфавит Maple-языка содержит 26 малых латинских букв от **a** до **z**, 26 больших латинских букв (от **A** до **Z**), 10 арабских цифр (от **0** до **9**) и 32 специальных символа (арифметические операторы **+, -, *,** /, знак возведения в степень ^) и др. Все они будут рассмотрены в данной главе. Имеется пять пар альтернативных символов:

^ и ****** **[** и **(|** **]** и **|)** **{** и **(*** **}** и ***)**

К специальным одиночным и составным знакам относятся элементы синтаксиса языка:

;	Знак фиксации конца выражения с выводом результата на экран
:	Знак фиксации конца выражения, предотвращающий вывод результата на экран.
:=	Оператор присваивания (например, **x:=4).**
=	Оператор равенства для задания равенств и логических условий (например, **a=b**), указания областей изменения переменных (например, **i=1..5**) и значений параметров - опций в функциях и командах (например, **color=black**).
;;	Пустой оператор
::	Указатель типа переменной (например. **n::integer** или **z::complex**).
→	Создание функции (пример. **f:=(x)→tan(x)** или **g:=(x,y)→sin(x)+cos(y/2)**).

Комментарии в программе задаются после символа **#** . В них допустимо использовать все символы кодовых таблиц, что важно при вводе русскоязычных комментариев, использующих символы кириллицы. Применение последних для идентификаторов (имен) объектов недопустимо.

Зарезервированные слова используются для создания условных выражений, циклов, процедур и управляющих команд.

Список зарезервированных слов:

!	**#**	**error**	**catch**	**break**	**by**	**do**	**done**	**elif**	**else**
end	**fi**	**for**	**from**	**finally**	**if**	**in**	**next**	**od**	**proc**
quit	**read**	**return**	**save**	**stop**	**to**	**then**	**while**	**export**	**description**
global	**local**	**module**	**option**	**options**	**try**	**use**			

К зарезервированным относятся также слова, используемые при работе с множествами **(intersect, minus, union)**, логические опраторы **(and, not и or)** и оператор модуля **mod**. Зарезервированные слова имеют особый статус. Их нельзя использовать в качестве идентификаторов для переменных, функции, процедур и т.д.

Совокупность правил, по которым записываются определения всех объектов Maple-языка, называются его синтаксисом. Некоторые особенности синтаксиса полезно знать сразу, в начале освоения системы. Например, то, что знак - (минус) имеет двойное значение. Применительно к одному числу, переменной или выражению он меняет их знак. Однако два знака минус подряд (например, в записи **--3**) задавать нельзя. Другое назначение знака минус - создание операции вычитания, например, **5-2** или **a-b.**

Соответственно двойное назначение имеет и знак **+,** причем число без знака считается положительным, так что **+5=5.**

При вводе действительных чисел с порядком для ввода порядка используется символ **e** (например **10e100** или **10e-100**). Для изменения общепринятого приоритета вычислений

используются круглые скобки, в них же задаются параметры функций и процедур. Более подробно синтаксис Maple-языка рассматривается ниже.

Некоторые операторы представлены двумя символами. Например оператор, присваивания переменным их значения (**: =**) содержит двоеточие и знак равенства, а функциональный оператор, представляющий стрелку (- >), содержит знак минус (-) и "больше" (>). В таких операторах между символами недопустим знак пробела. Однако его можно использовать между отдельными частями выражений; так, **(a+b)/c** эквивалентно **(a + b) / c.**

1.5 Определение функций пользователя.

1) Один из наиболее простых способов задания функции пользователя - присвоение введенной функции (в виде выражения) некоторой переменной:

Name:=выражение

```
> m:=sqrt(x^2+y^2);
```

$$m := \sqrt{x^2+y^2}$$

```
> x:=3:y:=4:
> m;
```

$$\sqrt{25}$$

```
> evalf(m);
```

$$5.000000000$$

Этот прием фактически означает операцию присваивания. Заданный таким образом объект все же не является полноценной функцией пользователя. Прежде всего потому, что в нем используются только глобальные переменные (x и y). Их значения приходится определять заведомо, используя операцию присваивания. Подобные "функции" нальзя ввести в библиотеки Maple.

2) Более гибкий способ задания полноценных функций пользователя - использование функционального оператора →. Его синтаксис:

```
> x:=0;y:=0;
```

$$x := 0$$

$$y := 0$$

```
> m:=(x,y)->sqrt(x^2+y^2);
```

$$m := (x,y) \rightarrow \sqrt{x^2+y^2}$$

```
> m(3,4);
```

$$5$$

```
> m(3.,4.);
```

$$5.000000000$$

```
> x;y;
```

$$0$$

$$0$$

name:=(x,y,...)→Выражение,
где **name** -имя функции (без аргументов),
(x,y,...) - список формальных параметров функции,
Выражение - вид функции (некоторое выражение, зависящее от параметров (x,y,...)).
Вызов функции осуществляется в виде **name (x,y,...)**.Переменные, указанные в списке формальных параметров являются локальными.

При подстановке на их место фактических параметров они сохраняют их значения только в теле функции **Выражение.** За пределами этой функции переменные с этими именами оказываются либо неопределенными, либо сохраняют ранее присвоенное значение.

3) Еще один способ задания функции пользователя базируется на применении команды **unapplay**. Её синтаксис:

name:=unapplay(expr, var1,var2,...),
где **name** - имя функции (без аргументов),
expr - непосредственно выражение функции через аргументы **var1,var2,...**

В качестве **expr** может быть использовано имя переменной, которой присвоено выражение, зависящее от **var1,var2,...** (в этом случае команда **unapplay** преобразует выражение в функцию).

```
> fm:=unapply(sqrt(x^2+y^2),x,y);
```

$$fm := (x,y) \to \sqrt{x^2+y^2}$$

```
> evalf(fm(3,4));
```

$$5.000000000$$

```
> fm(3,4);
```

$$\sqrt{25}$$

```
> fm(sin(x),cos(x));
```

$$\sqrt{\sin(x)^2+\cos(x)^2}$$

```
> simplify(%);
```

$$1$$

```
> x;y;
```

$$x$$

$$y$$

Обращение к функции осуществляется в виде **name(var1,var2,...)**. Переменные , указанные в качестве формальных параметров являются локальными.

Замечание. Для задания сложных выражений можно использовать последовательное определение частей выражения с помощью простого присваивания (1) (при этом переменным не должно быть присвоено никакое значение), а затем, используя (2) или (3), определить сложную функцию, представляющую выражение от ранее определенных (в пункте(1)) выражений.

Задание 1.

1. Запишите комментарий "Основы работы в пакете Maple"
2. Переключитесь в режим набора текста и наберите "Введем функцию $f1(x)=\frac{4x^7}{\sqrt{x^3-3}}$", причем функцию *f1(x)* введите как вычисляемую в математической форме.
3. Определите функцию *f1(x)* через функциональный оператор или команду *unapply*.
4. Вычислите значение функции *f1(x)* при x=1.235
5. Введите выражение $f2(x)=\frac{\cos^5 x+\sin^4 x+\sqrt{tg(x^3)}}{35}$ в виде Maple-нотации, преобразуйте его в обычную математическую форму. В строке редактирования формул измените аргумент косинуса на *2*x*.
6. Постройте график функции используя контекстное меню (быстрое построение графиков). Добавьте на график функцию *sin(2*x)*. Отредактируйте графики по своему усмотрению, используя контекстное меню графиков.
7. Определите функцию *g(x)=ln(1+tan(x))*. Постройте ее график, используя команду *plot* на отрезке *[-10,10]*. Установите опцию discont=true и введите опцию другого-цвета графика.
8. Получите список пакетов расширений системы Maple.
9. Получите справку о пакете *student*.
10. Подключите пакет расширения *student*.
11. Воспользуйтесь функциями *maximize* и *minimize* из этого пакета для вычисления максимума и минимума функции $f3(x)= x^5-5x^4+5x^3+1$.Введите в функции *maximize* и *minimize* отрезок [-1,2] для *x*, на котором необходимо найти максимум и минимум функции *f3(x)*. Воспользуясь справкой, установите опцию, позволяющую определить координату точки *x*, в которой функция достигает экстремума.
12. Посетите сайты Waterloo Maple Inc.(http://www.mapleapps.com - галерею графики и др. на Ваше усмотрение) и Донецкого Университета (http://www.exponenta.ru - найдите список русскоязычной литературы по Maple)

Лекция 2.
Основные объекты и команды Maple .

2.1 Основные объекты (определение, ввод, действия с ними).

Числа.

Maple V работает с числами следующего типа:

- целыми десятичными (О, 1, 123, -456 и т.д.),
- рациональными в виде отношения целых чисел (7/9, -123/127 и т.д.),
- радикалами,
- вещественными с мантиссой и порядком (1.23E5, 123.456E-10)
- комплексными (2+3*I)

Целые числа задаются в виде последовательности цифр от 0 до 9.

```
> 55+7;
                62
> 65*7;
                455
> 45/5;
                9
> 70!;
119785716699\
   96989179607\
   27837216890\
   98736458938\
   14254642585\
   75553628646\
   28009582789\
   84531968000\
   00000000000\
   00
```

Maple может работать с целыми числами произвольной величины, количество цифр практически ограничено числом 2^{28}. Большие числа, которые не помещаются на одной строке области вывода, Maple переносит на следующую строку, используя символ обратного слэша (\).

Кроме стандартных арифметических операций, к которым относятся сложение, вычитание, умножение, деление и вычисление факториала (!), Maple предлагает достаточно большой набор команд, позволяющий выполнить действия, специфичные при обработке целых чисел.

```
> ifactor(54);
                (2) (3)^3
> iquo(14,3);
                4
> irem(14,3);
                2
> igcd(16,36);
                4
```

Получить список всех команд для работы с целыми числами можно, набрав команду: **?integer**. Приведем некоторые из этих команд:

ifactor	разложение на простые множители
iquo	вычисление частного при операции целого деления
irem	вычисление остатка при операции целого деления
igcd	нахождение наибольшего общего делителя двух целых чисел
isprime	проверка, является ли целое число простым

Обыкновенные дроби задаются с помощью операции деления лвух целых чисел.

```
> 64/24;
                8
                -
                3
> 2/3+2/5;
                16
                --
                15
> 2+1/3;
                7
                -
                3
> 4+8/2;
                8
> evalf(7/4,25);
1.750000000000\
   0000000000000
```

Заметим, что Maple автоматически произвадит сокращение дробей. Над обыкновенными дробями можно выполнять все основные арифметические операции. Если при задании дроби ее знаменатель сокращается, то такая "дробь" трактуется программой Maple как целое число. Для преобразования обыкновенной дроби в десятичную служит команда **evalf( )**. Второй параметр этой команды задает число значащих цифр. Заметим, что десятичное представление всего лишь аппроксимация точной величины, представленной обыкновенной дробью, т.е. дробь и ее десятичное представление не являются идентичными объектами Maple.

Радикалы задаются как результат возведения в дробную степень целых или дробных чисел, или вычисления из них же квадратного корня функцией **sqrt( )**, или корня *n*-ой степени функцией **surd(число, n)**.

```
> (3/4)^(2/3);
```

$$\frac{1}{4}3^{\left(\frac{2}{3}\right)}4^{\left(\frac{1}{3}\right)}$$

```
> sqrt(66/9);
```

$$\frac{1}{3}\sqrt{66}$$

```
> surd(75/3,4);
```

$$\sqrt{5}$$

В Maple возведение в степень задается символом *(^)* или последовательностью двух звездочек *(**)*. При задании радикалов также производятся всевозможные упрощения, связанные с вынесением из-под знака радикала максимально возможной величины. Вычисления с целыми, дробями и радикалами являются абсолютно точными, так как при работе с этими объектами Maple не производит никаких округлений в отличие от чисел с плавающей точкой.

Числа с плавающей точкой задаются в виде целой и дробной частей, разделенных десятичной точкой. Их можно представить также, используя так называемую экспоненциальную форму записи (для указания порядка применяется символ *e* или *E*).

```
> 3^5*0.1;
```

$$24.3$$

```
> 3^5*(1/10);
```

$$\frac{243}{10}$$

```
> 2e1+4/5+sqrt(3)*4/5*0.1+
  surd(5,3);
```

$$20.80000000 + .08000000000\sqrt{3} + 5^{\left(\frac{1}{3}\right)}$$

Если в выражении присутствует число с плавающей точкой, то результатом вычисления такого "смешанного" выражения будет число с плавающей точкой. Если в выражении присутствует радикал, то он вычисляется точно, а коэффициент при нем вычисляется либо точно, либо в виде числа с плавающей точкой в зависимости от вида сомножителей.

Комплексные числа. Для мнимой единицы в Maple используется константа *I*. Задание комплексного числа ничем не отличается от его обычного задания в математике (например 2+3*I или *d+k*I*).

Если хотя бы одна из частей комплексного часла (действительная или мнимая) вычисляются в виде числа с плавающей точкой, то и результат будет таким же.

Некоторые команды для работы с комплексными числами:

```
> (2/5+3*I)+(4+1/2*I);
```

$$\frac{22}{5}+\frac{7}{2}I$$

```
> (2/5+3*I)+(4.+1/2*I);
```

$$4.400000000 + 3.500000000\,I$$

```
> Re(2+3*I);
```

$$2$$

```
> conjugate(2+3*I);
```

$$2 - 3\,I$$

```
> argument(%);
```

$$-\arctan\left(\frac{3}{2}\right)$$

Re( )	выделение действительной части комплексного числа
Im( )	выделение мнимой части комплексного числа
argument( )	вычисление аргумента комплексного числа
conjugate( )	построение комплексно-сопряженного числа

Константы.

Maple содержит целый ряд предопределенных *именованных констант* - таких, к значениям которых можно обращаться по имени. Часть этих констант не может быть изменена. К ним относятся:

false	-логическое значение "не истинно";
gamma	-константа Эйлера (0.5772156649..);
infinity	-положительная бесконечность;

true	-логическое значение "истинно";
Catalan	-константа Каталана (0.915965594..);
FAIL	-специальная константа, используется в качестве третьего значения при вычислении функций трехзначной логики;
I	-мнимая единица;
Pi	-константа p=3.141..

Число **e** задается как **exp(1)**.

```
> evalf(Pi);
                3.141592654
> Digits:=30;
                Digits := 30
> evalf(Pi);
   3.14159265358979323846264338328
```

Константы, значения которых могут быть переопределены:

Digits	-задает число значащих цифр для чисел с плавающей точкой (по умолчанию 10);
Order	-определяе количество членов в разложении функции в ряд Тейлора (по умолчанию 6);

Посмотреть все константы, определенные в Maple, можно, исполнив команду: **?ininame**

Кроме перечисленных на странице Справки констант все переменные, имена которых начинаются с **_Env** , по умолчанию являются системными константами Maple.

Строки.

Строки - любой набор символов, заключенный в ДВОЙНЫЕ кавычки. Длина строки в Maple практически не ограничена и может достигать на 32-битных компьютерах длины 268 435 439 символов.

```
> "слоно"||"потам";
              "слонопотам"
> s1:="ба":
  s2:="баааа":
  s3:="ааах":
  cat(s1,s2,s3);
              "бабаааааах"
> "alfa"[3..4];
                   "fa"
> s2[2..4];
                  "ааа"
```

Если идут подряд две строки, разделенные символами-разделителями (пробел, табуляция или переход на новую строку), то в обрасти вывода эти две строки соединяются в одну, причем, без пробела между ними.

Соединение строк можно осуществить и с помощью операции конкатенации ||, или обращением к функции **cat(строка 1, строка 2)**.

Строка представляется как одномерный массив, поэтому можно использовать индекс (или диапазон) для выделения элемента (или подстроки) из заданной строки.

Переменные, неизвестные и выражения.

Каждая **переменная** Maple имеет имя, представляющее последовательность латинских символов, начинающихся с буквы, причем строчные и прописные буквы считаются различными. Кроме букв в именах переменных могут использоваться цифры и знак подчеркивания, но ПЕРВЫМ символом имени должна быть БУКВА.

```
> Catalan:=8;
Error, attempting to assign
to `Catalan` which is
protected

> `my definition`:=35;
        my definition := 35
```

В качестве имен запрещено использовать зарезервированные слова языка Maple (см. Лекцию 1), а также защищенные слова (справку по ним можно получить, выполнив команду **?protect**).

В Maple можно задавать переменные с именами, содержащими пробелы, но для этого их следует заключать в обратные кавычки.

Выражение представляет собой комбинацию имен переменных, чисел и, возможно, других объектов Maple, соединенных знаками допустимых операций.

```
> sqrt(exp(sin(x*y)));
                  sin(x y)
                 e
```

Если в выражении используется переменная, которой не присвоено никакого числового значения, то такая переменная рассматривается системой Maple как **неизвестная величина**, а выражение, содержащее неизвестные, называется **символьным выражением.** Именно для работы с такими выражениями прежде всего и разрабатывался Maple.

Важной операцией в Maple, связанной с выражениями, является операция **присваивания (:=)**. Она имеет следующий синтаксис:
переменная := выражение;
Здесь в левой части задается имя переменной, а в правой - любое выражение, которое может быть числовым, символьным или просто другой переменной.

Переменные позволяют хранить и обрабатывать разнообразные типы данных. При этом по умолчанию переменная Maple имеет тип symbol, представляющий символьную переменную, и ее значением является ее собственное имя. При присвоении переменной какого-нибудь значения, ее тип изменяется на тип присвоенного ей значения.

Задание 1.

1. Запишите комментарий "Основы работы в пакете Maple"
2. Переключитесь в режим набора текста и наберите "Введем функцию $f1(x)=\frac{4x^7}{\sqrt{x^3-3}}$", причем функцию *f1(x)* введите как вычисляемую в математической форме.
3. Определите функцию *f1(x)* через функциональный оператор или команду *unapply*.
4. Вычислите значение функции *f1(x)* при x=1.235
5. Введите выражение $f2(x)=\frac{\cos^5 x+\sin^4 x+\sqrt{tg(x^3)}}{35}$ в виде Maple-нотации, преобразуйте его в обычную математическую форму. В строке редактирования формул измените аргумент косинуса на *2*x*.
6. Постройте график функции используя контекстное меню (быстрое построение графиков). Добавьте на график функцию *sin(2*x)*. Отредактируйте графики по своему усмотрению, используя контекстное меню графиков.
7. Определите функцию *g(x)=ln(1+tan(x))*. Постройте ее график, используя команду *plot* на отрезке *[-10,10]*. Установите опцию discont=true и введите опцию другого цвета графика.
8. Получите список пакетов расширений системы Maple.

9. Получите справку о пакете *student*.
10. Подключите пакет расширения *student*.
11. Воспользуйтесь функциями *maximize* и *minimize* из этого пакета для вычисления максимума и минимума функции $f3(x)= x^5-5x^4+5x^3+1$.Введите в функции *maximize* и *minimize* отрезок [-1,2] для *x*, на котором необходимо найти максимум и минимум функции *f3(x)*. Воспользуясь справкой, установите опцию, позволяющую определить координату точки *x*, в которой функция достигает экстремума.
12. Посетите сайты Waterloo Maple Inc.(http://www.mapleapps.com - галерею графики и др. на Ваше усмотрение) и Донецкого Университета (http://www.exponenta.ru - найдите список русскоязычной литературы по Maple)

Лекция 2.
Основные объекты и команды Maple .

2.1 Основные объекты (определение, ввод, действия с ними).

- **Числа.**
- **Константы.**
- **Строки.**
- **Переменные," неизвестные и выражения.**
- **Последовательности, списки, множества.**
- **Массивы.**
- **Таблицы.**

 **Числа.**

Maple V работает с числами следующего типа:

- целыми десятичными (0, 1, 123, -456 и т.д.),
- рациональными в виде отношения целых чисел (7/9, -123/127 и т.д.),
- радикалами,
- вещественными с мантиссой и порядком (1.23E5, 123.456E-10)
- комплексными (2+3*I)

Целые числа задаются в виде последовательности цифр от 0 до 9.

```
> 55+7;
                62
> 65*7;
                455
> 45/5;
                9
> 70!;
11978571669\
   96989179607\
   27837216890\
   98736458938\
   14254642585\
   75553628646\
   28009582789\
   84531968000\
   00000000000\
   00
```

Maple может работать с целыми числами произвольной величины, количество цифр практически ограничено числом 2^{28}. Большие числа, которые не помещаются на одной строке области вывода, Maple переносит на следующую строку, используя символ обратного слэша (\).

Кроме стандартных арифметических операций, к которым относятся сложение, вычитание, умножение, деление и вычисление факториала (!), Maple предлагает достаточно большой набор команд, позволяющий выполнить действия, специфичные при обработке целых чисел.

```
> ifactor(54);
                (2) (3)^3
> iquo(14,3);
                4
> irem(14,3);
                2
> igcd(16,36);
                4
```

Получить список всех команд для работы с целыми числами можно, набрав команду: **?integer**. Приведем некоторые из этих команд:

ifactor	разложение на простые множители
iquo	вычисление частного при операции целого деления
irem	вычисление остатка при операции целого деления
igcd	нахождение наибольшего общего делителя двух целых чисел
isprime	проверка, является ли целое число простым

Обыкновенные дроби задаются с помощью операции деления лвух целых чисел.

```
> 64/24;
                8
                -
                3
> 2/3+2/5;
                16
                --
                15
> 2+1/3;
                7
                -
                3
> 4+8/2;
                8
> evalf(7/4,25);
1.750000000000\
   0000000000000
```

Заметим, что Maple автоматически произвадит сокращение дробей. Над обыкновенными дробями можно выполнять все основные арифметические операции. Если при задании дроби ее знаменатель сокращается, то такая "дробь" трактуется программой Maple как целое число. Для преобразования обыкновенной дроби в десятичную служит команда **evalf( )**. Второй параметр этой команды задает число значащих цифр. Заметим, что десятичное представление всего лишь аппроксимация точной величины, представленной обыкновенной дробью, т.е. дробь и ее десятичное представление не являются идентичными объектами Maple.

Радикалы задаются как результат возведения в дробную степень целых или дробных чисел, или вычисления из них же квадратного корня функцией **sqrt( )**, или корня *n*-ой степени функцией **surd(число, n)**.

```
> (3/4)^(2/3);
```

$$\frac{1}{4}3^{\left(\frac{2}{3}\right)}4^{\left(\frac{1}{3}\right)}$$

```
> sqrt(66/9);
```

$$\frac{1}{3}\sqrt{66}$$

```
> surd(75/3,4);
```

$$\sqrt{5}$$

В Maple возведение в степень задается символом *(^)* или последовательностью двух звездочек *(**)*. При задании радикалов также производятся всевозможные упрощения, связанные с вынесением из-под знака радикала максимально возможной величины. Вычисления с целыми, дробями и радикалами являются абсолютно точными, так как при работе с этими объектами Maple не производит никаких округлений в отличие от чисел с плавающей точкой.

Числа с плавающей точкой задаются в виде целой и дробной частей, разделенных десятичной точкой. Их можно представить также, используя так называемую экспоненциальную форму записи (для указания порядка применяется символ *e* или *E*).

```
> 3^5*0.1;
```

$$24.3$$

```
> 3^5*(1/10);
```

$$\frac{243}{10}$$

```
> 2e1+4/5+sqrt(3)*4/5*0.1+
  surd(5,3);
```

$$20.80000000 + .08000000000\sqrt{3} + 5^{\left(\frac{1}{3}\right)}$$

Если в выражении присутствует число с плавающей точкой, то результатом вычисления такого "смешанного" выражения будет число с плавающей точкой. Если в выражении присутствует радикал, то он вычисляется точно, а коэффициент при нем вычисляется либо точно, либо в виде числа с плавающей точкой в зависимости от вида сомножителей.

Комплексные числа. Для мнимой единицы в Maple используется константа *I*. Задание комплексного числа ничем не отличается от его обычного задания в математике (например 2+3*I или *d+k*I*).

Если хотя бы одна из частей комплексного часла (действительная или мнимая) вычисляются в виде числа с плавающей точкой, то и результат будет таким же.

Некоторые команды для работы с комплексными числами:

```
> (2/5+3*I)+(4+1/2*I);
```

$$\frac{22}{5}+\frac{7}{2}I$$

```
> (2/5+3*I)+(4.+1/2*I);
```

$$4.400000000 + 3.500000000\,I$$

```
> Re(2+3*I);
```

$$2$$

```
> conjugate(2+3*I);
```

$$2-3\,I$$

```
> argument(%);
```

$$-\arctan\left(\frac{3}{2}\right)$$

Re( )	выделение действительной части комплексного числа
Im( )	выделение мнимой части комплексного числа
argument( )	вычисление аргумента комплексного числа
conjugate( )	построение комплексно-сопряженного числа

Константы.

Maple содержит целый ряд предопределенных *именованных констант* - таких, к значениям котрых можно обращаться по имени. Часть этих констант не может быть изменена. К ним относятся:

false	-логическое значение "не истинно";
gamma	-константа Эйлера (0.5772156649..);
infinity	-положительная бесконечность;

true	-логическое значение "истинно";
Catalan	-константа Каталана (0.915965594..);
FAIL	-специальная константа, используется в качестве третьего значения при вычислении функций трехзначной логики;
I	-мнимая единица;
Pi	-константа p=3.141..

Число **e** задается как **exp(1)**.

```
> evalf(Pi);
                3.141592654
> Digits:=30;
                Digits := 30
> evalf(Pi);
   3.14159265358979323846264338328
```

Константы, значения которых могут быть переопределены:

Digits	-задает число значащих цифр для чисел с плавающей точкой (по умолчанию 10);
Order	-определяе количество членов в разложении функции в ряд Тейлора (по умолчанию 6);

Посмотреть все константы, определенные в Maple, можно, исполнив команду: **?ininame**

Кроме перечисленных на странице Справки констант все прерменные, имена которых начинаются с **_Env** , по умолчанию являются системными константами Maple.

Строки.

Строки - любой набор символов, заключенный в ДВОЙНЫЕ кавычки. Длина строки в Maple практически не ограничена и может достигать на 32-битных компьютерах длины 268 435 439 символов.

```
> "слоно"||"потам";
            "слонопотам"
> s1:="ба":
  s2:="баааа":
  s3:="аах":
  cat(s1,s2,s3);
             "бабаааааах"
> "alfa"[3..4];
                  "fa"
> s2[2..4];
                 "aaa"
```

Если идут подряд две строки, разделенные символами-разделителями (пробел, табуляция или переход на новую строку), то в обрасти вывода эти две строки соединяются в одну, причем, без пробела между ними.

Соединение строк можно осуществить и с помощью операции конкатенации ||, или обращением к функции **cat(строка 1, строка 2)**.
Строка представляется как одномерный массив, поэтому можно использовать индекс (или диапазон) для выделения элемента (или подстроки) из заданной строки.

Переменные, неизвестные и выражения.

Каждая **переменная** Maple имеет имя, представляющее последовательность латинских символов, начинающихся с буквы, причем строчные и прописные буквы считаются различными. Кроме букв в именах переменных могут использоватьсчя цифры и знак подчеркивания, но ПЕРВЫМ символом имени должна быть БУКВА.

```
> Catalan:=8;
Error, attempting to assign
to `Catalan` which is
protected

> `my definition`:=35;
        my definition := 35
```

В качестве имен запрещено использовать зарезервированные слова языка Maple (см. Лекцию 1), а также защищенные слова (справку по ним можно получить, выполнив команду **?protect**).

В Maple можно задавать переменные с именами, содержащими пробелы, но для этого их следует заключать в обратные кавычки.

Выражение представляет собой комбинацию имен переменных, чисел и, возможно, других объектов Maple, соединенных знаками допустимых операций.

```
> sqrt(exp(sin(x*y)));
                  sin(x y)
                 e
```

Если в выражении используется переменная, которой не присвоено никакого числового значения, то такая переменная рассматривается системой Maple как **неизвестная величина**, а выражение, содержащее неизвестные, называется **символьным выражением.** Именно для работы с такими выражениями прежде всего и разрабатывался Maple.

Важной операцией в Maple, связанной с выражениями, является операция **присваивания (:=)**. Она имеет следующий синтаксис:
переменная := выражение;
Здесь в левой части задается имя переменной, а в правой - любое выражение, которое может быть числовым, символьным или просто другой переменной.

Переменные позволяют хранить и обрабатывать разнообразные типы данных. При этом по умолчанию переменная Maple имеет тип symbol, предсталяющий символьную переменную, и ее значением является ее собственное имя. При присвоении перемнной какого-нибудь значения, ее тип изменяется на тип присвоенного ей значения.

Последовательности, списки, множества.

Последовательность - это ряд выражений, разделенных запятыми и завершенных фиксатором:

```
> sq1:=1,2,sqrt(3),z,
  sin(Pi/2);
        sq1 := 1, 2, √3, z, 1
> sq2:=1,x,2,x,3,ln(y)+5;
     sq2 := 1, x, 2, x, 3, ln(y) + 5
```

Последовательность является базовым объектом Maple, на основе которого строятся другие сложные объекты. Последовательность сохраняет порядок следования выражений и может содержать повторяющиеся элементы.

Ее можно мыслить как последовательность выражений, перенумерованных натуральными числами,начиная с единицы.

В связи с этим, можно получить значение любого элемента последовательности, используя индексную форму записи - после имени переменной в квадратных скобках задать индекс элемента. Однако присвоить новое значение элементу последовательности с использованием индексной формы обращения нельзя.

```
> r4:=a,b,c;
  r4[2];
          r4 := a, b, c
                b
> r4[2]:=6;
Error, cannot assign to an
expression sequence
```

Использование последовательностей.

```
> r1:=sin(x),x;
  int(r1);
                r1 := sin(x), x
                  -cos(x)
> r2:=1,2,3;
  r3:=y||r2;
                  r2 := 1, 2, 3
                r3 := y1, y2, y3
> f,g,h:=r2;
  h;
                f, g, h := 1, 2, 3
                        3
```

1. Если этот тип данных передается в качестве параметра функции Maple, то каждый элемент рассматривается как соответствующий параметр функции. (Пример с вычислением интеграла.)

2. Использование последовательности в качестве одного из операндов приводит к выполнению этой операции для каждого ее элемента в качестве соответствующего операнда и формирования новой последовательности.(Пример с созданием новой последовательности)

3. Использование последовательности переменных в ЛЕВОЙ части операции присваивания и последовательности значений в ПРАВОЙ приводит к множественному присваиванию с помощью одной операции.

Создание последовательностей. Для создания длинных последовательностей, элементы которых подчиняются некоей закономерности, можно использовать каманду **seq( )** и операцию повторения **$**.

1) Команда **seq( )** имеет две формы:
seq(f,i=m..n)
seq(f,i=X)
В них **f** - выражение, зависящее от переменной, имя которой определяется параметром **i**,
m и **n** -числа, определяющие диапазон изменения переменной **i** с шагом 1,
X может быть списком, множеством, суммой, произведением или строкой. (В последнем случае переменная **i** последовательно принимает значения, равные символам строки).

```
> seq(sin(Pi*i/6),i=0..4);
        0, 1/2, 1/2 √3, 1, 1/2 √3
> seq(t[k],k=3..6);
                t3, t4, t5, t6
> seq(cos(u),u=r4);
          cos(a), cos(b), cos(c)
```

```
> $3..7;
                3, 4, 5, 6, 7
> w^3 $ w=1..4;
                1, 8, 27, 64
> G[t] $ t=1..3;
                G1, G2, G3
> p $5;
              p, p, p, p, p
```

2) Операция повторения **$** может быть унарной и бинарной. В первом случае она применяется к диапазону **m..n**, где **m** и **n** - числа; а во втором случае первым операндом является выражение, зависящее от переменной, которая указана во втором операнде, содержащем также операцию диапазон.

Список - упорядоченная последовательность выражений, заключенная в квадратные скобки.
Множество - неупорядоченная последовательность выражений, заключенная в фигурные скобки.

```
> L:=[a,b,a,c,a,d];
          L := [a, b, a, c, a, d]
> S:={a,b,a,c,a,d};
            S := {a, b, c, d}
```

Свойства списка:

- В списке все повторяющиеся элементы существенны, т.е. список может содержать повторяющиеся элементы, стоящие на разных местах.

```
> L[3];L[3]:=10;L;
                    10
                  L3 := 10
            [a, b, 10, c, a, d]
```

- Списки сохраняют порядок своих элементов, поэтому с помощью индекса можно получить значение любого элемента списка
- Индексной форме элемента списка МОЖНО присвоить новое значение, изменив тем самым значение этого элемента.

Свойства множества:

- В множестве все повторяющиеся элементы игнорируются, т.е. повторяющимся элементам последовательности во множестве будет соответствовать один элемент.

```
> S[2];
                    b
> S[3]:=5;
Error, cannot assign to a
set

> (S minus {c}) union {10};
              {10, a, b, d}
```

- При создании множества порядок его элементов определяется системой Maple и может меняться от сеанса к сеансу при работе с одним и тем же рабочим листом. Но с помощью индекса можно получить значение любого элемента списка, если "выцепить" этот элемент сразу после вывода множества в области вывода.

- Индексной форме элемента списка НЕЛЬЗЯ присвоить новое значение. Чтобы изменить элемент множества, его следует удалить операцией minus, а затем добавить новый элемент операцией union(разность и объединение).

При выборе нескольких элементов списка или множества и помощью индекса можно использовать объект диапазон, приче положительные значения индекса соответствуют отсчету элементов слева направо, а отрицательные -справа налево. Все элементы списка или множества можно вывести, указав в качестве индекса диапазон [1..-1]. (Значение индекса равное -1 соответствует последнему элементу списка или множества.)

Команды для структурной обработки списков и множеств. Часто возникает необходимость выполнить какую-либо команду или вычислить функцию применительно к каждому элементу списка или множества; выделить элементы, удовлетворяющие некоторому условию и др.

Команды **map( )** и **map2( )** позволяют применить функцию или команду, заданную первым параметром, ко свем элементам списка или множества, возвращая, соответственно, список или множество. Синтаксис этих команд:
map(функция, список|множество, [пар2, пар3,..., парN]);
map2(функция, пар1,список|множество, [пар3,..., парN]);

```
> map(int,[x,x^2,ln(x)],x);
                  [1/2 x^2, 1/3 x^3, x ln(x) - x]
> map(x->x^k,{x,y,z});
                  {y^k, z^k, x^k}
> map2(diff, x*y*z, [x,y,z]);
                  [y z, x z, x y]
```

Если для выполнения функции, заданной первым параметром команды **map ( )**, необходимы дополнительные параметры, то их следует задать после списка или множества. В команде **map2( )** элементы списка или множества передаются в качестве второго параметра функции, определенной первым параметром.

```
> seq(ln(dd),dd=[a,b,c,d]);
        ln(a), ln(b), ln(c), ln(d)
> add(ln(dd),dd=[a,b,c,d]);
      ln(a) + ln(b) + ln(c) + ln(d)
> mul(ln(dd),dd=[a,b,c,d]);
      ln(a) ln(b) ln(c) ln(d)
```

Команда **seq( )** формирования последовательности может также поэлементно обрабатывать список или множество, формируя при этом (в зависимости от вида скобок или их присутствия/отсутствия) список, множество или последовательность.

Команда **add( )** формирует сумму элементов списка или множества, команда **mul( )** - произведение элементов списка или множества.

```
> cond:=x->is(x^2>1);
          cond := x → is(1 < x^2)
> ls:=[1,Pi,exp(1),0];
              ls := [1, π, e, 0]
> select(cond,ls);
                 [π, e]
> remove(cond,ls);
                 [1, 0]
> selectremove(cond,ls);
              [π, e], [1, 0]
```

Maple позволяет выбирать из списка или множества элементы, удовлетворяющие некоторому условию. Для этого прежде всего необходимо определить функцию-условие, результатом работы которой будет булево значение *true* или *false*.
А затем можно воспользоваться командой
select(функция-условие, список|множество)
Действие команды
remove(функция-условие, список|множество)
противоположно действию команды **select( )**.

Она возвращает список/множество, состоящий из элементов, не удовлетворяющих условию булевой функции.

Выполнить обе операции одновременно позволяет команда **selectremove( )**, которая возвращает последовательность двух списков, первый из которых представляет результат выполнения команды**select( )**, а второй - команды**remove( )**.

Линейное объединение двух списков можно реализовать с помощью команды **op(выражение|список|множество)**. Эта команда предназначена, вообще говоря, для выделения операндов выражения в виде последовательности. Так как операндами списка или множества являются его элементы, то результатом работы этой команды со списком| множеством будет последовательность, которую можно "оформить" с помощью скобок в виде множества или списка.

Более сложные объединения списков реализуются командой **zip( )**, имеющей следующий синтаксис:
zip(бинарная функция, список1, список2);
zip(бинарная функция, список1, список2, значение);

```
> zip((x,y)->x*y,[a,b,c],[3,4,5]);
              [3 a, 4 b, 5 c]
> zip((x,y)->x*y,[a,b,c],[3,4],0);
              [3 a, 4 b, 0]
```

Первая из команд работает так: бинарная (двух аргументов) функция выполняется, последовательно используя в качестве своих параметров элементы двух списков, формируя новый список из вычисленных значений.
Длина полученного списка равна длине наименьшего из переданных (список1 и список2).

Вторая команда работает аналогично, но заданный четвертый параметр (значение) используется в качестве элементов списка наименьшей длины при продолжении последовательного выбора элементов списка большей длины. Т.е. короткий список дополняется

элементами "значение" до длины большего списка. Таким образом, команда **zip( )** формирует список , длина которого равна длине наибольшего списка-параметра.

Массивы.

Массив является дальнейшим развитием концепции списка. Если список можно мыслить как перенумерованную последовательность, индексы которой могут принимать только положительные целые значения (нумерация обязательно начинается с единицы), то в массиве каждый элементтакже связан с индексом, однако не ограничен одной размерностью (массив может иметь много размерностей, каждую со своим индексом), причем индексы могут принимать наряду с целыми положительными и отрицательные значения, и нуль.

Создание массивов.. Создать массив в Maple можно несколькими способами. Простейший из них - использование функции **array( )** стандартной библиотеки. Синтаксис создания массива следующий:

array(границы, список, опции);

Параметр **границы** представляет диапазон(ы) изменения индекса(ов) массива (для многомерного массива диапазоны задаются через запятую).

```
> A:=array(1..2,identity);
```

$$A := \mathrm{array}(identity, 1 .. 2, [\,])$$

```
> eval(A);
```

$$[1, 1]$$

```
> M:=array(-1..0,-2..-1,
  [[a,a],[b,b]]);
```

$$M := \mathrm{array}(-1 .. 0, -2 .. -1, [\,(-1, -2) = a,\ (-1, -1) = a,\ (0, -2) = b,\ (0, -1) = b\,])$$

Список задает значения элементов массива, причем для двумерного массива элементами списка являются списки, т.е. этот параметр является списком скписков. Параметр **опции** используется для задания массивов специального вида. Этот параметр может принимать следующие значения: symmetric, antisymmetric, sparse, identity, diagonal , для задания , соответственно, симметричных, антисимметричных, разреженных, единичных и диагональных массивов.

Все пераметры этой функции являются необязательными, однако, либо параметр **границы**, либо **список** значений должен обязательно присутствовать.

```
> V:=vector(3,[ff,gg,hh^2]);
```

$$V := [ff, gg, hh^2]$$

```
> M:=matrix(2,2,[x,x^2,2*x,3*
  z]);
```

$$M := \begin{bmatrix} x & x^2 \\ 2x & 3z \end{bmatrix}$$

Для задания вектора и матрицы можно использовать соответственно команды
vector( ) и **matrx( )**. Синтаксис этих команд:
vector(n,[элемент1, элемент2, ...]);
matrx(n, m, [элемент1, элемент2, ...])
Здесь целые величины **n** и **m** задают размерности вектора и матрицы, а значения их элементов задаются в виде простого списка.

```
> M1:=array(1..4);
```

$$M1 := \text{array}(1 .. 4, [\,])$$

```
> M1[1]:=x;M1[2]:=y;
  M1[3]:=z;M1[4]:=t;
```

$$M1_1 := x$$
$$M1_2 := y$$
$$M1_3 := z$$
$$M1_4 := t$$

```
> M1;
```

$$M1$$

```
> eval(M1);
```

$$[x, y, z, t]$$

```
> M2:=array(1..4,[a,b,c,d]);
```

$$M2 := [a, b, c, d]$$

```
> M2;
```

$$M2$$

```
> print(M2);
```

$$[a, b, c, d]$$

Значения элементов вектора или матрицы не обязательно задавать при создании этих объектов. Можно позднее с помощью индексной ссылки на элементы вектора или матрицы (в квадратных скобках) присвоить им значения.

Для отображения содержимого вектора или матрицы используется команда **print(V)** или **eval(V)**, так как переменная, содержащая сложные объекты, вычисляется не полностью, а только до своего имени.

```
> M3:=matrix(2,2,[1,2,3,4]);
```

$$M3 := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> M4:=matrix(2,2,[a,b,c,d]);
```

$$M4 := \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```
> M34:=M3+M4;
  eval(M34);
  print(M34);
```

$$M34 := M3 + M4$$
$$M3 + M4$$
$$M3 + M4$$

```
> evalm(M34);
```

$$\begin{bmatrix} 1+a & 2+b \\ 3+c & 4+d \end{bmatrix}$$

```
> evalm(M3&*M4);
```

$$\begin{bmatrix} a+2c & b+2d \\ 3a+4c & 3b+4d \end{bmatrix}$$

Для отображения результатов ВЫЧИСЛЕНИЙ с матрицами и векторами служит команда **evalm(V)**. Эта команда отображает результат вычислений на уровне элементов. Над матрицами и векторами (без подключения специальных пакетов **linalg** и **LinearAlgebra**) возможны любые действия, заданные операторами сложения**(+)**, вычитания **(-)**, некоммутативного умножения **(&*)**, деления **(/)** и возведения в степень **(^)**.

Пакеты **linalg** и **LinearAlgebra** содержат множество функций для структурных преобразований матриц и векторов, команд выполнения матричных операций. Перечень всех доступных команд можно найти на страницах справки, отображаемых командами **?linalg** (для пакета **linalg**) и **?LAOverview** (для нового пакета **LinearAlgebra**, а дальше - путешествуйте по ссылкам, находящимся на этой странице).

Таблицы.

Таблица является дальнейшим развитием массива, как структуры данных. В ней в качестве индекса можно использовать не только целые числа, а всё, что угодно. Для создания таблицы используется функция **table( )**. Её параметрами являются индексная функция (см. параметр **опция** для команды **array( )**)и список или множество пар **индекс=значение**. Первый параметр необязательный.

```
> Data:=table([Аня=25, Катя=27.6, Лена =23]);
```

$$Data := \text{table}([Аня = 25, Катя = 27.6, Лена = 23])$$

```
> Data[Катя];
```

$$27.6$$

Таблицы достаточно удобный объект, когда в одном "массиве" надо хранить данные, относящиеся к какому-либо реальному объекту, и ссылаться на них по индексам, предсавляющим естественную запись их наименований. Maple выводит элементы таблицы в произвольном порядке, но, используя соответствующий индекс, можно получит значение любого элемента таблицы.

2.2 Внутренняя структура объектов Maple.

Каждое алгебраическое выражение хранися системой Maple в виде древовидной структуры, обеспечивая тем самым доступ к любому ее члену или подвыражению, а также позволяя выполнять над ними разнообразные символьные преобразования. В представлении этой структуры каждый объект Maple делится на подобъекты первого уровня, которые, в свою очередь, такде делятся на подобъекты и т.д.

Команды, позволяющие выделять части объектов:

rhs(уравн)	Выделение правой части уравнения (или конца диапазона)
lhs(уравн)	Выделение левой части уравнения (или начала диапазона)
numer(дробь)	Выделение числителя числовой или алгебраической дроби
denum(дробь)	Выделение знаменателя числовой или алгебраической дроби
nops(выр)	Определяет количество операндов в выражении
op(выр) **op(n,выр)**	Выдает операнды выражения в виде списка, Извлекает n-ый операнд выражения
select(б ф, выр)	Выделяет в выражении операнды, для которых булева функция дает значение *true*
remove(б ф, выр)	Выделяет в выражении операнды, для которых булева функция дает значение *false*
indets(выр, тип)	Выделяет в выражении подвыражения заданного типа('*', '+' ...)

Познакомимся с этими командами более подробно.

Уравнение представляется в виде двух выражений, соединенных знаком равенства. Его не следует путать с операцией присваивания (:=). Уравнение является объектом Maple и служит для задания действительных уравнений. Его можно использовать в правой части операции присваивания, именуя тем самым уравнение.

```
> eq:=x^2+sin(x)=cos(x)-1;
```

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

```
> lhs(eq); rhs(eq);
```

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

```
> lhs(3..9); rhs(3..9);
```

$$3$$
$$9$$

Команды **lhs( )** и **rhs( )** позволяют выделить левую и правую части уравнений (или, применительно к диапазонам, их начало и конец).

Команды **numer( )** и **denom( )** выделяют, соответственно, числитель и знаменатель дроби, причем перед выделением этих частей дробей Maple осуществляет их упрощение, приводя к нормальной форме.

```
> dr:=(sin(x)+x/y)/
  (cos(x)/sin(x)+2);
```

$$dr := \frac{\sin(x) + \frac{x}{y}}{\frac{\cos(x)}{\sin(x)} + 2}$$

```
> numer(dr);
```

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

```
> denom(dr);
```

$$y\,(\cos(x) + 2\sin(x))$$

```
> expr:=x^2/sin(x)+
        3*sqrt(x)*sin(x);
```

$$expr := \frac{x^2}{\sin(x)} + 3\sqrt{x}\,\sin(x)$$

```
> nops(expr);
```

$$2$$

```
> op(expr);
```

$$\frac{x^2}{\sin(x)},\ 3\sqrt{x}\,\sin(x)$$

```
> a2:=op(2,expr);
```

$$a2 := 3\sqrt{x}\,\sin(x)$$

```
> op(3,op(2,expr));op(3,a2);
```

$$\sin(x)$$
$$\sin(x)$$

Команда **nops( )** определяет количество операндов в выражении, а команда **op( )** выдает их в виде последовательности выражений. Эта же команда позволяет извлечь конкретный операнд выражения, указав в качестве первого параметра его порядковый номер. Отметим, что операндами списка или множества являются его элементы.

```
> cd:=x->evalb(is (x<0)=true);
```

$$cd := x \rightarrow \mathrm{evalb}(\mathrm{is}(x < 0) = true)$$

```
> select(cd,sin(x)-5-x^2);
```

$$-5$$

```
> remove(cd,sin(x)-5-x^2);
```

$$\sin(x) - x^2$$

```
> selectremove(cd,sin(x)-5-x^2);
```

$$-5,\ \sin(x) - x^2$$

Команды **select( )** и **remove( )** были ранее рассмотрены в разделе ???. Для работы этих команд необходимо в качестве первого параметра указать булеву функцию, результатом работы которой будет ответ *true* или *false*. Maple предлагает большое количество булевых функций, которые можно использовать в командах**select( )** и **remove( )** для работы со структурой выражений.

```
> ex1:=(x+3)*exp(cos(x+2));
```

$$ex1 := (x+3)\,e^{\cos(x+2)}$$

```
> has(ex1,cos);has(ex1,x+2);
```

$$true$$
$$true$$

```
> remove(has,ex1,exp);
```

$$x+3$$

Команда **has(выражение, подвыражение)** определяет, содержится ли некоторое подвыражение в заданной выражении.
Команда **has( )** понимает только те подвыражения, которые могут быть определены с помощью команды **op( ).** Для выделения в выражении членов, содержащих некоторую функцию в команде **has( )** следует задавать лишь ИМЯ этой функции (без аргументов).

В функции **has( )** можно задать несколько подвыражений в виде списка. Ее результатом будет ИСТИНА тогда и только тогда, когда найдено хотя бы одно из подвыражений в списке.

Еще одна булева функция **hastype(выражение, тип)** определяет, содержит ли выражение подвыражения заданного типа.
Если из выражения необходимо выделить не операнды, содержащие подвыражения заданного типа, а сами подвыражения, то следует использовать функцию **indets(выражение, тип)**. Эта функция возвращает в виде множества все подвыражения указанного типа.

```
> ex1:=(x+3)*exp(x*cos(x+2))
      +sin(x);
```

$$ex1 := (x+3)\,e^{(x\cos(x+2))} + \sin(x)$$

```
> hastype(ex1,`*`);
```

true

```
> select(hastype,ex1,`*`);
```

$$(x+3)\,e^{(x\cos(x+2))}$$

```
> indets(ex1,`*`);
```

$$\{(x+3)\,e^{(x\cos(x+2))},\ x\cos(x+2)\}$$

2.3 Подстановка и преобразование типов.

При выполнении математических преобразований часто необходимо произвести замену переменных в выражении, функции, уравнении и т.д., то есть вместо какой-то переменной подставить ее представление через некоторые другие переменные. А иногда необходимо выполнить преобразование выражения одного типа в другой. (Такое преобразование типов может потребоваться для выполнения некоторых команд, не работающих с исходным тиом выражения). Для этих целей в Maple существуету несколько команд:

subs(подстановка, ВЫРАЖЕНИЕ)	Синтаксическая подстановка одного выражения вместо другого в ВЫРАЖЕНИЕ
algsubs(подстановка, ВЫРАЖЕНИЕ)	Алгебраическая подстановка одного выражения вместо другого в ВЫРАЖЕНИЕ
subsop(N=новое значение, ВЫРАЖЕНИЕ)	Подстановка нового значения вместо N-го операнда ВЫРАЖЕНИЯ
convert(ВЫРАЖЕНИЕ, тип)	Преобразует ВЫРАЖЕНИЕ в новый тип данных
whattype(ВЫРАЖЕНИЕ)	Определяет тип выражения.

Для подстановки вместо некоторой переменной (выражения) другого выражения служит команда **subs( )**, синтаксис которой имеет следующий вид:

subs(старое выражение=новое выражение, ВЫРАЖЕНИЕ)
subs(s1, s2, .. sn, ВЫРАЖЕНИЕ)
subs([s1, s2, .. sn], ВЫРАЖЕНИЕ)
где каждое из **s1,..sn** является уравнением, определяющим подстановку.

Первая форма команды анализирует **ВЫРАЖЕНИЕ** , определяет в нем все вхождения **старое выражение** и подставляет вместо них **новое выражение**.

Вторая форма команды позволяет выполнить серию подстановок в **ВЫРАЖЕНИЕ**, причем подстановки выполняются последовательно, начиная с **s1**. Это означает, что после выполнения первой подстановки, определенной **s1**, Maple отыскивает вхождения левой части уравнения **s2** во вновь полученном выражении и заменяет каждое такое вхождение на выражение, заданное в правой части уравнения **s2**.

```
> subs(x=y,y=x,[x,y]);
                    [x, x]
> subs({x=y,y=x},[x,y]);
                    [y, x]
```

Третья форма команды содержит подстановки в виде множества или списка. Эти подстановки выполняются ОДНОВРЕМЕННО, а не последовательно, как в предыдущем варианте команды.

То есть вхождения выражений, заданных в левых частях уравнений **s1, s2**, определяются в исходном параметре **ВЫРАЖЕНИЕ**. (см. примеры)

Команда **subs( )** осуществляет так называемую "синтаксическую подстановку", т.е. замена в **ВЫРАЖЕНИИ** происходит только тогда, когда левая часть уравнения подстановки совпадает с одним из операндов в структурном представлении **ВЫРАЖЕНИЯ**. В примере слева команда **subs( )** не подставила в выражение s^3 вместо s^2выражение
$1-c^2$.
Для осуществления подстановки можно воспользоваться несколькими способами:

```
> ex2:=s^3;
                    ex2 := s^3
> subs(s^2=1-c^2,ex2);
                      s^3
> subs(s=sqrt(1-c^2),ex2);
                      (3/2)
                  (1 - c^2)
> simplify(ex2,{s^2=1-c^2});
                    s - s c^2
> algsubs(s^2=1-c^2,ex2);
                    s(1 - c^2)
```

- Явно выразить старую переменную через новые и воспользоваться функцией **subs( )**.
- Воспользоваться командой **simplify( )**, указав в качестве параметра требуемую замену (см. след раздел).
- Воспользоваться командой **algsubs( )**, которая осуществляет алгебраическую подстановку.

Отметим, что полное исключение "старой" переменной произведено только при использовании первого из указанных способов. В остальных случаях "старая" переменная все-таки остается в преобразованном выражении.

2.4 Элементарные математические функции.

Maple V имеет полный набор элементарных математических функций.

Наиболее распространенные целочисленные функции:	
factoial	Функция вычисления факториала (альтернатива-оператор !);
iquo(a,b)	частное для a/b;
irem(a,b)	остаток для a/b;
igcd(a,b)	наибольший общий делитель;
lcm(a,b)	наименьшее общее кратное.

Тригонометрические функции:			
sin	Синус;	**csc**	косеканс;
sec	секанс;	**tan**	тангенс;
cos	косинус	**cot**	котангенс.

Обратные тригонометрические функции:			
arcsin	Арксинус;	**arccsc**	арккосеканс;
arcsec	арксеканс;	**arctan**	арктангенс;
arccos	арккосинус	**arccot**	арккотангенс.

Алгебраические функции:	
exp	Экспоненциальная функция;
ilog10	целочисленный логарифм по основанию 10;
ilog	целочисленный логарифм (библиотечная функция);
ln	натуральный логарифм;
log	логарифм по заданному основанию (библиотечная функция);
log10	логарифм по основанию 10;
sqrt	квадратный корень.

Также имеются гиперболические, обратные гиперболические, функции с элементами сравнения, функции комплексного аргумента и специальные математические функции.

Задание №3:

1. Записать комментарий «Функции математического анализа».
2. Вычислить сумму ряда $\sum_{n=1}^{N} a_n$ и записать в файл при:

   a) $a_n = \frac{36}{n^2 - 5n + 4}$, $N=30$;

   b) $a_n = \frac{54}{n^2 + 5n + 4}$, $N=50$.
3. Вычислить произведение $\prod_{n=1}^{N} b_n$ и записать в файл при:

   a) $b_n = e^{1/n^2} - 1$, $N=5$;

   b) $b_n = \cos\frac{1}{\sqrt{n}} - 1$, $N=4$.
4. Вычислить 1-ую и 2-ую производныефункции $f(x)=x^2+1+2\ln(x+1)$ в точке $x_0=3.027$, результаты занести в файл в виде списка $[f'(x_0), f''(x_0)]$.
5. Вычислить частные производные 2-ую по x и 1-ую по y функции $g(x,y)=xy^2 e^{x^4+2y}$ в точке $(x_0,y_0)=(7.22, 0.67)$, результаты занести в файл в виде списка $[g''_{xx}(x_0,y_0), g'_y(x_0,y_0)]$.
6. Вычислить интегралы и результаты записать в файл:

   a) $\int \frac{dx}{1+x^2}$, при $x = \frac{\pi}{3}$;

   b) $\int_b^a \frac{dx}{x(c^2+x^2)}$, при $a=1$, $b=3$, $c=5$;

   c) $\int_0^1 \int_0^{(1-\sqrt{x})^2} xy\,dx\,dy$

Лекция 3.
Пакеты Maple V.

3.1 Обзор пакетов Maple V.

Встроенные в Maple пакеты позволяют выполнятьматематические построения и преобразования, начиная от элементарной математикии заканчивая общей теорией относительности.

Для того, чтобы использовать команды какого-нибудб пакета, необходимо подключить его или целиком, или только требуеме команды этого пакета (см. Лекцию 1).

В таблице, приведенной ниже, содержится список всех пакетов Maple с их кратким описанием:

Пакет	Содержит
algcurves	Средства для изучения однормерных алгебраических кривых, определяемых полиномами нескольких переменных.
codegen	Средства для создания, обработки и перевода процедур Maple в код языков программирования C и Fortran.
combinat	Комбинаторные функции, включая вычисление перестановок и сочетаний. В настоящее время считается устаревшим. Ждя этих же целей рекомендуется использовать пакет **combstruct.**
combstruct	Команды для работы с комбинаторными структурами.
context	Средства для построения и изменения контекстных меню в графическом интерфейсе пользователя.
DEtools	Средства для выполнения преобразований обыкновенных дифференциальных уравнений, их решения и графического отображения решений с возможностью построения фазовых портретов и полей направлений систем дифференциальных уравнений.
diffalg	Команды для работы с системами полиномиальных дифференциальных уравнений, как обыкновенных, так и в частных производных.
difforms	Команды для работы с дифференциальными формами при решении задач дифференциальной геометрии.
Domians	Команды для создания областей вычисления. Поддерживают работу с полиномами, матрицами и рядами над числовыми кольцами, конечными полями, кольцами полиномов и матриц.
finance	Команды для выполнения финансовых вычислений (финансовая математика).
GaussInt	Команды для работы с гауссовыми целыми числами - комплексными числами вида $a + b\,I$, где a и b целые.
genfanc	Команды для работы с рациональными производящими функциями.
geom3d	Команды для выполнения построений и вычислений в трехмерном евклидовом пространстве. Позволяют строить и работать в трехмерном пространстве с точками, линиями, плоскостями, треугольниками, сферами и т.д.
geometry	Команды для выполнения построений и вычислений нам евклидовой плоскости. Позволяют строить и работать с точками, линиями, плоскостями, треугольниками, окружностями и т.д.
GF	Команды для работы с полями Галуа.
Groebner	Команды для организации вычислений в базисе Гренбера.
group	Команды для работы с группами перестановок и конечными группами.

inttrans	Команды для работы с интегральными преобразованиями и их обратными преобразованиями.
linesumm	Команды определения симметричности ссистем дифференциальных уравнений в частных производных.
linalg	Команды для работы с символьными матрицами и векторами: сложени, умножение матриц, собственные числа и векторы в символьном виде и др.
LinearAlgebra	Усовершенствованные команды линейной алгебры для работы со специальным видом числовых матриц Matrix.
LREtools	Команды для преобразования, графического отображения и решения рекуррентных уравнений.
Matlab	Команды для подключения и использования некоторых матричных функций системы численных вычислений Matlab. Работают при установленном пакете Matlab.
networks	Команды для создания работы с различными типами графов.
numapprox	Команды для построения полиномиальной аппроксимации функций на заданном интервале.
numtheory	Команды для вычислений в области классической теории чисел.
Ore_algebra	Команды для основных вычислений в алгебрах линейных операторов.
orthopoly	Команды построения различных типов ортогональных полиномов.
padic	Команды *p*-адического приближения вещественных чисел.
PDEtools	Средства для выполнения преобразований дифференциальных уравнений в частных производных, их решения и графического отображений решений.
plots	Команды построения специальных видов графиков функций, включая построение линий уровня, отображение неявно заданных функций, включение текстовых надписей в график, построение графиков в различных системах координат.
plottools	Команды для создания и работы с графическими объектами.
polytools	Команды для работы с полиномами.
powseries	Команды построения и работы с формальными степенными рядами.
process	Команды, позволяющие писать многопроцессорные Maple-программы в системе UNIX.
simplex	Команды решения задач линейной оптимизации на основе симплекс-метода.
Slode	Команды построения формального решения линейных обыкновенных уравнений в виде степенных рядов.
Spread	Команды, позволяющие программировать электронные таблици Maple.
stats	Команды статистической обработки данных
student	Команды, наиболее часто используемые студентами (?)

sumtools	Команды вычисления конечных и бесконечных сумм.
tensor	Команды работы с тензорами и их пременение в общей теории относительности.

В предлагаемом курсе рассматриваются несколько пакетов, наиболее полезных, с нашей точки зрения, для выполнения студентами курсовых и дипломных работ. В этой лекии рассматриваются пакеты **linalg**, **LinearAlgabra** и **student**. Пакеты **DEtools** и **PDEtools** рассмотрены в Лекции 6 , графическим пакетам **plots** и **plottools** - посвящена часть Лекции 7 .

В Maple 6 выполнение преобразований линейной алгебры можно осуществлять с помощью команд двух пакетов: **linalg** и **LinearAlgabra**, функциональность которых практически одинакова. Первый пакет входил в состав всех предыдущих версий Maple, тогда как второй пакет - это новое средство, позволяющее работать с числовыми матрицами, в том числе и с матрицами больших размеров, используя всю мощь известного пакета численных расчетов **NAG** (Numerical Algorithms Group).

Основными объектами, с которыми работают команды этих пакетов, являются матрицы, однако матрицы одного пакета не эквивалентны матрицам другого. В пакете **linalg** используются матрицы, построенные на основе массива, создаваемого командой **array( )**, тогда как в пакете **LinearAlgabra** применяются векторы и матрицы, построенные на основе новой структуры r-таблицы (r-table) и создаваемые специальными конструкторами **Vector( )** и **Matrix( )** или с использованием краткой нотации **< a, b, c >**. Матрицы в пакете **linalg** вычисляются только до уровня своих имен, поэтому в нем невозможно вычислить операции поэлементного суммирования или вычитания, используя постые операции над идентификаторами матриц, и приходится пользоваться специальной командой **evalm( )** для вывода результирующих матриц. В пакете **LinearAlgebra** матрицы вычисяются до уровня своих элементов, поэтому простое задание имени матрицы в области ввода рабочего листа приводит к отображению ее элементов, а не имени матрицы, как в случае с пакетом **linalg**. Кроме этого, в пакете **LinearAlgebra** матрицы могут задаваться в качестве операндов сложения и вычитания, что приводит к поэлементному выполнению указанных операций без использования дополнительных команд.

При выборе пакета линейной алгебры для работы рекомендуется принять во внимание следующее:

- Пакет **linalg** полезен при выполнении абстрактных вычислений над матрицами и векторами.
- Пакет **LinearAlgebra** обладает более дружественным интерфейсом и особенно эффективен при работе с ЧИСЛОВЫМИ матицами больших размеров из-за возможности обращения к откомпилированным программам пакета численных расчетов **NAG**.

3.2 Пакет linalg.

Пакет линейной алгебры **linalg** содержит команды создания матриц и векторов, предлагает большой набор функций для работы со структурой этих объектов, для выполнения основных матричных и векторных операций и для решения основных задач линей ной алгебры: решение систем линейеых уравнений, нахождение собственных значений и собственных векторов матрицы, приведение матриц к спкциальным формам и т.д. И все эти действия можно выполнять с матрицами и векторами, элементами которых являются общие алгебраические выражения, получая результаты такде в виде алгебраических выражений.

Определить матрицу или вектор в Maple можно двумя способами: либо с помощью команды **array ()** стандартной библиотеки, либо командами **matrix()** и **vector()** (см Лекцию 2).

Команды для работы со структурой векторов и матриц:

rowdim(M)	Количество строк матрицы M
coldim(M)	Количество столбцов матрицы M
vectdim(V)	Количество элементов вектора V
delrows(M, i..j)	Удаление из матрицы M строк с номерами от **i** до **j**
delcols(M, i..j)	Удаление из матрицы M столбцов с номерами от **i** до **j**
extend (M, Nr, Nc, expr)	Добавление строк и столбцов в матрицу M. Здесь **Nr** и **Nc** -целые числа (включая 0),представляют количество добавляемых строк и столбцов; **expr** - выражение, которое используется в качестве значений добавляемых элементов строк и столбцов
row(M,i)	Выделение строки с номером **i** из матрицы M.
col(M,j)	Выделение столбца с номером **j** из матрицы M.
submatrix (M, i1..i2, j1..j2)	Выделение подматрицы, состоящей из элементов столбцов с номерами от **i1** до **i2** и строк с номерами от **j1** до **j2**
subvector(V, i1..i2)	Выделение вектора, состоящего из из элементов с номерами от **i1** до **i2**
minor(M, i, j)	Матрица минора элемента с индексами **(i, j)**

Команды для выполнения линейных преобразований над строками и столбцами исходной матрицы.

addcol(M, j1, j2, expr)	Создание новой матрицы из матрицы M путем прибавления к столбцу номер **j1** столбца с номером **j2**, умноженного на значение параметра **expr**
addrow(M, i1, i2, expr)	Создание новой матрицы из матрицы M путем прибавления к строке номер **i1** строки с номером **i2**, умноженной на значение параметра **expr**
mulcol(M, j, expr)	Умножение столбца с номером **j** на **expr**
mulrow(M, i, expr)	Умножение строки с номером **i** на **expr**
swaprow(M, i1,i2)	Меняет местами две строки матрицы M
swapcol(M, j1, j2)	Меняет местами два столбца матрицы M

Основные команды для выполнения векторных и матричных операций:

add(A, B) **add(A, B, s1, s2)**	Сумма матриц A и B Линейная комбинация двух матриц: **s1*A+s2*B**, где **s1** и **s2** скаляры.

evalm(A оператор B)	Команда для вычисления матрицы или вектора на уровне их элементов, используется для вычисления любых возможных действий, заданных операторами сложения **(+)**, вычитания **(-)**, умножения **(&*)**, деления **(/)** и возведения в степень**(^)**.
multiply(A,B)	Умножение матрицы на матрицу или вектор
angle(U,V)	Вычисление угла между векторами **U** и **V**.
grad(expr, V)	Вычисление градиента скалярного выражения над векторным полем V (задается как vector() или в виде списка координат)
diverge(F,V)	Вычисление дивергенции векторной функции **F**(в виде вектора или списка выражений) над полем вектора **V** (вектор или список)
curl(F, V)	Вычисление ротора **F** над полем вектора **V** (**F** и **V** векторы или списки из 3-х элементов)
interprod(U,M1,M2,..,Mn, V)	Вычисление скалярного произведения (**U, V** - векторы, **M1,M2,..,Mn** -матрицы)
potential(F,var,'V')	Вычисление потенциала векторного поля **F**. **var** -список переменных, **' V'** имя функции, которой присваивается вычисленный потенциал.
augment(A,B,...)	Объединение двух и более матриц горизонтально ("бок о бок")
stack(A,B,...)	Объединение двух и более матриц вертикально ("одна под другой")
copyinto(A,B,m,n)	Копирование элементов матрицы **A** в матрицу **B** начиная с позиции **[m,n]** (B[m,n]=A[1,1])
transpose(M)	Вычисление транспонированной матрицы или вектора.
inverse(M)	Вычисление обратной матрицы
jacobian(F,V)	Якобиан-матрица векторной функции. (**F** - вектор или список выражений, **V** - вектор или список переменных). (i,j)-й элемент матрицы представляет diff(F[i], V[j]).
charmat(M,lambda)	Создание характеристической матрицы для матрицы **M** (**lambda** - переменная, используемая для обозначения характеристических чисел)
charpoly(M,lambda)	Выдает характеристический полином матрицы **M**

Команды, позволяющие получить те или иные характеритики матриц:

det(A)	Определитель матрицы A

rank(A)	Ранг матрицы A
definite(A, kind)	Тест на положительно- (отрицательно) определенные матрицы
cond(A)	Вычисление условных чисел матрицы
orthog(M)	Тест на ортогональность матрицы **M**
singularvals(M)	Вычисляет сингулярные значения матрицы.
Eigenvals(M,V)	Вычисляет собственные значения матрицы M и соответствующие собственные векторы, которые помещает в вектор V.
eigenvals(M)	Вычисляет собственные числа матрицы M
eigenvects(M,V)	Вычисляет собственные числа, их кратность и соответствующие каждому собственному числу собственные векторы.

Команды для решения линейных уравнений и систем:

linsolve(M,B)	Решение системы линейных уравнений, представленных в матричной форме: M*x=B
leastsqrd(M,B)	Решение уравнений по методу наименьших квадратов.

В пакете **linalg** имеется также достаточное количество команд для всевозможных разложений матриц, представления их в той или иной форме (Эрмитова, Жордана, Гильберта и др.)

3.3 Пакет LinearAlgebra.

Все команды пакета **LinearAlgebra** можно вызвать непосредственно по имени, предварительно подключив весь пакет стандартным способом, или можно подключить отдельную команду с использованием синтаксиса

with(LinearAlgebra, имя команды);

Можно вызвать команду, предварительно не подключая ее, а используя длинное имя

LinearAlgebra[имя команды] (параметры);
LinearAlgebra['имя команды'] (параметры);

Последняя форма (имя команды, заключенное в кавычки), вызывает соответствующую команду пакета, даже если в текущем сеансе используется какой-либо объект с таким же именем.

Пакет **LinearAlgebra** реализован в виде модуля, новой языковой конструкции Maple, использующей элементы объектно-ориентированного программирования. Каждая команда является методом объекта LinearAlgebra, и поэтому ее можно вызвать, используя специальную операцию **:-** обращения к методу объекта.

LinearAlgebra :- имя команды (параметры);

В этом случае вызываемая команда также будет загружена, не конфликтуя с объектом другого типа, созданным в текущем сеансе.

Для получения более полной информации по пакету **LinearAlgebra** можно загрузить справку командой **?LAOverview** . На этой странице справки расположены ссылки на другие страницы, подробно описывающие работу и программирование пакета **LinearAlgebra**, включая рабочие листы с примерами использования подпрограмм пакета **NAG**.

3.3.1. Основные типы данных. Создание векторов и матриц.

Основные типы данных, с которыми работают команды пакета **LinearAlgebra**, являются скаляры, представляющие как числа, так и алгебраические выражения, а также матрицы и векторы, определяемые на базе нового типа данных r- таблицы. (В данном курсе не рассматривается этот тип данных).

Конструктором матриц является команда **Matrix( )** (обязательно с заглавной буквы), все параметры которой являются необязательными. Её синтаксис :

Matrix(r, c, init, ro, scan, shape, storage, order, dat, fill, attr);

Значение параметров и их допустимые значения приводятся в таблице ниже

Параметры конструктора матриц

Параметр	Описание
r	Неотрицательное целое число или диапазон целых чисел, начинающийся с 1. Представляет количество строк в матрице.
c	Неотрицательное целое число или диапазон целых чисел, начинающийся с 1. Представляет количество столбцов в матрице.
init	Задает значения элементов матрицы. Может быть одним из следующих объектов Maple: • ***процедурой***, входными параметрами которой является пара целых положительных чисел, определяющих индексы элемента, а возвращаемым значением - величина этого элемента, например,**(i,j)→i*j**; • ***алгебраическим выражением***, которое вычисляется как процедура с двумя параметрами, возвращающая значение элемента; • ***таблицей***, элементы которой с неорицательными индексаим представляют значения соответствующих элементов матрицы; • ***множемтвам уравнений*** вида **(i,j)= значение**, в которых неотрицательные индесы представляют индексы соответствующего элемента матрицы; • ***массивом*** на основе таблицы или r-таблицы, созданным, соответственно, либо командой **array( )** , либо командой **Array( )**, у которого индекы начинаются с 1;

	• ***матрицей*** на основе r-таблицы, т.е. матрицей, созданной конструктором **Matrix( )**; • ***списком***, элементы которого интерпретируются как значения первой строки матрицы, или списком, элементами котрого являются списки, интерпретируемые как последовательные строки матрицы
ro	Задается в виде **readonly** или **readonly=true** и определяет, что значения элементов матрицы, определенные при ее создании, не могут быть изменены в дальнейшем;
scan	Уравнение вида **scan=имя** или **scan=список**, определяющее структуру и/или порядок данных при интерпретации начальных значений, заданных параметром **init**;
shape	Уравнение вида **shape=имя** или **shape=список**, определяющее одну или более встроенных или пользовательских индексных функций, задающих расположение в памяти элементов матрицы;
storage	Уравнение вида **storage=имя**, где **имя** является одним из допустимых режимов памяти, определяя тем самым требования памяти для размещения элементов матрицы;
order	Уравнение вида **order=имя**, где **имя** может быть либо , либо , задавая хранение матрицы в памяти, соответственно, по строкам или столбцам
dat	Уравнение вида **datatype=имя**, где **имя** может быть любым типом Maple, определяющим тип данных, хранимых в матрице;
fill	Уравнение вида **shape=значение**, определяющее значение, присваемое неопределенным элементам матрицы. По умолчанию оно равно**0**(нуль);
attr	Уравнение вида **attributes=список**, определяющее атрибуты (положительно-определенная, эрмитова и т.д.), с которыми матрица была создана.

```
> Matrix(2);
```

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

```
> Matrix(2,3);
```

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

```
> Matrix(1..1,1..4,6);
```

$$[6 \quad 6 \quad 6 \quad 6]$$

```
> Matrix([[1,2,3],[4,5,6]]);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
> Matrix(2,(i,j)->x^(i+j));
```

$$\begin{bmatrix} x^2 & x^3 \\ x^3 & x^4 \end{bmatrix}$$

Как уже указывалось, все параметры являются необязательными, и в случае их отсутствия создается матрица размерности **0 x 0**. Вообще, для создания матрицы важны первые три параметра. Остальные используются различными командами для ускорения ее обработки.

На рисунке слева приводятся примеры создания матриц.

Создать вектор можно конструктором **Vector( )** со следующими синтаксисом:

Vector (d, init, ro, shape, storage, dat, fill, attr, orient);
Vector[column] (d, init, ro, shape, storage, dat, fill, attr);
Vector[row] (d, init, ro, shape, storage, dat, fill, attr);

```
> Vector(2);
```

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

```
> Vector(1..3,5, orientation=row);
```

$$[5, 5, 5]$$

```
> Vector[row]([1,2,3]);
```

$$[1, 2, 3]$$

```
> Vector(2,(i)->x^i);
```

$$\begin{bmatrix} x \\ x^2 \end{bmatrix}$$

При интерактивной работе в Maple иногда не совсем удобно создавать матрицы или векторы с помощью конструкторов. Разработчики пакта **LinearAlgebra** предоставили пользователю возможность создавать вектора и матрицы, используя краткую форму их определения:

< a, b, c > создает матрицу или вектор по строкам;
< a | b | c > создает матрицу или вектор по столбцам.

Если величины, задаваемые в угловых скобках, не являются скалярами, то создается матрица, в противном случае - вектор.

В пакете **LinearAlgebra** различаются векторы-столбцы и веторы-строки. Векторы-столбцы определяются с помощью первых двух форм конструктора, причем в 1-ой форме необходимо задать последний параметр **orientation = column**. Для создания вектора-строки используется третья форма команды и первая с последним параметром**orientation = row**. Первый из параметров **d** задает размерность вектора и может принимать только целые положительные значения, большие или равные 1. Остальные параметры соответствуют аналогичным в конструкторе матриц.

```
> V1:=<1,2,3>;
```
Создание вектора-столбца

$$V1 := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
> V2:=<1|2|3>;
```
Создание вектора-строки

$$V2 := [1, 2, 3]$$

```
> M1:=<<1,2>|<3,4>>;
```
Создание матрицы по столбцам

$$M1 := \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

```
> M2:=<<1|a>,<2|b>>;
```
Создание матрицы по строкам

$$M2 := \begin{bmatrix} 1 & a \\ 2 & b \end{bmatrix}$$

```
> MM:=<M1,M2>;
```
Создание матрицы из двух других

$$MM := \begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 1 & a \\ 2 & b \end{bmatrix}$$

```
> IdentityMatrix(2,2): Единичная матрица
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
> ZeroMatrix(2,3): Нулевая матрица
```

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

```
> ConstantMatrix(6,2): Матрица-константа
```

$$\begin{bmatrix} 6 & 6 \\ 6 & 6 \end{bmatrix}$$

```
> ScalarMatrix(a^2,2): Скалярная матрица
```

$$\begin{bmatrix} a^2 & 0 \\ 0 & a^2 \end{bmatrix}$$

```
> UnitVector[row](2,3): Единичный вектор
```

$$[0, 1, 0]$$

```
> ZeroVector[row](3): Нулевой вектор
```

$$[0, 0, 0]$$

```
> ConstantVector[row](5,3): Вектор-константа
```

$$[5, 5, 5]$$

```
> ScalarVector[row](x^2+y^2,3,4): Скалярный вектор
```

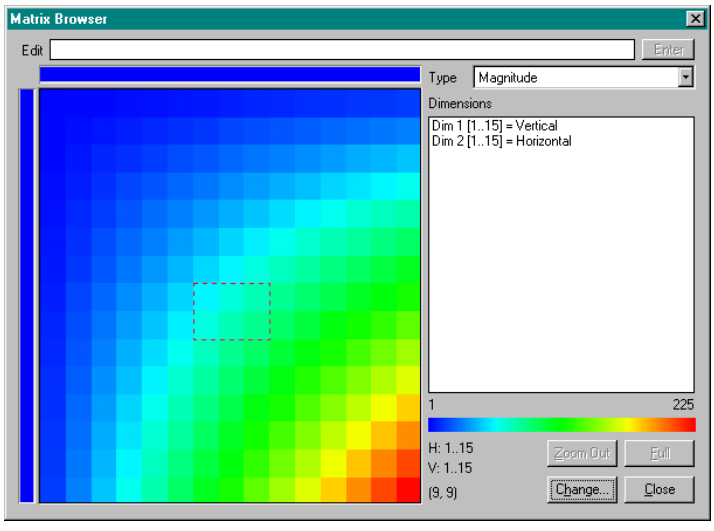
$$[0, 0, x^2+y^2, 0]$$

Для создания специальных типов матриц и векторов - единичных, нулевых, матриц и векторов констант и скалярных - можно использовать специальные конструкторы, хотя объекты подобного типа можно создать и при помощи общих конструкторов. На рисунке слева демонстрируется работа специальных типов конструкторов

При задании матриц и векторов больших размеров они не отображаются на рабочем листе, Вместо их содержимого отображается подсказка, что здесь расположен соответствующий объект и указывается его структура и размерность.

```
Matrix(15,15,(i,j)->i*j);
```

```
15 x 15 Matrix
Data Type: anything
Storage: rectangular
Order: Fortran_order
```



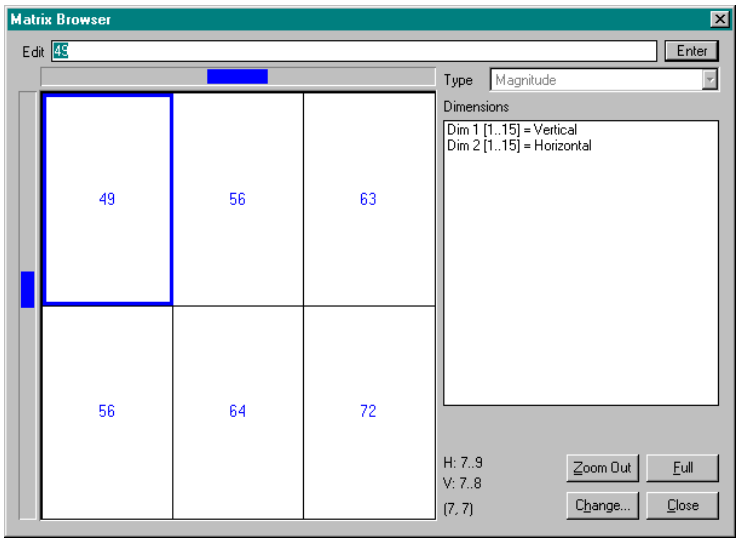
Для просмотра подобных векторов и матриц в Maple включена специальная программа просмотра структурированных данных (Structured Data Browser), которую можно вызвать из контекстного меню командой **Browser**. Слева на рисунке представлено окно этой программы

В этом окне отображается вся матрица, элементы которой представлены квадратиками разных цветов, в зависимости от величины значения.

В поле со списком **Tupe** могут быть установлены следующие режимы цветового отображения содержимого ячеек матрицы:

- **Structure** - отображение нулевых элементов белым цветом, а ненулевых - черным;
- **Magnitude** - цветная легенды (по умолчанию);

- **Density** - градации серого цвета;



При выделении мышью необходимых элементов матрицы (см. рисунок выше) в том же окне отображаются их значения, которые можно корректировать в поле **Edit** (рисунок слева)

3.3.2. Элементарные операции с матрицами и векторами.

Как уже отмечалось ранее, основные операции с матрицами в пакете **LinearAlgebra** выполняются проще, чем такие же в пакете **linealg**.

```
> M1:=<<1|2>,<3|4>>;
```

$$M1 := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> M2:=<<10|7>,<8|15>>;
```

$$M2 := \begin{bmatrix} 10 & 7 \\ 8 & 15 \end{bmatrix}$$

```
> M1+M2;
```

$$\begin{bmatrix} 11 & 9 \\ 11 & 19 \end{bmatrix}$$

```
> M1-M2;
```

$$\begin{bmatrix} -9 & -5 \\ -5 & -11 \end{bmatrix}$$

```
> 3.1*M1+5*M2;
```

$$\begin{bmatrix} 53.1000000000000014 & 41.2000000000000029 \\ 49.2999999999999972 & 87.4000000000000057 \end{bmatrix}$$

Это связано с тем, что идентификаторы векторов и матриц здесь вычисляются не до уровня имени, а непосредственно до уровня вычисления их компонентов. В связи с этим возможно выполнение поэлементного сложения, вычитания и составления линейных комбинаций векторов и матриц одинаковой размерности с использованием обычных арифметических операций.

```
> V1:=<1|4>;
```

$$V1 := [1, 4]$$

```
> V2:=<3|8>;
```

$$V2 := [3, 8]$$

```
> 3*V1-6*V2;
```

$$[-15, -36]$$

```
> V3:=<3,8>;
```

$$V3 := \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

```
> V1+V3;
Error, (in rtable/Sum) invalid arguments
```

```
> 10+<<2|5|11>,<4|6|7>>;
  IdentityMatrix(2,3);
  10*IdentityMatrix(2,3)+<<2|5|11>,<4|6|7>>;
```

$$\begin{bmatrix} 12 & 5 & 11 \\ 4 & 16 & 7 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 12 & 5 & 11 \\ 4 & 16 & 7 \end{bmatrix}$$

```
> 2+<1,2>;
Error, (in rtable/Sum) invalid arguments
```

Если складывается скаляр с матрицей, то это равносильно сложению матрицы с единичной матрицей, элементы которой умножены на заданный скаляр. НО!!! Вектор нельзя складывать со скаляром.

Построение линейной комбинации матриц и векторов можно также выполнить, используя, соответственно, команды **MatrixAdd( )** и **VectorAdd( )**.

```
> -3*<1|2|3>;
                        [-3, -6, -9]
> 4*<<7,8>|<1,6*t>>;
                        [28    4  ]
                        [32   24t ]
```

Так как произведение матриц (имеется ввиду операция скалярного умножения) не является коммутативной, то использование операции коммутативного умножения **(*)** для векторов и матриц приводит к ошибке. (Исключение допускается только для умножения матрицы саму на себя, причем в этом случае выполняется скалярное умножение.) Коммутативное умножение можно использовать для перемножения скаляра и матрицы/вектора. В этом случае все элементы этих объектов умножаются на соответствующий скаляр.

Однако, если скаляр содержит неопределенную переменную, то перемножения не происходим, так как Maple не знает, какой объект в дальнейшем эта переменная может содержать. Для выполнения такого умножения следует использовать команду **simplify** с параметром **symbolic** или опцией **assume=scalar**.

```
> mull:=x^2*<1|2|3>;
                        mull := x^2 [1, 2, 3]
> simplify(mull,symbolic);
                        [x^2, 2 x^2, 3 x^2]
```

```
> m1:=x.y.z;
                        m1 := x . y . z
> m2:=x.z.y;
                        m2 := x . z . y
> is(m1=m2);
                        FAIL
```

Выполнить некоммутативное умножение в Maple (начиная с 6-ой версии программы) можно операцией, символом которой является точка **(.)**. Она никогда не меняет сомножители местами, поэтому произведения **x.y.z** и **x.z.y** не являются тождественными.

Эта же операция, примененная к матрицам и векторам, выполняет их скалярное произведение.

```
> <1,3>.<4|6>;
```
Вектор-столбец умножается на вектор-строку

```
                        [ 4    6]
                        [12   18]
> <4|6>.<1,3>;
```
Вектор-строка умножается на вектор-столбец

```
                          22
> <<3,-1>|<-8,5>|<9,10>>.<<1,x,y>|<4,-7,2>>;
```
Матрица 2 x 3 умножается на матрицу 3 x 2

```
                        [3 - 8 x + 9 y     86]
                        [-1 + 5 x + 10 y   -19]
```

Для получения степени квадратной матрицы можно последовательно применить операцию скалярного умножения необходимое число раз или операцию возведения в степень **(^)**.

Показатель степени может быть и отрицательным числом, что позволяет вычислять обратную матрицу и ее степени.

```
> M:=<<0.1|0.2>,<0.3|0.4>>;
                        M := [.1   .2]
                             [.3   .4]
> M.M.M.M.M;
        [.0106900000000000016   .0155800000000000036]
        [.0233700000000000019   .0340600000000000069]
> M^5;
        [.0106900000000000034   .0155800000000000054]
        [.0233700000000000019   .0340600000000000069]
> M^(-1);
        [-20.000000000000072    10.000000000000036]
        [15.000000000000036     -5.000000000000178]
> %.M;
                        [1.   0.]
                        [0.   1.]
```

```
> F:=Matrix(4,(i,j)->3*i-j);
```

$$F := \begin{bmatrix} 2 & 1 & 0 & -1 \\ 5 & 4 & 3 & 2 \\ 8 & 7 & 6 & 5 \\ 11 & 10 & 9 & 8 \end{bmatrix}$$

```
> F[3,2];
```
Выбор элемента матрицы

$$7$$

```
> F[3,1..-1];
```
Выбор третьей строки

$$[8, 7, 6, 5]$$

```
> F[1..-1,2];
```
Выбор второго столбца

$$\begin{bmatrix} 1 \\ 4 \\ 7 \\ 10 \end{bmatrix}$$

```
> F[2..3,1..3];
```
Выбор блока размерности 2 x 3

$$\begin{bmatrix} 5 & 4 & 3 \\ 8 & 7 & 6 \end{bmatrix}$$

Для выделения элементов матрицы и ее подматриц используется индексная запись, причем в качестве индекс можно указывать диапазон, что позволяет выделять целые блоки исходной матрицы.

Присваивание новых значений элементам матрицы также осуществляется с помощью индексов, причем и здесь можно использовать диапазон. Главное, чтобы размерность матрицы в левой части оператора присваивания соответствовала размерности матрицы в правой части.

Возможности выбора и присваивания значений целым строкам или столбцам используется для осуществления операций со строками или столбцами матрицы. (см. рисунки ниже)

```
> A:=Matrix(3,(i,j)->i*5+j-5);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 6 & 7 & 8 \\ 11 & 12 & 13 \end{bmatrix}$$

```
> A[[1,2],1..-1]:=A[[2,1],1..-1];
```
Перестановка местами строк 1 и 2

$$A_{[1,\,2],\,1\,..\,-1} := \begin{bmatrix} 6 & 7 & 8 \\ 1 & 2 & 3 \end{bmatrix}$$

```
> A;
```

$$\begin{bmatrix} 6 & 7 & 8 \\ 1 & 2 & 3 \\ 11 & 12 & 13 \end{bmatrix}$$

```
> A[3,1..-1]:=5*A[3,1..-1];
```
Умножение строки 3 на 5

$$A_{3,\,1\,..\,-1} := [55, 60, 65]$$

```
> A;
```

$$\begin{bmatrix} 6 & 7 & 8 \\ 1 & 2 & 3 \\ 45 & 40 & 35 \end{bmatrix}$$

```
> A[3,1..-1]:=A[3,1..-1]-10*A[2,1..-1];
```
Вычитание из строки 3 строки 2, умноженной на 10

$$A_{3,\,1\,..\,-1} := [35, 20, 5]$$

```
> A;
```

$$\begin{bmatrix} 6 & 7 & 8 \\ 1 & 2 & 3 \\ 35 & 20 & 5 \end{bmatrix}$$

Все перечисленные выше операции можно выполнять с помощью команд пакета **LinearAlgebra**, которые рекомендуется использовать при программировании в Maple, хотя и допустимо их использование при интерактивной работе. Список команд допустимых операций над матрицами и векторами приведен в таблице.

Команды для работы с матрицами и векторами.

Команда	Описание
DeleteRow	Удаление строки матрицы

DeleteColumn	Удаление столбца матрицы
Row	Выделение строки матрицы
Column	Выделение столбца матрицы
SubMatrix	Выделение подматрицы из заданной матрицы
SubVector	Выделение подвектора из заданного вектора
ScalarMultiply	Умножение матрицы/вектора на скаляр
MatrixVectorMultiply	Скалярное произведение матрицы на вектор-столбец
VectorMatrixMultiply	Скалярное произведение на вектора-строки на матрицу
MatrixMatrixMultiply	Скалярное произведение матрицы на матрицу
MatrixInverse	Вычисление обратной матрицы
Determinant	Вычисление определителя матрицы
Minor	Вычисление миноров матрицы
ConditionNumber	Вычисление числа обусловленности матрицы
Eigenvalues	Вычисление собственных значений матрицы
Eigenvectors	Вычисление собственных векторов матрицы

3.3.3 Решение систем линейных уравнений.

В пакет **LinearAlgebra**, как и в пакет **linalg** входит специальная команда **LinearSolve( )** решения систем линейных алгебраических уравнений. В отличие от своего двойника **linsolve( )** из пакета **linalg** в этой команде можно указать способ, которым следует решать систему, при этом сама система уравнений задается так же, как и для функции **linsolve( )** в матричной форме (A.x=B), т.е. в качестве параметров ей передается матрица **A** системы и вектор правых частей **B**. Общий синтаксис команды:

> LinearSolve(A, B, m, free, c, inp, outops);

Параметры команды **LinearSolve( );**

Параметр	Описание
A	Матрица системы
B	Правая часть системы (вектор-столбец или матрица)
m	Задается в форме **method = имя**, где **имя** может принимать следующие значения: **'none', 'subs', 'solve' ,'Cholesky', 'LU', 'QR', 'SparseLU'** и определят метод решения системы уравнений (необязательный параметр).
free	Определяет базовое имя переменной в форме **free = имя**, которое используется для конструирования имен параметров в случае, если исходная система уравнений имеет множество решений (необязательный параметр).
c	Задается в форме уравнения **conjugate = true/ false** и определяет, следует ли строить эрмитову сопряженную матрицу при использовании метода Холецкого иои QR-декомпозиции (необязательный параметр).
inp	Задается в виде **inplace = true/ false** и определяет, поме-

	щать ли решение в вектор или матрицу **B**, или формировать новый объект для решения. Значение по умолчанию **false**.
outops	Определяет опции **outputoptions**, представляющие дополнительную информацию конструктору решения (неизменяемая матрица, тип, математические атрибуты и т.д.)

Несколько замечаний по поводу обязательных параметров. Параметр **B**, представляющий правую часть системы, может задаваться как в виде вектора, так и в виде матрицы. В последнем случае за одно обращение к команде решения системы линейных уравнений будет решаться множество систем с правыми частями, представленными векторами-столбцами матрицы **B**.

Параметр **B** можно и не задавать, передавая в качестве первого параметра расширенную матрицу системы **< A | B >**. Размерности матрицы решения согласовываются с размерностями матриц правой и левой частей уравнения. Если матрица системы **A** имеет размерность **m x n**, и правая часть представлена матрицей **m x p**. то результатом будет матрица размерности **n x p**, столбцы которой будут являться решениями соответствующих систем.

```
> M;
```

$$\begin{bmatrix} .1 & .2 \\ .3 & .4 \end{bmatrix}$$

```
> B:=<3.,-4.5>;
```

$$B := \begin{bmatrix} 3. \\ -4.5 \end{bmatrix}$$

```
> y:=LinearSolve(M,B,method='LU');
```

$$y := \begin{bmatrix} -105.000000000000028 \\ 67.5000000000000142 \end{bmatrix}$$

```
> y:=LinearSolve(<M|B>,method='LU');
```

$$y := \begin{bmatrix} -105.000000000000028 \\ 67.5000000000000142 \end{bmatrix}$$

```
> M;
```

$$\begin{bmatrix} .1 & .2 \\ .3 & .4 \end{bmatrix}$$

```
> R:=<<1.|2.5>,<5.3|6.>>;
```

$$R := \begin{bmatrix} 1. & 2.5 \\ 5.3 & 6. \end{bmatrix}$$

```
> y:=LinearSolve(M,R,method='solve');
```

$$y := \begin{bmatrix} 33. & 10. \\ -11.50000000 & 7.500000000 \end{bmatrix}$$

3.4 Пакет student.

В этом пакете содержится почти полсотни функций, которые наиболее часто используются студентами на практических занятиях и при подготовке курсовых и дипломных проектов.

Команды пакета **student**, относящиеся к разделу математики "Математический анализ":

Команда	Описание.
D	Дифференциальный оператор.
Diff	Инертная форма функции вычисления производной.
Int	Инертная форма функции интегрирования
Doubleint	Инертная форма функции вычисления двойного интеграла.
Tripleint	Инертная форма функции вычисления тройного ин-

	теграла.
Limit	Инертная форма функции вычисления предела
Lineint	Инертная форма функции вычисления линейного интеграла
Product	Инертная форма функции вычисления произведения членов последовательности.
Sum	Инертная форма функции вычисления суммы членов последовательности.
intparts	Интегрирование по частям.
integrand	Вывод подынтегрального выражения из-под знака инертного интеграла.
leftsum	Числовое приближение к интегралу левыми прямоугольниками.
middlesum	Числовое приближение к интегралу центральными прямоугольниками.
rightsum	Числовое приближение к интегралу правыми прямоугольниками.
simpson	Числовое приближение к интегралу по методу Симпсона.
trapezoid	Числовое приближение к интегралу методом трапеции.
extrema	Вычисление экстремума выражения.
maximize	Вычисление максимума функции.
minimize	Вычисление минимума функции.
slope	Вычисление и построение касательной к заданной точке функции.
value	Вычисляет инертные функции.

Примеры команд из раздела "Математический анализ"

```
> f:=x^2*y*z^3;
```

$$f := x^2 y z^3$$

```
> Diff(f,x);
```

$$\frac{\partial}{\partial x} x^2 y z^3$$

```
> Diff(f,x$2)=value(Diff(f,x$2));
```

$$\frac{\partial^2}{\partial x^2} x^2 y z^3 = 2 y z^3$$

```
> gg:=(x,y,z)->x^2*y*z^3;
```

$$gg := (x,y,z) \to x^2 y z^3$$

```
> D[1](gg); # производная по первому
  аргументу функции g (по x)
```

$$(x,y,z) \to 2 x y z^3$$

```
> D[1,3](gg);#смешанная производной
  по 1-му (x) и третьему (z)аргументу
```

$$(x,y,z) \to 6 x y z^2$$

```
> g1:=(x)->sin(x^2);
```

$$g1 := x \to \sin(x^2)$$

```
> D(g1)(x);
  diff(g1(x),x);
```

$$2\cos(x^2)\,x$$

$$2\cos(x^2)\,x$$

```
> D[1,1](g1);
  (D@@2)(g1)(x);
```

$$x \to -4\sin(x^2)\,x^2 + 2\cos(x^2)$$

$$-4\sin(x^2)\,x^2 + 2\cos(x^2)$$

```
> with (student):
  g:=(x,y)->x*exp(y);
```

$$g := (x,y) \to x\,e^y$$

```
> Doubleint(g(x,y),x=1..2,y=-1..1);
```

$$\int_{-1}^{1}\int_{1}^{2} x\,e^y\,dx\,dy$$

```
> evalf(%);
```

$$3.525603581$$

```
> value(%%);
```

$$\frac{3}{2}e - \frac{3}{2}e^{(-1)}$$

```
> D1:=Doubleint(g(x,y),
  x=1..2,y=-1..1):
  D1=value(D1);
```

$$\int_{-1}^{1}\int_{1}^{2} x\,e^y\,dx\,dy = \frac{3}{2}e - \frac{3}{2}e^{(-1)}$$

```
> slope(y = sin(x), y, x);
```

$$\cos(x)$$

```
> leftsum(sin(x)*x+sin(x),
  x=0..2*Pi, 4);
  value(%);
```

$$\frac{1}{2}$$

$$\pi\left(\sum_{i=0}^{3}\left(\frac{1}{2}\sin\left(\frac{1}{2}i\,\pi\right)i\,\pi + \sin\left(\frac{1}{2}i\,\pi\right)\right)\right)$$

$$-\frac{1}{2}\pi^2$$

```
> f:=(x,y,z)->(x^2+y^2+z);
```

$$f := (x,y,z) \to x^2 + y^2 + z$$

```
> T1:=Tripleint(f(x,y,z),x,y,z);
```

$$T1 := \iiint x^2 + y^2 + z\,dx\,dy\,dz$$

```
> T1=value(T1);
```

$$\iiint x^2 + y^2 + z\,dx\,dy\,dz = \frac{1}{3}x^3 y z + \frac{1}{3}y^3 x z + \frac{1}{2}z^2 x y$$

```
> integrand(T1);
```

$$x^2 + y^2 + z$$

```
> intparts(Int(x^k*ln(x), x),
  ln(x));
```

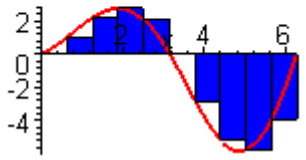
$$\frac{\ln(x)\,x^{(k+1)}}{k+1} - \int \frac{x^{(k+1)}}{x\,(k+1)}\,dx$$

Графические функции пакета **student**:

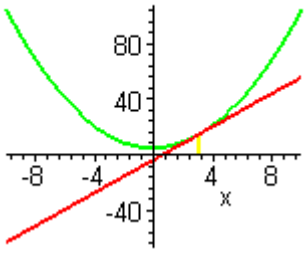
Команда	Описание.
leftbox	Графическая иллюстрация интегрирования методом левых прямоугольников.
middlebox	Графическая иллюстрация интегрирования методом центральных прямоугольников.
rightbox	Графическая иллюстрация интегрирования методом правых прямоугольников.
showtangent	График функции и касательная линия.

Примеры графических команд

```
> leftbox(sin(x)*x+sin(x),
  x=0..2*Pi, 10,
  shading=BLUE);
```




```
> showtangent(x^2+5, x = 3)
```



Геометрические функции пакета **student**:

Команда	Описание.
Point	Тестирование объекта на соответствие типу точки
distance	Вычисляет расстояние между точками.
intercept	Нахождение точки пересечения двух кривых.
midpoint	Вычисляет среднюю точку сегмента линии.

Примеры геометрических расчетов

```
> intercept(y+4 = x+5,
  x = 3*y);
```

$$\{y = \frac{-1}{2}, x = \frac{-3}{2}\}$$

```
> distance([a, b], [c, d]);
```

$$\sqrt{(c-a)^2+(d-b)^2}$$

Функции преобразования выражений и команд

Команда	Описание.
changevar	Замена переменной.
combine	Объединение подобных членов.
completesquare	Вычисление полного квадрата (многочлена).
equate	Создание системы уравнений из списков, таблиц, массивов.
isolate	Выделение подвыражения.
makeproc	Преобразование выражения в процедуру Maple
powsubs	Подстановка для множителей выражения.

Примеры команд преобразрвания выражений.

```
> changevar(cos(x)+1=u,
  Int((cos(x)+1)^3*sin(x),
  x), u);
```

$$\int -u^3\,du$$

```
> value(Int(-u^3,u));
```

$$-\frac{1}{4}u^4$$

```
> p:= x^2 + 2*x + 3;
```

$$p := x^2+2x+3$$

```
> ff := makeproc(p, x);
```

$$ff := x \rightarrow x^2+2x+3$$

```
> ff(5);
```

$$38$$

```
> ff(u);
```

$$u^2+2u+3$$

Задание 3.

1. Создайте последовательность S1, состоящую из трех элементов, каждый из которых определяется как x^i, где *i* меняется от 2 до 3.
2. Создайте список S2, содержащий элементы *x, y, z*

   .

3. Выполните подстановку *x=2y, y=3z, z=4x* в список S2: а) последовательно, в) одновременно.
4. Выполните подстановку $x^3=Cos(y)$ в последовательность S1, Определите новую последовательность, заменив в первом члене полученной последовательности функцию *cos* на *tg(x)*.
5. Создайте вектор V1, элементами которого являются члены списка S2; создайте вектор V2, элементами которого являются элементы списка S2, возведенные в степень *1/3*.
6. Подключите пакет *linalg*. Вычислите скалярное произведение векторов V1 и V2 (*dotprod*). Определите длину вектора V1 (*norm*). Нормируйте вектор V1 (*normalize*). Проверьте, что длина вектора V1 равна единице.
7. Подключите пакет *LinearAlgebra*. Определите матрицу M1 размерности 3 x 3, вводя ее элементы построчно: $1^{ая}$ строка - (2,4,8), $2^{ая}$ строка - (1,11,-5), $3^{ья}$ строка - (-4,5,7). Создайте матрицу M2, присвоив те же значения элементам матрицы, но по столбцам. Вычислите скалярное произведение матриц M1 и M2.
8. а) Вычислите для матрицы M1 определитель, ранг, собственные значения и собственные векторы.
   б) Составьте для матрицы M1 характеристическую матрицу, вычислите ее определитель, в полученном уравнении определите неизвестные и сравните полученный результат с собственными значениями, полученными в п.а).
9. Подключите пакет *student*. Вычислите, используя его функции, площадь фигуры заключенной между кривыми $(x^2 / 3) - 2$ и $ln(x)$ (для $x>0$). (intercept, Doubleint, value). Сравните полученный результат со стандартным повторным интегралом.
10. Определите функцию F(x) , вычисляющую числовое значение неопределенного интеграла от функции sin(x), используя:
    а) функциональный оператор,
    б) функцию unapplay.
    Вычислите значение F в точках x = Pi/2; и x = Pi/4;
11. Создайте функцию *g(x)*, равную x^2, если $x^2>4$, и равную $-2*x^2$, если $x^2<4$, используя функцию piecewise (см. справку Maple).
    Постройте график функции, используя контекстно- зависимое меню.(Быстрое построение графиков)

Лекция 4.
Основы программирования в MAPLE V.

4.1 Задание функций пользователя.
4.2 Условные выражения.
4.3 Операторы цикла.
4.4 Операторы прерывания и обработки ошибок.
4.5 Процедуры.
4.6 Средства отладки процедур, их сохранение и использование (подключение) .

4.7 Задания.

4.1 Задание функций пользователя.

1) Один из наиболее простых способов задания функции пользователя - присвоение введенной функции (в виде выражения) некоторой переменной:
Name:=выражение
Этот прием фактически означает операцию присваивания.

```
> m:=sqrt(x^2+y^2);
                    2    2 1/2
              m := (x  + y )

> x:=3:y:=4:
> m;
                  1/2
              25

> evalf(m);
              5.000000000
```

Заданный таким образом объект все же не является полноценной функцией пользователя. Прежде всего потому, что в нем используются только глобальные переменные (x и y). Их значения приходится определять заведомо, используя операцию присваивания. Подобные "функции" нельзя ввести в библиотеки Maple.

2)Более гибкий способ задания полноценных функций пользователя - использование функционального оператора →.Его синтаксис:

name:=(x,y,...)→Выражение,
где **name** -имя функции (без аргументов),
(x,y,...) - список формальных параметров функции,
Выражение - вид функции (некоторое выражение, зависящее от параметров (x,y,...)).

Вызов функции осуществляется в виде **name (x,y,...)**.Переменные, указанные в списке формальных параметров являются локальными. При подстановке на их место фактических параметров они сохраняют их значения только в теле функции **Выражение**. За пределами этой функции переменные с этими именами оказываются либо неопределенными, либо сохраняют ранее присвоенное значение.

```
> restart;
> x:=0;y:=0;
              x := 0
              y := 0
> m:=(x,y)→sqrt(x^2+y^2);
                              2    2
              m := (x, y) → sqrt(x  + y )
> m(3,4);
              5
```

```
> m(3.,4.);
                5.000000000
> x;y;
                0
                0
```

3) Еще один способ задания функции пользователя базируется на применении функции **unapplay**. Её синтаксис:
name:=unapplay(expr, var1,var2,...),
где

name - имя функции (без аргументов),
expr - непосредственно выражение функции через аргументы **var1,var2,...** .
Обращение к функции осуществляется в виде **name(var1,var2,...)**. Переменные, указанные в качестве формальных параметров являются локальными.

```
> restart;
> x:=1;y:=1;
              x := 1
              y := 1

> fm:=unapply(sqrt(x^2+y^2),x,y);
                           2    2 1/2
              fm := (x, y) → (x  + y )

> evalf(fm(3,4));
              5.000000000

> fm(sin(x),cos(x));
                   2         2 1/2
              (sin(x)  + cos(x) )

> simplify(%);
                1

> x:=x;y:=y;
                x := 1
                y := 1
```

Замечание. Для задания сложных выражений можно использовать последовательное определение частей выражения с помощью простого присваивания (1) (при этом переменным не должно быть присвоено никакое значение), а затем, используя (2) или (3), определить сложную функцию, представляющую выражение от ранее определенных (в пункте(1)) выражений.

```
> a:=x^2+1;
                   2
              a := x  + 1

> b:=y^2+3;
                   2
```

```
        b := y  + 3

> fl:=unapply(a,x);
                    2
        fl := x → x  + 1

> ff:=unapply(a+b,x,y);
                      2       2
        ff := (x, z) → x  + 4 + y

> f2:=unapply(a^2/b,x,y);
                        2    2
                      (x  + 1)
        f2 := (x, y) → ---------
                        2
                       y  + 3

> f2(x,1);
                 2    2
        1/4 (x  + 1)
```

4.2 Условные выражения.

Обычно программы реализуют некоторый алгоритм решения задачи. В любом алгоритме возникает необходимость выполнения определенной последовательности операторов в зависимости от истинности или ложности того или иного выражения. Для подготовки разветвляющихся программ в Maple-язык программирования включен оператор **if**, позволяющий создавать следующую конструкцию:

if булево выражение **then** последовательность операторов
|**elif** булево выражение **then** последовательность операторов |
|**else** последовательность операторов| **end if**;

В вертикальных черточках | | указаны необязательные элементы.

Этот оператор работает следующим образом: если истинно булево выражение после ключевого слова **if** , то выполняется последовательность операторов после ключевого слова **then** до первого встретившегося **elif, else** или **end if**; если значение булева выражения равно **false** или **FAIL**, то проверяется на истинность выражение после ключевого слова **elif**, если оно задано, и в случае истинности выполняются операторы после второго ключевого слова **then** .

```
> x:=0.25;
                    x := .25
> if x<0 then g:=-1
  elif x<1 then g:=0
  else g:=1
  end if;
                    g := 0
```

Если ни одно из булевых значений не истинно, то выполняются операторы блока **else** , опять таки в случае его задания. Блоков **elif** может быть сколько угодно, тогда как блок**else** всегда только один.

Оператор **if** в предыдущих версиях Maple завершался ключевым словом **fi** . Для совместимости с предыдущими версиями его можно использовать и в Maple 6, однако при написании новых программ рекомендуется для завершения конструкции ветвления использовать ключевое слово **end if** .

```
> if x=0 then 0
  elif x=1 then 1
  elif x=2 then 2
  else print("Out of
  range!")
  end if;
      "Out of range!"
```

В Maple нет оператора, реализующего конструкцию переключателя. Для этих целей следует использовать оператор **if** с несколькими блоками **elif**.

Синтаксис Maple позволяет использовать вложенные конструкции **if**, т.е. последовательности операторов в блоках **then** и **else** могут содержать операторы ветвления.

На практике чаще всего используются следующие конструкции оператора **if** :

if (условие сравнения) **then** (элементы) **fi**;
if (условие сравнения) **then** (элементы) **else** (элементы) **fi**;

В условиях используются любые логические конструкции со знаками сравнения **(<,<=,>,>=,=,<>)** и логические операторы **and**, **or**, **not**, конструкции с которыми возвращают логические значения true и false.

Для организации ветвлений в программах и отдельно в виде команды может быть использована операция **`if`** (в ОБРАТНЫХ кавычках). Она имеет следующий синтаксис:

`if `(Условие, Выражение1, Выражение2)

Если **Условие** истинно, то будет выполнено **Выражение1**, в противном случае выполняется **Выражение2**.

```
> a:=4; b:=3;
                          a := 4
                          b := 3
> c:=`if`(a>b,a,b)+sin(`if`(a>b,a,b));
                      c := 4 + sin(4)
```

```
> 5*(Pi + `if`(a > b,a,b));
      5 Pi + 25
> r:=5;
          r := 5
> `if`(r<0,print('Negative'),print('Positive'));
          Positive
```

Ввиду компактности записи последняя форма условного выражения бывает предпочтительна, хотя она и менее наглядна.

4.3 Циклы

Для организации повторяющихся вычислений в Maple V предусмотрены две формы оператора цикла: **for-from** и **for-in**. Первый оператор цикла является универсальным и включает в себя как циклы, повторяющиеся заданное число раз, так и циклы, выполняющиеся,

пока некоторое булево выражение является истинным. Вторая форма цикла **for** реализует цикл по элементам списка или множества, и в других языках программирования он известен как цикл **for-each**.

Цикл **for-from** задается следующим образом:

|**for** имя| |**from** выражение1| |**by** выражение2| |**to** выражение3| |**while** булево выражение| **do** (последовательность операторов) **end do**;

имя - имя управляющей переменой цикла,
выражение1, выражение2, выражение3- выражения, задающие начальное значение переменной **name**, шаг ее изменения и конечное значение переменной **name**,
булево выражение - выражение, задающее условие, пока будет выполняться цикл (набор объектов между словами **do** и **end do**).

```
> # Не выполняется ни разу
> for i from 1 to -1
  do evalf(sqrt(i)) end do;
>
> # Выполняется 2 раза
> for i from 1 to 2
  do evalf(sqrt(i)) end do;
                    1.
                1.414213562
> # Определяет первое простое число,
  большее 500000, но меньшее 500010
> for i from 5*10^5 to 500010
  while not isprime(i) do end do;
> i;
                   500009
```

В ходе выполнения цикла управляющая переменная меняется от значения **выражение1** до значения **выражение3** с шагом, заданным **выражение2**. Если блок **by** отсутствует, то управляющая переменная будет меняться с шагом +1 при **выражение1>выражение3**. Выполнение цикла в обратном порядке, когда**выражение1<выражение3** не предусмотрено.
Цикл можно прервать с помощью дополнительного блока **while**. Цикл с таким блоком просчитывается до тех пор, пока истинно **булево выражение**, стоящее после ключевого слова **while**.
В последнем примере, обратите внимание, тело цикла является пустым.

При задании оператора цикла **for-from** обязательным является только блок **do**, определяющий тело цикла, причем он может быть единственным блоком цикла:

do последовательность операторов **end do**

Подобная конструкция определяет бесконечный цикл, прервать выполнение которого может только один из операторов: **break, return, quit** или возникновение ошибки.

Все остальные блоки цикла **for-from** являются необязательными и могут задаваться в произвольном порядке за одним исключением: если присутствует блок **for**, то он должен быть задан первым. Если какой-либо блок не задан, то его параметры по умолчанию принимают следующие значения:

for - фиктивная переменная
from - 1
by - 1
to - infinity
while - true

Если в операторе цикла отсутствуют все необязательные блоки за исключением блока **while**, то получается классический цикл **while** с предусловием: сначала проверяется ис-

тинность булева выражения условия, а затем, в зависимости от результатов проверки, либо выполняются операторы тела цикла (условие истинно), либо цикл завершает свою работу (условие ложно).

```
> x:=2;
```

$$x := 2$$

```
> while x>1/2 do x:=x/2 end do;
```

$$x := 1$$

$$x := \frac{1}{2}$$

При использовании цикла **while** надо внимательно следить за тем, чтобы в теле цикла изменялись переменные, входящие в выражение условия, так как иначе цикл будет выполняться бесконечно.

Циклы могут быть вложенными. Это иллюстрирует следующий пример, в котором сначала создается пустая матрица, а затем определяются значения её элементов.

```
> M:=array(1..3,1..3):
> for i to 3 do for j to 3 do
> M[i,j]:=i^2-j^2 end do end do;
> print(M);
```

$$\begin{bmatrix} 0 & -3 & -8 \\ 3 & 0 & -5 \\ 8 & 5 & 0 \end{bmatrix}$$

```
> evalm(M);# Другой способ вывода матрицы
```

$$\begin{bmatrix} 0 & -3 & -8 \\ 3 & 0 & -5 \\ 8 & 5 & 0 \end{bmatrix}$$

Цикл **for-in** организует цикл по элементам объекта, который может быть представлен последовательностью, списком, множеством, суммой, произведением или строкой. Его общий синтаксис имеет вид:

|**for** имя| |**in**выражение1| |**while**(булево выражение)| **do** последовательность операторов **od;**

```
> s:=1:
> for z in sin(x)+cos(x)
  do s:=s*z end do;
```

$$s := \sin(x)$$

$$s := \sin(x)\cos(x)$$

```
> S:={x,y,z}:s:=0:
> for i in S do s:=s+i end do:s;
```

$$z + x + y$$

Переменная цикла **name**, определяемая в блоке **for...in**, последовательно принимает значения операндов объекта, как они определяются командой **op( )**. Цикл выполняется столько раз, сколько операндов задано в объекте, если только булево выражение в необязательном блоке **while** не станет ложным раньше, чем будут последовательно перебраны все операнды объекта. В цикле данного вида переменная может меняться произвольно, принимая убывающие и отрицательные значения.

4.4 Операторы пропуска и прерывания.

Иногда бывает нужным пропустить определенный цикл или вообще прекратить выполнение цикла. Для этого совместно с оператором условия **if** используются операторы:

next - обеспечивает пропуск определенной итерации цикла
break - прерывает выполнение фрагмента программы (или цикла), как только он встречается в ходе её выполнения

```
> S:={x,y,z}:s:=0:j:=1:
> for i in S do
  s:=s+i;
  if j>2 then break;end if;
  j:=j+2;
  end do:
  s;j;
```

```
x+y
  3
```

```
> for i from 1 to 5 do
  if is(i,even) then next
  end if;
  i; end do;
```

```
1
3
5
```

Любой из операторов **quit, done, stop** - прерывает выполнение текущей программы, но при этом окно текущего документа закрывается и все имеющиеся в нем определения исчезают.

4.5 Процедуры.

Процедурами называют модули программы, имеющие самостоятельное значение.

Определение процедуры Maple имеет следующий общий синтаксис:

proc (|список формальных параметров|)

|**local** список локальных параметров |
|**global** список глобальных параметров|
|**option** список опций|
|**description** строка описания|
последовательность операторов
end proc;

При объявлении процедуры единственным обязательным параметром является последовательность операторов, формирующих тело процедуры. Остальные параметры, определяющие локальные и глобальные переменные, список формальных параметров, задающие специальные опции режима выполнения процедуры и строку описания, могут полностью отсутствовать.

В предыдущих версиях Maple завершающим процедуру оператором был **end;** или **end:** . Последовательность ключевых слов **end proc;** в предыдущих версиях недопустима.

Простейшая форма задания процедуры следующая:

name:= **proc**(Параметры)
  Тело процедуры
end;

Процедуры имеют имя и список параметров (даже если он пустой).

Процедуры вызываются указанием их имени со списком фактических параметров.

name(Фактические параметры);

```
> vect_length:=proc(x::numeric,y::numeric)
  sqrt(x^2+y^2)
  end proc;
   vect_length := proc(x::numeric, y::numeric) sqrt(x^2 + y^2) end proc
> vect_length(2,5);
                              √29
> vect_length(a,b);
Error, vect_length expects its 1st argument, x, to
be of type numeric, but received a
```

Параметры процедуры задаются перечислением имен переменных **proc(x,n,a);** с помощью знака **::** после имени переменной можно определить ее тип **proc(n::integer).** При вызове процедуры фактические параметры подставляются на место формальных.

Несоответствие фактических параметров типу заданных переменных ведет к сообщению об ошибке и к отказу от выполнения процедуры.
Переменные, указанные в списке формальных параметров, являются локальными, т.е. их изменение происходит только в теле процедуры.

```
> mXYZ:=proc(x,y,z)
  if x>y then x else z
  end if
  end proc;
   mXYZ := proc(x, y, z) if y < x then x else z end if end proc
> mXYZ(1,2,3,4,5);
                              3
> mXYZ(6,3);
                              6
> mXYZ(3,6);
Error, (in mXYZ) mXYZ uses a 3rd argument,
z, which is missing
```

Количество фактических параметров не обязательно должно быть равно количеству формальных параметров процедуры. Если их меньше, то ошибка при выполнении процедуры возникнет только тогда, когда при вычислении тела процедуры действительно потребуется значение этого отсутствующего параметра. Если фактических параметров больше, чем формальных, то никакой ошибки не будет сгенерировано - дополнительные параметры будут просто проигнорированы.

Для явного определения переменных в теле процедуры локальными служит ключевое слово **local**, а глобальными - ключевое слово **global**.

```
> m:=0;
        m := 0

> modc:=proc(z)
> m:=evalf(sqrt(Re(z)^2+Im(z)^2))
> ### WARNING: `m` is implicitly declared local
> end;
Warning, `m` is implicitly declared local

  modc := proc(z) local m;
  m := evalf(sqrt(Re(z)^2 + Im(z)^2)) end
```

В данном примере Maple выводит сообщение о том, что переменная **m** должна быть объявлена локальной, и сам же это делает в строке вывода.

Иногда бывает необходимо, чтобы некоторые переменные после выполнения процедуры приняли значения, полученные в ходе ее выполнения. Тогда в процедуре эти переменные надо объявить глобальными.Это иллюстрирует следующий пример:

```
> a:=1;b:=1;
          a := 1
          b := 1

> fg:=proc(x,y) global a,b;
> a:=x^2;b:=y^2;
> return(sqrt(a+b));
> end;

  fg := proc(x, y)
global a, b;
   a := x^2; b := y^2; return(sqrt(a + b))
end

> fg(3,4);
          5

> a;b;
          9
          16
```

Обычно процедура возвращает значение последнего выраженияв ее теле. Для вывода какого-либо другого значения используют оператор возврата **return**

```
> modc:=proc(z)
> evalf(sqrt(Re(z)^2+Im(z)^2));
> return(Re(z));
> end;

  modc := proc(z) evalf(sqrt(Re(z)^2 + Im(z)^2)); return(Re(z)) end

> modc(3.+I*4.);
          3.
```

При профессиональной подготовке процедур пользователь должен предусмотреть поведение процедуры при возможных ошибках и выдачу соответствующего сообщения. Для этого используется оператор**error**:

error строка сообщения |, сприсок параметров|,

где **строка сообщения** - это строка, которая участвует в формировании общего сообщения об ошибке в процедуре после ее завершения оператором **error**

Если в ней заданы специальные переменные, начинающиеся с символа процента **(%)**, за которым следует целое число без знака или со знаком минус, то вместо нее в строку сообщений подставляется значение переменной из **списка параметров**, причем абсолютное значение целого числа соответствует порядковому номеру переменной в списке. В случае целого числа со знаком минус значение соответствующего параметра преобразуется в порядковое числительное (с точки зрения английского языка) и подставляется в строку сообщений.

```
> k:=6:
> error "%1ий параметр больше нуля," " равен to %2", 3, k;
Error, 3ий параметр больше нуля, равен to 6
```

Специальный параметр **%0** будет отображать через запятую и пробел все значения переменных из **списка параметров**.

```
> error "Проверьте все параметры: %0", a,b,c;
Error, Проверьте все параметры: a, b, c
```

Наиболее часто этот оператор просто выводит сообщение об ошибке в виде:

error ('string'), где **string** - строковое сообщение.

> f := proc (x) if x<0 then error(`invalid x`, x) else x^(1/2) fi end:
> f(-3);
Error, (in f) invalid x, -3

В объявлении процедуры можно определить опции, включаемые словами:

option (список опций через запятую)

remember - определяет таблицу памяти для процедуры
builtin - определяет встроенную процедуру Maple
system - определяет процедуру как системную
operator, arrow - определяет процедуру в нотации функционального оператора
trace - задает в процедуре вывод отладочной информации
package - используется для придания процедуре статуса пакетной
copyright - защищает процедуру от просмотра (запрещает отображать операторы тела процедуры.)

Опция **builtin** применяется для идентификации встроенных процедур Maple, находящихся в его ядре и реализующих разнообразные команды и функции. При полном вычислении подобных функций командой **eval( )** опция **builtin** информирует пользователя о том, что команда является встроенной, а отображаемый уникальный номер служит для ее быстрой идентификации и выполнения.

```
> f1:= proc(x) h(x) end proc;
        f1 := proc(x) h(x) end proc
> f2:=proc(x) option operator;
  h(x); end proc;
                f2 := h
```

Обычно при объявлении процедуры Maple автоматически производит некоторые упрощения в ее теле. Опция **operator** предписывает дальнейшие упрощения в теле процедуры, рассматривая ее как математический оператор.

```
> proc(x,y)
  option operator,
  arrow;
  sqrt(x^3+y^3);
  end proc;
```

$$(x,y) \to \sqrt{x^3+y^3}$$

Используемая совместно с опцией **operator** опция **arrow** предписывает отображать процедуру как функциональный оператор с использованием "стрелочной" нотации и эквивалентна заданию процедуры в области вывода с помощью стрелки.

```
> F:=proc(n::integer)
  if n<2 then n
  else F(n):=F(n-1)+F(n-2)
  end if
  end proc;
```

```
F := proc(n::integer)
  if n < 2 then n
  else F(n) := F(n - 1) + F(n - 2)
  end if
end proc
> F(2000);
Error, (in F) too many
levels of recursion
> FR(2000);
42246963333923048787067256023\
   41482782579852840250681098010\
   28013731430858437013070722412\
   35996391415110884460875389096\
   03607640194711643596029271983\
   31259873732625355580260699158\
   59152294924539049987222567953\
   16982874482472992263901833716\
   77806060701161549788671987985\
   83114688708762645973690867228\
   84023654422295243347964480139\
   51534956297208765265606952980\
   64998419774487201556128026654\
   04554171717881930324025204312\
   082516817125
```

Опция **remember** . При вычислениях может оказаться так, что при реализации некоторого алгоритма необходимо много раз вызывать процедуру с одним и тем же набором фактических параметров. Для ускорения процесса ее вычисления она может быть сконструирована так, что результаты ее вычисления с заданными параметрами автоматически записываются в специальную *таблицу значений*. При последующем обращении к процедуре с этим набором фактических параметров ее значение не вычисляется вновь, а берется из таблицы значений. Для конструирования процедуры, записывающей результаты своего вычисления в таблицу, достаточно задать в ее теле опцию **remember**.

Использование этой опции особенно полезно при определении рекурсивных процедур (в теле которых имеется обращение к самой себе).
На рисунке слева вычисляется 2000 число Фибоначи с помощью процедуры **F** предыдущего примера (она переименована в **FR**), в которую вставлена опция **remember**. Сначала вычисляется **FR(1000)**, а затем вычислено значение **FR(2000)**. Обратный слэш в области вывода указывает на перенос строки. Представляете, что это за число!

Таблица значений процедуры представлена четвертым операндом типа **procedure** , т.е. ее можно отобразить, выполнив команду **op(4, eval(F))**, где **F** - имя процедуры.

Для удаления записи из таблицы значений достаточно соответствующему элементу присвоить его же собственное имя с помощью команды **evaln( )** . Удалить полностью всю таблицу значений процедуры можно, подставив значение **NULL** в четвертый операнд ее типа **procedure**.

Опция **system** используется совместно с опцией **remember** и идентифицирует процедуру как "системную функцию", для которой таблица значений удаляется во время процедуры сборки мусора, которая автоматически запускается системой Maple. Если для какой-либо пользовательской процедуры эта опция не указана, то при сборке мусора ее таблица значений не удаляется из памяти. Самостоятельно инициировать процедуру сборки мусора можно, выполнив команду **gc()** (garbage cjllection) без параметров.

```
> f:=proc(x,y)
  option Copyright;
  x^2+y^2
  end proc;
      f := proc(x, y) ... end proc
> f:=proc(x,y)
  option Copyright;
  description
  `Length_of_vector`;
  x^2+y^2
  end proc;
f := proc(x, y)
 description Length_of_vector
    ...
end proc
```

Любая строка в списке опций, начинающаяся со слова "Copyright", трактуется Maple как опция **Copyright**, которая запрещает отображать операторы тела процедуры, если только значение переменной интерфейса **verboseproc**, отвечающей за отображение на рабочем листе текста процедуры, не установлено равным 2 или больше.

Последним в заголовке процедуры может быть задана строка описания в операторе **description.** Она не влияет на выполнение процедуры и единственное ее предназначение предоставить строку комментария, которая отображается в области вывода при объявлении процедуры или ее чтения из файла или библиотеки. Дело в том, что при распечатке процедуры из ее тела удаляются все комментарии и только эта строка может дать пользователю информацию о предназначении процедуры. Более того, если процедура определена с опцией **Copyright**, то эта строка является единственной отображаемой из всех операторов тела процедуры.

Опция **trace** задает в прцедуре вывод кроме результатов вычислений и отладочной информации при обращении к процедуре.

4.6 Средства отладки процедур, их сохранение и использование (подключение).

Одним из основных средств отладки процедур является функция трассировки **trace(имя процедуры)** . Детальность ее работы задается системной переменной **printlevel:=n** (уравень вывода). При **n=1**результат выводится только для непосредственно исполняемой функции или оператора. Для вывода информации о выполнении k-го уровня вложеннсти надо использовать знечения этой переменной от 5*k до 5*(k+1). Так, при **n** от 1 дл 5 выводятся результаты трассировки первого уровня, при **n** от 6 до 10 - второго и т.д. Максимальное значение **n=100** обеспечивает трассировку по всем уровням вложенности процедуры.

Для отладки процедур в системе Maple имеется специальный интерактивный отладчик - **debugger**. Для включения отладчика в работу необходимо подать команду:

stopat(имя процедуры)

Признаком, указывающим на работу отладчика, является изменение приглашения к вводу со знака **>** на **DBG>.** Теперь, подавая команды **next** (следующий), **step** (шаг) и **stop** (остановка), можно проследить выполнение процедуры.

С помощью команды **showstat (имя процедуры)** можно вывести ее листинг, причем в нем строки вычисляемых элеменотов пронумерованы, что весьма удобно при разборе работы процедуры.

Существует множество других команд отладчика. Справку по ним можно получить выполнив команду **?Debugging**

Когда процедура отлажена и оттестирована, ее можно сохранить в файле с расширением **m** командой **save()**, которая позволяет сохранить любой объект Maple.

```
> save f, "c:\\Мои документы\\Myproc.m";
```

Впоследствии, когда для вычислений потребуется данная процедура, ее можно прочитать в другом сеансе Maple командой **read()** .

```
> read "c:\\Мои документы\\Myproc.m";
> f(2,4);
                    20
```

Задание 4.

1. Определите функцию *f1(x)*, равную x^2, если $x^2 >= 4$, и равную $-2*x^2$, если $x^2<4$, используя оператор **`*if*`**.
   Определите функцию *f2(x)* (по тем же формулам), используя команду ***piecewise*** (см. справку Maple).
   Вычислите *r = f1(4) - f2(4)*.
2. Создайте пустую матрицу M размерности (4 x 4). С помощью оператора цикла заполните матрицу: при *i = j M [i, j] = 1*(единице), при *i* отличном от *j - M [i, j] = i+j*. Используя функцию **map**, определите матрицу MM, каждый элемент которой MM[i,j]=exp(M[i,j]). Выведите матрицу MM на экран, вычислив значение каждого элемента в формате с плавающей точкой (т.е до числа)
3. Создайте список *S*, содержащий целые числа от -5 до 5. С помощью операторов *if* и цикла организуйте создание массива *SA* следующим образом:
   - если элемент списка *S* положителен и четный, то соответствующий элемент массива *SA* определяется делением этого элемента на 2;
   - если элемент списка *S* положителен и нечетный, то соответствующий элемент массива *SA* определяется умножением этого элемента на 2;
   - если элемент списка *S* отрицателен и четный, то соответствующий элемент массива *SA* определяется делением этого элемента на (- 2);
   - если элемент списка *S* отрицателен и нечетный, то соответствующий элемент массива *SA* определяется делением этого элемента на (- 2);
4. Составьте процедуру Differ(y,n), вычисляющую n-ую производную от произвольной функции y(x). Определите n как целое, и предусмотрите сообщение об ошибке, если n не является положительным. Проверьте работу процедуры на любой функции. Сохраните процедуру Differ в вашей директории на диске.
5. Закройте, сохранив, ваш текущий рабочий файл. Создайте новый документ. Подключите процедуру Differ и вычислите с ее помощью 2-ую производную от функции e^{2x}.

Глава 5.
Решение алгебраических уравнений и систем уравнений.

5.1 Основная функция solve.
5.2 Одиночные нелинейные и тригонометрические уравнения.
5.3 Системы нелинейных и трансцендентных уравнений.
5.4 Решение неравенств.
5.5 Решение уравнений в численном виде.
5.6 Решение функциональных, рекуррентных и др. уравнений.

5.7 Функция RootOf.

5.1 Основная функция solve.

Для решения линейных и нелинейных уравнений в аналитическом виде используется достаточно универсальная и гибкая функция

solve(eqn, var)
solve({eqn1, eqn2,...}, {var1, var2,...})

где **eqn**-уравнение, содержащее функцию ряда переменных, **var**-переменная, по которой ищется решение. Если решение содержит повторяющиеся выражения, Maple V обозначает их как %1, %2 и т.д. и выводит решение с такими переменными.
Характер решений можно изменить с помощью глобальных переменных:
_SolutionsMayBeLost - при значении true дает решение при возврате функцией solve значения **NULL**;
_MaxSols - задает максимальное число решений;
_EnvAllSolutions - при значении **true** задает выдачу всех решений.

В решениях помимо переменных **%N** могут встречаться следующие обозначения:
_NN - указывает на неотрицательные решения;
_B - указывает на решения в бинарной форме;
_Z - указывает на то, что решение содержит целые числа.

5.2 Одиночные нелинейные и тригонометрические уравнения.

Решение одиночных нелинейных уравнений вида **f(x)=0** легко обеспечивается функцией **solve(f(x),x)**.Например:

```
> solve(x^3-2*x+1,x);
     1/2            1/2
1,1/25   -1/2,-1/2-1/25
> solve(x^(3/2)=3,x);
 2/3
3
```

Функция **solve** старается дать решение в аналитическом виде. Для получения корней в численном виде надо использовать функцию **evalf** или **convert**.

```
> s:=solve(sqrt(ln(x))=2,x);
> s1:=evalf(s);
> convert(s,float);
            s := exp(4)
            s1 := 54.59815003
            54.59815003
```

Часто бывает удобно представлять уравнение и его решения в виде отдельных объектов, отождествленных с определенной переменной. Это позволяет в дальнейшем использовать

полученные решения в других вычислениях. В частности, это позволяет легко проверить решение подстановкой **subs**:

```
> e1:={x^3-2*x+1=0};
> s:=solve(e1,x);
                    3
          e1 := {x  - 2 x + 1 = 0}
          s := {x = 1}, {x = -1/2 + 1/2 sqrt(5)},
              {x = -1/2 - 1/2 sqrt(5)}

> subs(s[2],e1);evalf(%);
                          3
         {(-1/2 + 1/2 sqrt(5))  + 2 - sqrt(5) = 0}
         {0 = 0}
```

Если результат решения представлен через функцию **RootOf**, то получить все корни можно через функцию **allvalues**.

```
> solve({exp(x)=sin(x)},x);
          {x = RootOf(_Z - ln(sin(_Z)))}

> allvalues(%);
          {x = .3627020561 - 1.133745919 I}

> a:=solve({x^5-2*x=-3},x);
                        5
         a := {x = RootOf(_Z  - 2 _Z + 3)}

> allvalues(a);
        {x = -1.423605849},
        {x = -.2467292569 - 1.320816347 I},
        {x = -.2467292569 + 1.320816347 I},
        {x = .9585321812 - .4984277790 I},
        {x = .9585321812 + .4984277790 I}
```

Сводящиеся к одному уравнения вида **f1(x)=f2(x)** также решаются функцией **solve(f1(x)=f2(x),x)**.

При решении **тригонометрических** уравнений периодичность тригонометрических функций и связанная с этим множественность решений не учитывается.

```
> solve( sin(x)=cos(x)-1, x );
                - 1/2 Pi, 0
```

В данном примере найдено одно главное решение. Используя системную переменную **_EnvAllSolutions**, которой присваивается значение **true** (по умолчанию - **false**), можно получить все решения.

```
> _EnvAllSolutions := true:
> solve( sin(x)=cos(x)-1, x );

        - 1/2 Pi + 2 Pi _Z~, 2 Pi _Z~
```

В данном примере переменные _Z~ означают ряд натуральных чисел. Таким образом, получены все решения уравнения с учетом периодичности тригонометрических функций.

5.3 Системы нелинейных и трансцендентных уравнений.

При решении систем уравнений они и список переменных задаются как множества, т.е. в фигурных скобках. При этом и результат решения получается в виде множества.

```
> sys:={x*y=c,x+y=d};
          sys := {x y = c, x + y = d}

>s1:=solve(sys,{x,y});                                                    2
 s1 := {x = -RootOf(_Z  - _Z d + c) + d,
                  2
       y = RootOf(_Z  - _Z d + c)}

> s2:=allvalues(%);

  s2 := {y = 1/2 d + 1/2 %1, x = 1/2 d - 1/2 %1},
       {y = 1/2 d - 1/2 %1, x = 1/2 d + 1/2 %1}
            2
  %1 := sqrt(d  - 4 c)

> r1:=s2[1];s2[2];
                          2
    r1 := {y = 1/2 d + 1/2 sqrt(d  - 4 c),
                         2
          x = 1/2 d - 1/2 sqrt(d  - 4 c)}

                                                          2
         {y = 1/2 d - 1/2 sqrt(d  - 4 c),
                        2
          x = 1/2 d + 1/2 sqrt(d  - 4 c)}
```

Для того чтобы преобразовать его к обычному решению, нужно использовать функцию **assign**, которая обеспечивает присваивание переменным значений, взятых из множества.

```
> assign(s2[1]);x;y; # Присвоение первого решения
                        2
            1/2 d - 1/2 sqrt(d  - 4 c)
                        2
            1/2 d + 1/2 sqrt(d  - 4 c)
```

> subs(s2[2],sys);expand(%);#Проверка 2-го решения

```
    {d = d,
              2                        2
(1/2 d - 1/2 sqrt(d  - 4 c)) (1/2 d + 1/2 sqrt(d  - 4 c)) = c}
    {c = c, d = d}
```

5.4 Решение неравенств.

Решение неравенств с помощью функции **solve** осуществляется также, как и уравнений (вместо равенств вводятся неравенства).

```
> solve(5*x>10,x);
            RealRange(Open(2), infinity)

> solve(x^2-2*x-1>0,x);
    RealRange(-infinity, Open(1 - sqrt(2))),
    RealRange(Open(1 + sqrt(2)), infinity)

> solve({x*y*z>0,y+z>10,x>-1},{x,y,z});
    {10 < z, -1 < x, y = 0}, {-1 < x, 10 < y, z = 0}
```

5.5 Решение уравнений в численном виде.

Функция **solve** имеет несколько производных от нее функций. Одна из них:

fsolve (eqns,vars,options)

- сразу дает решения в форме действительных или комплексных чисел
Некоторые из опций, используемые этой функцией:
complex - находит один или все корни полинома в комплексной форме
maxsols = n - задает нахождение только **n** корней
interval - задается в виде **a..b** или **x = a..b** или **{x = a..b, y = c..d}** и обеспечивает поиск корней в указанном интервале.

Локализация поика корней позволяет отыскать решения, которые не дают функции **solve** и **fsolve**.

```
> fsolve(sin(x)=0.5);
             .5235987756

> fsolve(sin(x)=0.5,x=4..8);
              6.806784083

> fsolve(x^5-x,x);
          -1., 0, 1.000000000
```

```
> fsolve(x^5-x,x,complex);
        -1., -1. I, 0, 1. I, 1.000000000

> fsolve(x^5-x,x,complex,maxsols=2);
        -1., -1. I
```

5.6 Решение функциональных, рекуррентных и др. уравнений.

Рекуррентными называются уравнения, у которых решение на заданном шаге находится по одному или нескольким предшествующим шагам. Для решения таких уравнений в Maple V существует функция**rsolve**:

rsolve(eqns,fcns)

eqns - одиночное уравнение или система уравнений
fcns - функция, имя функции или множество имен функций.

```
> rsolve({y(n)*y(n-1)+y(n)-y(n-1)=0,
>y(0)=a},y);
              a
            -------
            1 + n a

> rsolve({y(n+1) + f(n) = 2*2^n + n,
> f(n+1) - y(n) = n - 2^n +3,
  y(k=1..5)=2^k-1, f(5)=6}, {y, f});

             n
     {y(n) = 2  - 1, f(n) = n + 1}
```

Для решения целочисленных уравнений служит функция **isolve**, для отыскания решений по модулю **m** функция **msolve**. Более подробно с работой этих функций можно ознакомиться в справочной системе Maple V.

Решение **функционального** уравнения, содержащего некоторую функцию **f(x)** в составе равенства, заключается в нахождении этой функции. Функция **solve** с указанием **f** или **f(x)** в качестве независимой переменной **var** успешно справляется с этой задачей. Причем указание только имени функции (без переменной) в качестве **var** приводит к созданию процедуры, и для получения вида найденной функции требуется предпринять дополнитель-

ные шаги (например, использовать функцию **allvalues** или **convert**).

```
> A:=solve(f(x)^2-x+1,f);# var=f
```

$$A := \mathbf{proc}(x)\ \mathrm{RootOf}(\_Z^2 - x + 1)\ \mathbf{end}$$

```
> allvalues(A(x));
```

$$\sqrt{x-1}, -\sqrt{x-1}$$

```
>
> C:=solve(f(x)*x^2=a*x^2+b*x+c,f);
```

$$C := \mathbf{proc}(x)\ (a*x^2 + b*x + c) / x^2\ \mathbf{end}$$

```
> convert(C(x),radical);
```

$$\frac{a\,x^2 + b\,x + c}{x^2}$$

```
> A:=solve(f(x)^2-x+1,f(x));# var=f(X)
```

$$A := \sqrt{x-1}, -\sqrt{x-1}$$

<=”” p=””>

5.7 Функция RootOf.

В решениях уравнений нередко появляется функция **RootOf**, означающая, что корни нельзя выразить в радикалах. Эта функция применяется и самостоятельно в виде:

RootOf(expr) или **RootOf(expr,x)**

где **expr**-алгебраическое выражение или равенство, имя переменной, относительно которой ищется решение. Если **x** не указана, ищется универсальное решение по переменной **_Z**. Когда **expr** задано не в виде равенства, решается уравнение **expr=0**. Для получения решений вида **RootOf** в явном виде может использоваться функция **allvalues**. Например:

```
> RootOf(x^3-1,x) mod 7;
       3
RootOf(_Z +6)
> allvalues(");
  1/3      1/3      1/2 1/3     1/3      1/2 1/3
-6   , 1/26  ,-1/2I3   6   ,1/26   +1/2I3   6
> evalf(");
-1.817120593, .9085602965-1.573672596I, .9085602965+1.573672596I
```

Лекция 6.
Решение дифференциальных уравнений и систем уравнений.

6. Основные возможности системы Maple V .
6.1 Обыкновенные дифференциальные уравнения.
6.2 Численное решение обыкновенных диф. уравнений.
6.3 Пакет DEtools решения обыкновенных дифференциальных уравнений.
6.4 Пакет PDEtools для работы с дифференциальными уравнениями в частных производных.
6.5 Задания.

6. Основные возможности системы Maple V R6.

Решение дифференциальных уравнений самых различных типов - одно из достоинств системы Maple V. Для этой цели могут быть использованы функции ядра системы (**dsolve, pdsolve, pdesolve**), инструментальные пакеты **DEtools** (решение обыкновенных диф. уравнений) и **PDEtools** (решение диф. уравнений в частных производных).Кроме этого существует множество функций, позволяющих классифицировать уравнения, производить замены в диф. уравнениях, осуществлять преобразование решений, визуализировать полученные решения (строить графики различных типов).

6.1 Обыкновенные дифференциальные уравнения.

Для решения обыкновенных дифференциальных уравнений и системы простых дифференциальных уравнений используется следующая функция ядра системы:

dsolve(deqn)
dsolve({deqn,InC}, var)
dsolve({deqns,Inc},{vars})
dsolve({deqns,InC},{vars}, option)
где **deqn(s)**-одно дифференциальное уравнение или система из дифференциальных уравнений первого порядка,
Inc начальные условия,
var(s) - переменная или переменные, относительно которых ищется решение,
option-необязательный параметр, указывающий на метод решения.

Параметр **option** задает один из методов решения:
exact - аналитическое решение (принято по умолчанию);
explicit - решение в явном виде;
laplace - решение через преобразование Лапласа;
series - решение в виде ряда с порядком, указываемым знчением переменной **Order** (указывается до обращения к функции **dsolve**);
numeric - решение в численном виде.

```
> ode1 :=
  diff(y(x),x)-y(x)^2+y(x)*sin(x)-cos(x);
```

$$ode1 := \left(\frac{\partial}{\partial x} y(x)\right) - y(x)^2 + y(x)\sin(x) - \cos(x)$$

```
> ans1 := dsolve(ode1);
```

$$ans1 := y(x) = \sin(x) - \frac{\_C1\, e^{(-\cos(x))}}{\_C1 \int e^{(-\cos(x))}\, dx + 1}$$

```
> restart;
> ode2:=(D@@2)(y)(x)+y(x);
```

$$ode2 := (D^{(2)})(y)(x) + y(x)$$

```
> dsolve(ode2,y(x));
```

$$y(x) = \_C1\cos(x) + \_C2\sin(x)$$

```
> assign(%);y(x);
```

$$\_C1\cos(x) + \_C2\sin(x)$$

```
> g(x):=diff(y(x),x);
```

$$g(x) := -\_C1\sin(x) + \_C2\cos(x)$$

Для решения задачи Коши в **InC** надо задать начальные условия, а при решении краевых задач - краевые условия. Если Maple способна найти решение при числе начальных или краевых условий меньше порядка системы, то в решении будут появляться неопределенные константы вида **_C1,_C2** и т.д. Они же могут быть при аналитическом решении системы, когда начальные условия не заданы. Если решение найдено в неявном виде, то в нем появится параметр **_T**. Для присвоения полученного решения искомой функции используется команда **assign**.

Производные при записи дифференциальных уравнений и начальных/граничных условий могут задаваться функцией **diff** или оператором **D**.
Если условие на производную задается в **произвольной** точке **x=b**, то для его задания может быть использован оператор **diff**: например, **diff(y(b),b)=1**. Условия на производные в **конкретной** точке должны быть записаны с помощью дифференциального оператора **D: D(y)(0)=5, D(D(y)(4))=7**.

```
> de:=m*diff(y(x),x$2)-k*diff(y(x),x);
  yx0:=y(0)=0,D(y)(0)=1;
  # Определение диф.уравнения и нач.усл.
```

$$de := m\,y'' - k\,y'$$

$$yx0 := y(0) = 0, D(y)(0) = 1$$

```
> dsolve({de,yx0},y(x));
```

$$y = -\frac{m}{k} + \frac{m\, e^{\left(\frac{k\,x}{m}\right)}}{k}$$

```
> de2:=m*diff(y(x),x$2)-k*diff(y(x),x);
  yx2:=y(0)=0,diff(y(b),b)=1;
  yy:=dsolve({de2,yx2},y(x));
```

$$de2 := m\,y'' - k\,y'$$

$$yx2 := y(0) = 0, y_b = 1$$

$$yy := y = -\frac{m}{k\, e^{\left(\frac{k\,b}{m}\right)}} + \frac{m\, e^{\left(\frac{k\,x}{m}\right)}}{k\, e^{\left(\frac{k\,b}{m}\right)}}$$

```
> subs(y(x)=yy,de2):
  simplify(%);
```

$$m\,y'' - k\,y' = 0$$

Граничные условия при решении задач указываются так же как и начальные.

```
> de3:=m*diff(y(x),x$2)-k*diff(y(x),x);
  yx3:=y(0)=0,y(1)=1;
```

$$de3 := m\,y'' - k\,y'$$

$$yx3 := y(0) = 0, y(1) = 1$$

```
> dsolve({de3,yx3},y(x));
```

$$y = -\frac{1}{-1+e^{\left(\frac{k}{m}\right)}} + \frac{e^{\left(\frac{k\,x}{m}\right)}}{-1+e^{\left(\frac{k}{m}\right)}}$$

```
> sys:=diff(y(x),x)=2*z(x)-y(x)-x,
  diff(z(x),x)=y(x);

  funcs:={y(x),z(x)};
  InC:=y(0)=0,z(0)=1;
```

$$sys := y' = 2\,z(x) - y - x,\ z' = y$$

$$funcs := \{y, z(x)\}$$

$$InC := y(0) = 0, z(0) = 1$$

```
> F:=dsolve({sys,InC},funcs);
```

$$F := \left\{y = \frac{1}{2}+\frac{1}{3}e^{x}-\frac{5}{6}e^{(-2\,x)},\ z(x) = \frac{1}{4}+\frac{1}{2}x+\frac{1}{3}e^{x}+\frac{5}{12}e^{(-2\,x)}\right\}$$

```
> Y:=subs(F,y(x));Z:=subs(F,z(x));
```

$$Y := \frac{1}{2}+\frac{1}{3}e^{x}-\frac{5}{6}e^{(-2\,x)}$$

$$Z := \frac{1}{4}+\frac{1}{2}x+\frac{1}{3}e^{x}+\frac{5}{12}e^{(-2\,x)}$$

При решении систем дифференциальных уравнений сами уравнения и начальные (граничные) условия указываются в фигурных скобках. Множество искомых функций также указывается в фигурных скобках.

Обратите внимание на последние строки примера. Здесь для выделения отдельных функций решения системы используется функция подстановки **subs**. Далее можно построить графики решений, используя функцию **plot**.

При использовании опции **laplase** для решения системы мы получим тот же ответ:

При использовании опции **series** получаемое в виде ряда решение отличается от явного решения и решения с применением преобразования Лапласа.

```
> Order:=8:
  FS:=dsolve({sys,InC},funcs,series);
```

$$FS := \{z(x) = 1 + x^2 - \frac{1}{2}x^3 + \frac{7}{24}x^4 - \frac{13}{120}x^5 + \frac{3}{80}x^6 - \frac{53}{5040}x^7 + O(x^8),$$
$$y = 2\,x - \frac{3}{2}x^2 + \frac{7}{6}x^3 - \frac{13}{24}x^4 + \frac{9}{40}x^5 - \frac{53}{720}x^6 + \frac{107}{5040}x^7 + O(x^8)\}$$

```
>
> FF:=dsolve({sys,InC},funcs,laplace);
```

$$FF := \{y(x) = -\frac{5}{6}e^{(-2\,x)} + \frac{1}{3}e^{x} + \frac{1}{2},\ z(x) = \frac{1}{3}e^{x} + \frac{5}{12}e^{(-2\,x)} + \frac{1}{2}x + \frac{1}{4}\}$$

```
> F=FF:evalb(%);
```

$$true$$

```
> evalb(F=FS);
```

$$false$$

Для проверки найденного решения может быть использована функция
odetest(sol,ODE),
где в **sol** -найденное решение , **ODE** - исходное дифференциальное уравнение

Для классификации дифференциальных уравнений в Maple V существует функция **odeadvisor**:

odeadvisor(ODE)
odeadvisor(ODE, y(x), [type1, type2, ...], help)

```
> with(DEtools):
> ODE :=
  x*diff(y(x),x)+a*y(x)^2-y(x)+s*x^2;
```

$$ODE := x\left(\frac{\partial}{\partial x}y(x)\right) + a\,y(x)^2 - y(x) + s\,x^2$$

```
> odeadvisor(ODE);
```

$$[[\_homogeneous, class\ D], \_rational, \_Riccati]$$

ODE- обыкновенное дифференциальное уравнение
y(x)- искомая функция
type1, type2,... - опция, подмножество типов классификации
help - опция, указывает надо ли выводить справку по поводу методов решения данного вида диф. уравнений (выводится в отдельных окнах).

6.2 Численное решение дифференциальных уравнений.

Для решения дифференциальных урвнений в численном виде используется функция **dsolve** с опцией **numeric** или **type=numeric**. При этом решение возвращается в виде специальной процедуры, по умолчанию реализующий широко известный метод решения дифференциальных уравнений Рунге-Кутта-Фельберга порядка 4 и 5 (в зависимости от условий адаптации решения к скорости его изменения). Эта процедура называется **rkf45** и символически выводится (без тела) при попытке решения заданной системы дифференциальных уравнений.
В список параметров функции **dsolve** можно явным образом включить указание на метод решения, например, опция **method=dverk78** задает решение методом Рунге-Кутта порядка 7 или 8.

```
  sys1:=diff(y(x),x)=2*z(x)*sin(x)-y(x)-x,
  diff(z(x),x)=y(x);

  funcs:={y(x),z(x)};
  InC:=y(0)=0,z(0)=1;
```

$$sys1 := \frac{\partial}{\partial x}y(x) = 2\,z(x)\sin(x) - y(x) - x,\ \frac{\partial}{\partial x}z(x) = y(x)$$

$$funcs := \{y(x), z(x)\}$$

$$InC := y(0) = 0, z(0) = 1$$

```
> FNUM:=dsolve({sys1,InC},funcs,numeric,
  output=listprocedure);
```

$$FNUM := [x = (\mathbf{proc}(x)\ \ldots\ \mathbf{end\ proc}),\ y(x) = (\mathbf{proc}(x)\ \ldots\ \mathbf{end\ proc}),\ z(x) = (\mathbf{proc}(x)\ \ldots\ \mathbf{end\ proc})]$$

```
> Y1:=subs(FNUM,y(x));Z1:=subs(FNUM,z(x));
```

$$Y1 := \mathbf{proc}(x)\ \ldots\ \mathbf{end\ proc}$$

$$Z1 := \mathbf{proc}(x)\ \ldots\ \mathbf{end\ proc}$$

```
> plot({Y1,Z1},0..6,color=[blue,green],
  thickness=2);
```

Для преобразования листа выходных данных в векторы решения используется функция **subs**.

В ряде случаев иметь явное решение дифференциальных уравнений нецелесообразно. Для неявного их представления в Maple V введена специальная структура **DESol**:

DESol(expr,vars)
expr - исходная система уравнений или одно уравнение, **vars** - заданный список переменных (или одна переменная).

Структура **DESol** образует некоторый объект, напоминающий **RootOf.** С этим объектом можно обращаться как с функцией: интегрировать, дифференцировать, получать разложение в ряд, использовать в вычислениях.

```
> dde:=DESol(D(y)-y,y,{y(0)=1});
```

$$dde := \mathrm{DESol}(\{D(y)-y\},\{y\},\{y(0)=1\})$$

```
> series(dde(x),x,8);
```

$$y(0)+y(0)\,x+\frac{1}{2}y(0)\,x^2+\frac{1}{6}y(0)\,x^3+\frac{1}{24}y(0)\,x^4+\frac{1}{120}y(0)\,x^5+\frac{1}{720}y(0)\,x^6+\frac{1}{5040}y(0)\,x^7+O(x^8)$$

```
> D(dde)-dde;
```

$$0$$

6.3 Инструментальный пакет решения дифференциальных уравнений DEtools.

Решение дифференциальных увнений самых различных типов - одно из достоинств системы Maple V. Пакет **DEtools** предоставляет ряд полезных функций для аналитического и численного решения дифференциальных уравнений и систем с такими уравнениями. Для использования функций этого пакета он должен быть загружен командой:

> with(DEtools): Пакет (в реализации R6) состоит из нескольких частей. Мы познакомимся с некоторыми из них.

1. Функции для преобразований дифференциальных уравнений

DEnormal	возвращает нормализованную форму диф. уравнений
convertAlg	возвращает лист коэффициентоа для диф. уравнений
Dchangevar	изменяет переменные в диф. уравнениях
convertsys	преобразует систему диф. уравнений в систему диф уравнений 1-го порядка
autonomous	тестирует диф. уравнения на автономность
reduceOrder	понижает порядок диф. уравнений
regularsp	вычисляет регулярные особые точки для диф. уравнений 2-го порядка
varparam	находит общее решение диф. уравнений методом вариации произвольных постоянных

DEnormal(des,ivar,dvar) - возвращает нормализованную форму дифференциальных уравнений, т.е. представляет уравнение в виде, когда группируются коэффициенты при неизвестной функции и ее производных.
des - обыкновенное диф. уравнение
ivar - независимая переменная
dvar - зависимая переменная

```
> with(DEtools):
  DE := x^3*y(x) + x^2*(x-1)*D(y)(x) +
  x^3/(x-1)*(D@@2)(y)(x) = x*sin(x);
```

$$DE := x^3\,y(x) + x^2\,(x-1)\,D(y)(x) + \frac{x^3\,(D^{(2)})(y)(x)}{x-1} = x\,\sin(x)$$

```
> DEnormal(DE,x,y(x));
```

$$(x^2 - 2x + 1)\left(\frac{\partial}{\partial x}y(x)\right) + y(x)\,x^2 - y(x)\,x + x\left(\frac{\partial^2}{\partial x^2}y(x)\right) = \sin(x) - \frac{\sin(x)}{x}$$

Функция **convertAlg** возвращает лист коэффициентов дифференциального уравнения (системы):
convertAlg(des,dvar)
des - диф. уравнение
dvar - зависимая переменная

```
  A := diff(y(x),x)*sin(x)-x*cos(x)*diff(y(x),x,x)-tan(x)*y(x)=5;
```

$$A := \left(\frac{\partial}{\partial x}y(x)\right)\sin(x) - x\,\cos(x)\left(\frac{\partial^2}{\partial x^2}y(x)\right) - \tan(x)\,y(x) = 5$$

```
> convertAlg(A,y(x));
```

$$[[-\tan(x),\ \sin(x),\ -\cos(x)\,x],\ 5]$$

Для преобразования диф. уравнения порядка m или системы диф. уравнений к системе диф. уравнений 1 -го порядка служит функция
convertsys(deqns, inits, vars, ivar, yvec, ypvec)

deqns - обыкновенное диф. уравнение, или множество (список) диф.уравнений
inits - множество или список начальных условий **vars** - зависимая переменная (переменные - для системы)
ivar - независимая переменная
yvec - имя для вектора решений системы диф.уравнений 1-го порядка.
ypvec - имя для производных вектора решений системы диф. уравнений 1-го порядка

```
> deq2 := (D@@3)(y)(x) = y(x)*x;
  init2 := y(0) = 3, D(y)(0) = 2, (D@@2)(y)(0) = 1;
  convertsys(deq2, [init2], y(x), x);
```

$$deq2 := (D^{(3)})(y)(x) = y(x)\,x$$

$$init2 := y(0) = 3, D(y)(0) = 2, (D^{(2)})(y)(0) = 1$$

$$\left[[YP_1 = Y_2, YP_2 = Y_3, YP_3 = Y_1\,x], \left[Y_1 = y(x), Y_2 = \frac{\partial}{\partial x}y(x), Y_3 = \frac{\partial^2}{\partial x^2}y(x)\right], 0, [3, 2, 1]\right]$$

```
> deq3 := diff(y(t), t$2) = 100*(exp(-10*t)+exp(10*t)):
  convertsys({deq3}, [], y(t), t, V);
```

$$\left[[YP_1 = V_2, YP_2 = 100\,e^{(-10\,t)} + 100\,e^{(10\,t)}], \left[V_1 = y(t), V_2 = \frac{\partial}{\partial t}y(t)\right], undefined, [\,]\right]$$

```
>  convertsys({deq3}, D(y)(3)=1, y(t), t, V);
```

$$\left[[YP_1 = V_2, YP_2 = 100\,e^{(-10\,t)} + 100\,e^{(10\,t)}], \left[V_1 = y(t), V_2 = \frac{\partial}{\partial t}y(t)\right], 3, [y(3), 1]\right]$$

Если начальные условия не определены или определены не полностью, то в результате работы функция **convertsys** в качестве второго аргумента - начальной точки возвращает **undefinit** и пустое множество**[]** начальных данных или недостающее начальное условие в общем виде (f(a), где a - начальная точка).

Функция **reduceOrder** обеспечивает понижение порядка дифференциального уравнения (или системы уравнений, представленных списком или множеством)
reduceOrder(des,dvar,partsol, solutionForm)

des - обыкновенное диф уравнение (список или множество уравнений)
dvar - зависимая переменная
partsol - частное решение (или их список)
solutionForm - параметр, указывающий, что решение должно быть найдено в явном виде.

```
de := diff(y(x),x$3) - 6*diff(y(x),x$2)
+ 11*diff(y(x),x) - 6*y(x);
```

$$de := \left(\frac{\partial^3}{\partial x^3}y(x)\right) - 6\left(\frac{\partial^2}{\partial x^2}y(x)\right) + 11\left(\frac{\partial}{\partial x}y(x)\right) - 6\,y(x)$$

```
sol := exp(x);
```

$$sol := e^x$$

```
reduceOrder( de, y(x), sol);
```

$$\left(\frac{\partial^2}{\partial x^2}y(x)\right) - 3\left(\frac{\partial}{\partial x}y(x)\right) + 2\,y(x)$$

```
reduceOrder( de, y(x), sol, basis);
```

$$\left[e^x, e^{(2\,x)}, \frac{1}{2}e^{(3\,x)}\right]$$

Функция **varparam(sols,v,ivar)** - находит общее решение диф. уравнения (или системы диф. уравнений) **sols** методом вариации произвольных постоянных

sols - список решений однородного уравнения
v - правая часть (неоднородность)исходного уравнения
ivar - независимая переменная

```
> de := diff(Y(x),x,x) + Y(x):
  sols := dsolve( de, Y(x), output=basis);
```

$$sols := [\sin(x), \cos(x)]$$

```
> varparam( sols, x, x );
```

$$\_C_1 \sin(x) + \_C_2 \cos(x) + \sin(x)^2 x + \cos(x)^2 x$$

<

2. Функции для построения решений определенного вида диф. уравнений в замкнутой форме. Справку по этим функциям можно найти в разделе Mathematics → Differential Equation → DEtools → Solving Methods

3.Функции, используемые для визуализации решений диф. уравнений:

DEplot	строит графики решений диф. уравнений
DEplot3d	строит трехмерные графики для систем диф. уравнений
dfieldplot	строит график решения диф. уравнения в виде векторного поля
phaseportrait	строит график решения диф. уравнений в форме фазового портрета

Функция DEplot строит график решения диф. уравнения.

DEplot(deqns, vars, trange, inits, opt)
DEplot(deqns, vars, trange, yrange, xrange, opt)
DEplot(deqns, vars, trange, inits, xrange, yrange, opt)

deqns - список или множество диф.уравнений 1-го порядка или одно уравнение любого порядка
vars - зависимая переменная или список (множество) зависимых переменных
trange - область изменения независимой переменной
inits - начальные условия для кривой решения
xrange - область изменения зависимой переменной
eqns - опции в форме равенств: ключевое слово=значение

Команда **DEplot()** численно решает как одно дифференциальное уравнение произвольного порядка, так и нормальную систему обыкновенных дифференциальных уравнений 1-го порядка, причем в этом случае должна быть только ОДНА независимая переменная, т.е. переменная, от которой зависят искомые функции системы.

Параметр **trange** определяет диапазон изменения независимой переменной в виде **t=a..b** .

Начальные условия определяются параметром **inits**, который представляет собой список, элементами которого являются списки (список списков). Каждый такой элемент-список определяет интегральную кривую дифференциального уравнения или системы, которая отображается на графике. Количество элементов-списков параметра **inits** соответствует количеству интегральных кривых на графике. Граничные условия задаются также как и для команды **dsolve()**, через дифференциальный оператор **D**. Например, следующий список определяет начальные условия для двух интегральных кривых одного дифференциального уравнения третьего порядка с неизвестной функцией **z(x)**

>[[z(0)=1, D(z)(0)=2, (D$2)(z)(0)=0.5], [z(1)=1, D(z)(1)=2, (D$2)(z)(1)=0.5]]

Если задаются граничные условия для отображения только одной интегральной кривой, то они также должны быть заданы как список:

[[z(0)=1, D(z)(0)=2, (D$2)(z)(0)=0.5]]

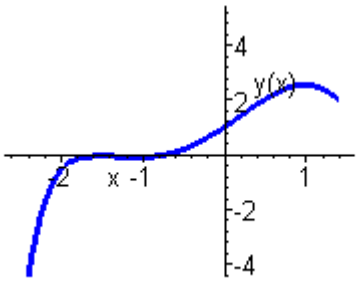
Параметры **xrange**, а их может быть столько, сколько неизвестных функций в системе, задают диапазон изменения неизвестных функций и используются для завершения процесса интегрирования. численное интегрирование осуществляется с заданным шагом, который по умолчанию равен abs(b-a)/20, где числа a и b задаются в параметре **trange**. Как только при очередном шаге значение какой-либо неизвестной функции выходит за пределы заданного в соответствующем параметре диапазона ее изменения, процесс интегрирования останавливается. Параметры **xrange** можно задавать в одной из двух форм:

x(t)=x1..x2 или **x=x1..x2**

В них **x** представляет имя неизвестной функции дифференциального уравнения или системы. Если неизвестных функций несколько, то перечисление областей интегрирования производится через запятую.

Необязательным параметром **eqns** задается ряд опций, определяющих общий вид графика решения: цвет линии интегральной кривой, шаг интегрирования, влияющий на гладкость отображения кривой, что и как откладывается по осям координат двумерного графика и т.д. Кроме некоторых специальных опций они в основном совпадают с опциями команды **plot()** пакета **plots** , а также с опциями, которые можно задавать при построении численного решения **type=numeric** командой **dsolve()** . Команда **plot()** описывается в Лекции 7, посвященной графике в Maple.

```
DEplot(cos(x)*diff(y(x),x$3)-diff(y(x),x$2)+
Pi*diff(y(x),x)=y(x)-x,y(x),
x=-2.5..1.4,
[[y(0)=1,D(y)(0)=2,(D@@2)(y)(0)=1]],
y=-4..5,stepsize=.05, linecolor=blue);
```



В примере ниже решается дифференциальное уравнение второго порядка и отображаются три его интегральные кривые, проходящие через одну точку, но с разными углами наклона касательной (в начальных условиях задаются разные значения первой производной в точке *x=0*)

Для системы трех дифференциальных уравнеий первого порядка *{dif1, dif2, dif3}* строится график зависимости функции решения *z(t)* от функции *x(t)* и график решения *x(t)* от *t*. Это достигается использованием разных значений опции **scene** . Кроме нее для построения необходимых графиков в этом примере также используются опции **lintcolor** и **stepsize** , за-

дающие, соответственно, цвет линии графика и шаг независимой переменной для получения точек графика.

Команда **DEplot** использует большой набор опций, наиболее употребительные из них представлены в таблице ниже.

Опции команды **DEplot**	
arrow	Задает размеры стрелок при отображении поля напрвлений для автономниых линейных систем второго порядка или дифференциальных уравнений второго порядка. Допустимые значения:SMALL (установлен по умолчанию), MEDIUM, LARGE, LINE, NONE . Значение NONE подавляет отображение поля направлений.
color	Задает цвет стрелок поля направлений.
dirgrid	Определяет количество точек сетки в горизонтальном и вертикальном направлениидля отображения векторов поля направлений. Задается в виде списка из двух целых чисел, первое из которых соответствует горизонтальному направлению, а второе вертикальному. Минимальное допустимое значение [2, 2] , максимальное [20, 20] , которое используется по умолчанию.
iterations	Определяет количество шагов интегрирования между отображаемыми точками решения, по умолчанию равно единице. Данная опция полезна, когда для увеличесния точости решения приходится уменьшать шаг интегрирования. Увеличение ее значения уменьшает объемы хранения информации, так как в памяти хранятся только отображаемые точки решения.
linecolor	Задает цвет выводимых линий решения. Если на одном графике отображаются несколько решений дифференциального уравнения или системы, то можно задать список значений цветов, которыми будут отображены соответствующие решения (см. предыдущий пример).
obsrange	Установка этой опции , равной булевому значению TRUE (значение по умолчанию), останавливает процесс интегрирования системы дифференциальных уравнений, как только значение какой-либо искомой функции выходит за диапазон ее изменения, определенный параметром **xrange** . Значение FALSE отключает этот режим.
scene	Определяет, что выводится на двумерном графике решения. Задается в виде двухэлементого списка искомых функций или функции и независимой переменной. Например, scene=[x(t),y(t)]означает, по горизонтальной оси графика отображается функция x(t) , а по вертикальной - y(t) . Это позволяет для системы второго порядка отобразить фазовый портрет. Для отображеия решения следует задать эту опцию, например, в виде scene=[t, x(t)].
stepsize	Задает шаг изменеия независимой переменной при численном интегрировании дифференциального уравнения. По умолчанию равен abs(b-a)/20,, где [a, b] диапазон изменения независимой переменной, заданный параметром trange.

Остальные опции соответствуют опциям команды **plot()** и **dsolve()** , страницы справки которых можно отобразить командами **?plot[options]** и **?dsolve[numeric]**.

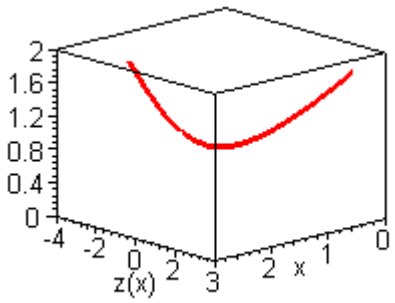
Если система дифференциальных уравнений или одно дифференциальное уравнение, задаваемые в команде **DEplot()** , являются автономными (правая часть системы или уравнения не зависят явным образом от независимой переменной), то задавать начальные значения не обязательно, однако обязательно задавать диапазон изменения искомых перемен-

ных. Для проверки автономности системы можно воспользоваться командой **autonomous()**.

В ряде случаев решение систем диф. уравнений удобно представлять в виде пространственных кривых - например, линий равного уровня или просто в виде кривых в пространстве. Для этого служит функция **DEplot3d**.

DEplot3d(deqns, vars, trange, inits, eqns)
DEplot3d(deqns, vars, trange, xrange, inits,opt).
Параметры имеют тот же смысл, что и для предыдущей функции, за исключением опций **arrows, dirgrid,color**, которые в ней не применяются. Дополнительные опции соответствуют опциям команды**plot3d()** из пакета **plots** и опциям команды **dsolve()** при выполнении команды численного интегрирования.

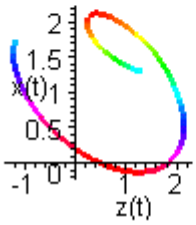
```
DEplot3d({diff(y(x),x)=y(x)-z(x),
diff(z(x),x)=z(x)-2*y(x)},
{y(x),z(x)},
x=0..3,
[[y(0)=1.638,z(0)=2.31]],
y=0..2,z=-4..4,
scene=[x,z(x),y(x)],
linecolour=red);
```



Для построения графика решения в виде фазового портрета служит функция **phaseportrait(deqns, vars, trange, inits, eqns)**

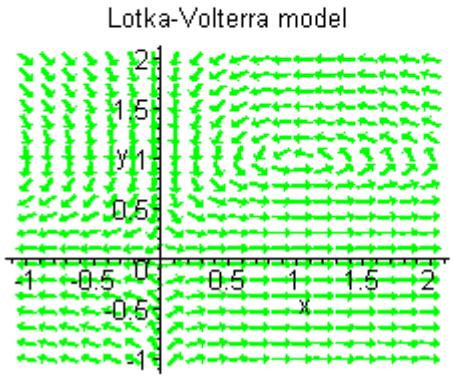
deqns - список или множество обыкновенных диф. уравнений 1-го порядка, или одно уравнение любого порядка
vars - зависимая переменная, или список (множество) зависимых переменных
trange - область изменения независимой переменной
inits - начальное условия (в виде списка)
eqns - опции в виде: ключевое слово=значение
(те же, что и для команды **DEplot()**).

```
> phaseportrait([D(x)(t)=y(t)-z(t),
  D(y)(t)=z(t)-x(t),
  (z)(t)=x(t)-y(t)*2],
  [x(t),y(t),z(t)],
  t=-2..2,[[x(0)=1,y(0)=0,z(0)=2]],
  stepsize=.05,scene=[z(t),x(t)],
  linecolour=sin(t*Pi/2),
  method=classical[foreuler]);
```



Для построения векторного поля (поля направлений) решений диф. уравнений используется функция **dfieldplot()**. Ее параметры те же, что и для функции **DEplot**, кроме опций, относящихся к построению решения (они не применяются).

```
> dfieldplot([diff(x(t),t)=x(t)*(1-y(t)),diff(
  y(t),t)=.3*y(t)*(x(t)-1)],
  [x(t),y(t)],t=-2..2,x=-1..2,y=-1..2,
  arrows=LARGE,title=`Lotka-Volterra model`,
  color=green);
```



6.4 Пакет PDEtools решения диф. уравнений в частных производных.

Это относительно новый пакет (в реализации R4 - отсутствует), в реализации R6 представленный 10-ю функциями, а в реализации R6 - пятнадцатью. В таблице представлены эти команды, а также их краткое описание

Команды пакета **PDEtools** .	
build	Решает обыкновенные дифференциальные уравнения, получаемые при использовании метода разделения переменных в команде **pdsolve()**, и возвращаетявный вид решения уравнения в частных производных. Используется также в команде **pdsolve()** в качестве опции.
dcoeffs	Приводит подобные коэффициенты в полиномиальном уравнении в частных производных относительно неизвестной и степеней ее производных и выдает

	список полученных коэффициентов.
dsubs	Входными параметрами команды являются список уравнений, в левых частях которых стоят производные от неизвестных функций, входящих в выражение, представленное самым последим параметром. Команда последовательно подставляет в это выражение соответствующие производные, причем пытается продолжить подстановку до тех пор, пока из выражения не будут исключены все производные, определяемые в передаваемых команде уравнениях.
casesplit	Разбивает систему уравнений и/или неравенств на последовательность систем уравнений и неравенств таких, что объединение их не особых решений равняется множеству решений исходной системы. Кроме того, каждая из полученных систем свободна от дифференциальной и алгебраической избыточности и для них автоматически удовлетворены условия интегрируемости.
charstrip	Вычисляет уравнение характеристик для заданного дифференциального уравнения в частных производных первого порядка, т.е. строится и решается соответствующая ему система обыкновенных дифференциальных уравнений в симметрической форме.
dchange	Осуществляет замену переменных в любых алгебраических объектах (уравнения в частных производных, кратные интегралы, интегродифференциальные уравнения и т.д.), а также в процедурах. Эта команда полезна для изменения уравнения в частных производных от формы, трудно решаемой, к форме, для которой можно найти решение.
difforder	Определяет общий или относительно некоторой переменной порядок частной производной в уравнении или выражении.
mapde	Преобразует уравнение в частных производных к уравнению в частных производных другого вида, которое, возможно, будет решаться командой **pdsolve()**.
PDEplot	Строит график решения уравнения в частных производных первого порядка, линейного или нелинейного, при заданных начальных условиях.
pdesolve	Ищет решение дифференциального уравнения в частных производных.
pdetest	Проверяет, является ли некоторое выражение решением уравнения в частных производных, возвращая 0 в случае положительности проверки и выражение невязки в противном случае.
separability	Определяет, при каких условиях для заданного уравнения в частных производных возможно получение полного решения при разделении переменных в виде суммы или произведения.
splitsys	Разбивает систему алгебраических или дифференциальных уравнений на независимые между собой системы уравнений.
splitstrip	Вычисляет характеристики уравнения в частных производных, но возвращает их, если это возможно, разбитыми на подмножества характеристик независимых между собой систем уравнений, полученных в результате выполнения команды **splitsys** .
declare	Преобразует вид дифференциальных уравнений, заменяя производные штрихами или индексами, указывающими, по какой независимой переменной берется производная; позволяет у функций в уравнении опустить независимые переменные (вместо *y(x)* вобласти вывода получить *y*).
undeclare	Отменяет все или некоторые из преобразований, проведенных функцией

	declare .

Ознакомимся с работой некоторых команд.

Функция **pdsolve** позволяет находить в аналитическом виде решение дифференциальных уравнений в частных производных. Эта функция находится в ядре системы и может вызываться без подключения пакета **PDEtools**.

```
d1:=diff(f(x,y),x,x)+5*diff(f(x,y),x,y)=3;
pdsolve( d1, f(x,y) );
```

$$d1 := f_{x,x} + 5 f_{x,y} = 3$$

$$f(x,y) = \_F1(y) + \_F2(y - 5x) + \frac{3}{2}x^2$$

```
d2:=3*diff(g(x,y),x)+7*diff(g(x,y),x,y)=x*y;
pdsolve( d2, g(x,y) );
```

$$d2 := 3 g_x + 7 g_{x,y} = x y$$

$$g(x,y) = \_F1(y) + e^{\left(-\frac{3}{7}y\right)} \_F2(x) + \frac{1}{18}x^2(3y - 7)$$

```
d3:=-a^2*diff(U(x,t),x,x)+diff(U(x,t),t,t)=0;
pdsolve(d3,U(x,t));
```

$$d3 := -a^2 U_{x,x} + U_{t,t} = 0$$

$$U(x,t) = \_F1(a t + x) + \_F2(a t - x)$$

```
d4:=y*diff(U(x,y),x)+x*diff(U(x,y),y)=0;
```

$$d4 := y U_x + x U_y = 0$$

```
pdsolve(d4, U(x,y) );
```

$$U(x,y) = \_F1(-x^2 + y^2)$$

Синтаксис вызова функции **pdesolve(deqns, vars)**

deqns - дифференциальное уравнение в частных производных
vars - переменная, которая должна быть найдена (неизвестная функция)

Если функция **pdsolve** не может выписать решение в явном виде, то она выдает его в виде некоторой структуры **ODESolStruc** (характерным для этой структуры является использование слова "**&where**").

Обычно это бывает при попытке решить уравнение методом разделения переменных, при этом после слова "&where" записываются выражения (уравнения) для определения функций, входящих в ответ.

Построить решение в этом случае можно с помощью функции **build**, либо можно использовать эту функцию в виде опции (третьего параметра) при вызове команды **pdsolve**

```
> PDE :=
  x^2*diff(f(x,y),x)+y*diff(f(x,y),y)+
  (diff(f(x,y),y)^2+diff(f(x,y),x))=0;
```

$$PDE := x^2 f_x + y f_y + f_y^2 + f_x = 0$$

```
> ans := pdsolve(PDE,f(x,y));
```

$$ans := (f(x,y) = \_F1(x) + \_F2(y)) \ \&where \left[\left\{\_F1_x = \frac{\_c_1}{x^2+1}, \_F2_y{}^2 = -\_c_1 - y \_F2_y\right\}\right]$$

```
> ans1:=build(ans);
```

$$ans1 := f(x,y) = \_c_1 \arctan(x) + \_C1 - \frac{1}{4}y^2 - \frac{1}{4}y\sqrt{-4\_c_1 + y^2} + \_c_1 \ln\left(y + \sqrt{-4\_c_1 + y^2}\right) + \_C2$$

```
> pdetest(ans,PDE);
```

$$0$$

Обратите внимание на использование функции **pdetest** для проверки полученного решения

Функция **dcoeffs** приводит подобные коэффициенты в полиномиальном уравнении в частных производных относительно неизвестной и степеней ее производных и выдает список полученных коэффициентов.

```
> expr:=a*x(t)*diff(y(t),t)^2+x(t)*diff(y(t),
  t)+b*y(t)*diff(y(t),t)+
  c*y(t)^2*diff(x(t),t);
```

$$expr := a\,x(f)\,y'^{2} + x(f)\,y' + b\,y\,y' + c\,y^{2}\,x'$$

```
> dcoeffs(expr,y(t));
```

$$a\,x(f),\ b,\ x(f),\ c\,x'$$

```
> dcoeffs(expr,x(t));
```

$$c\,y^{2},\ a\,y'^{2} + y',\ b\,y\,y'$$

Синтаксис команды:
dcoeffs(expr, y(x))

expr - дифференциальное выражение
y(x) - функция относительно которой вычисляются коэффициенты из **expr**

Команда **casesplit** - разбивает систему уравнений и/или неравенств на последовательность систем уравнений и неравенств таких, что объединение их не особых решений равняется множеству решений исходной системы. Кроме того, каждая из полученных систем свободна от дифференциальной и алгебраической избыточности и для них автоматически удовлетворены условия интегрируемости.

```
> ode[1] := diff(y(t),t,t)^2 +
  2*diff(y(t),t,t)*y(t)^3*diff(y(t),t)
  - 4*y(t)^2*diff(y(t),t)^3;
```

$$ode_1 := y''^{2} + 2\,y''\,y^{3}\,y' - 4\,y^{2}\,y'^{3}$$

```
> casesplit(ode[1]);
```

$$[y''^{2} = -2\,y''\,y^{3}\,y' + 4\,y^{2}\,y'^{3}]\ \&\text{where}\ [y'' + y^{3}\,y' \neq 0],$$
$$\left[y'^{2} = -\frac{1}{4}\,y'\,y^{4}\right]\ \&\text{where}\ [y^{4} + 8\,y' \neq 0],\ [y = 0]\ \&\text{where}\ [\,]$$

```
> pde[1] :=
  diff(f(x,y,z),x,y)*diff(f(x,y,z),y)=h(x)*g(y,z);
> casesplit(pde[1], f);
```

$$pde_1 := f_{x,y}\,f_y = h(x)\,g(y,z)$$

$$\left[f_{x,y} = \frac{h(x)\,g(y,z)}{f_y}\right]\ \&\text{where}\ [f_y \neq 0],\ [f_y = 0,\ g(y,z) = 0]\ \&\text{where}\ [\,],$$
$$[f_y = 0,\ h(x) = 0]\ \&\text{where}\ [\,]$$

Синтаксис команды
casesplit(sys)

sys - система уравнений (в виде списка или множества) или одно уравнение.

С помощью команды **op()** можно выделить любой операнд полученного ответа и получить решение для каждой из полученных систем.

Команда **charstrip** вычисляет уравнение характеристик для заданного дифференциального уравнения в частных производных первого порядка, т.е. строится соответствующая ему система обыкновенных дифференциальных уравнений в симметрической форме.

Синтаксис команды:
charstrip(PDE,f)

PDE- уравнение в частных производных
f - неизвестная функция

```
> PDE := x*diff(f(x,y,z),z)-f(x,y,z)+
  y^2*diff(f(x,y,z),y)=0;
```

$$PDE := x\,f_z - f(x,y,z) + y^{2}\,f_y = 0$$

```
> sys0 := charstrip(PDE,f(x,y,z));
```

$$sys0 := \{x\_s = 0,\ f\_s = f(\_s),\ y\_s = y(\_s)^{2},\ z\_s = x(\_s)\}$$

Система может быть решена с помощью команды **dsolve**

```
> dsolve(sys0, {f(_s),x(_s),y(_s),z(_s)}, explicit);
```

$$\{z(\_s) = \_C2\ \_s + \_C1,\ y(\_s) = -\frac{1}{\_s - \_C3},\ x(\_s) = \_C2,\ f(\_s) = \_C4\,e^{-5}\}$$

Команда **dchange** осуществляет замену переменных в любых алгебраических объектах (уравнения в частных производных, кратные интегралы, интегро-дифференциальные

уравнения и т.д.), а также в процедурах. Эта команда полезна для изменения уравнения в частных производных от формы, трудно решаемой, к форме, для которой можно найти решение.

Синтаксис команды:
dchange(tr,expr)
dchange(tr,expr,newvars,itr,known=...,unknown=...,params=...,simp_proc)

tr - комплект уравнений, левая часть которых содержит старые переменные, а правая - новые.
expr - алгебраическое выражении или процедура (программа)
newvars (требуется, если не очевидно) список новых переменных
itr - (доп.) список уравнений обратного преобразования
known=...- (доп.) уравнение, имеющее с левой стороны строку 'known', а справа - имя функции или список (имен) функций, которые должны преобразовываться как известные функции.
unknown=... - (доп.) уравнение, имеющее, с левой стороны строку 'unknown', а справа - имя функции или список (имен) функций, которые должны преобразовываться как НЕИЗВЕСТНЫЕ
params=... - (доп.) уравнение, имеющее с левой стороны строку 'params', а справа - множество имен параметров уравнения, которые должны считаться заданными параметрами при преобразовании, т.е. не являются переменными.
simp_proc- (доп.) процедура упрощения, которая должна применяться к результату.

```
> PDE := diff(f(x,y),x) + g(x,y) + diff(f(x,y),y)=0;
```

$$PDE := f_x + g(x,y) + f_y = 0$$

```
> tr := {x = r + s, y = r - s}:
  dchange(tr,PDE);
```

$$f_r + g(r,s) = 0$$

g объявлена известной ("known") функцией:

```
> dchange(tr,PDE,known=g);
```

$$f_r + g(r+s, r-s) = 0$$

```
> L := f -> x*diff(f,y) - y*diff(f,x);
```

$$L := f \to x\left(\frac{\partial}{\partial y} f\right) - y\left(\frac{\partial}{\partial x} f\right)$$

```
> tr := {x = s*r^(1/2), y = s*(1-r)^(1/2)};
```

$$tr := \{x = s\sqrt{r}, y = s\sqrt{1-r}\}$$

```
> L1 = dchange(tr,L,simplify);
```

$$L1 = \left(f \to -2\sqrt{r}\sqrt{1-r}\left(\frac{\partial}{\partial r} f\right)\right)$$

Команда **difforder** определяет порядок дифференциального уравнения.

Синтаксис команды
difforder(a)
difforder(a,x)

a - выражение
x - переменная, относительно которой вычисляется порядок производных

```
eq := h(t)*diff(f(x,y,z),x$2,z$3)+
h(x,y,z)*diff(g(x,y,z),y$2);
difforder(eq,x), difforder(eq,y),
difforder(eq,z), difforder(eq,t);
```

$$eq := h f_{x,x,z,z,z} + h(x,y,z)\, g_{y,y}$$

$$2, 2, 3, 0$$

```
eq := diff(f(u,v),u$2,v$3);
```

$$eq := f_{u,u,v,v,v}$$

```
difforder(eq), difforder(eq,u),
difforder(eq,v);
```

$$5, 2, 3$$

Команда **mapde** преобразует уравнение в частных производных к уравнению в частных производных другого вида, которое, возможно, будет решаться командой **pdsolve( ).**

Синтаксис команды
mapde(PDE,into)
mapde(PDE,into,f) , где

PDE - уравнение в частных производных
f - (доп.) имя неопределенной функции
into - строка, один из указанных ниже параметров:

```
> PDE :=
  diff(f(x,y,z),x)^3=f(x,y,z)*diff(f(x,y,z),y,
  y)*diff(f(x,y,z),z);
```

$$PDE := f_x{}^3 = f(x,y,z)\, f_{y,y}\, f_z$$

```
> pdsolve(PDE):   # No solutions found.
> PDE3 := mapde(PDE,noF);
```

$$PDE3 := (\_F1_x{}^3 - \_F1_z\, \_F1_{y,y} - \_F1_z\, \_F1_y{}^2 = 0)\ \&where\ \{\_F1(x,y,z) = \ln(f(x,y,z))\}$$

```
> ans := pdsolve(op(1,PDE3));
```

$$ans := (\_F1(x,y,z) = \_F2(x) + \_F3(y) + \_F4(z))\ \&where\ \left[\left\{\_F2_x = \_c_1,\ \_F4_z = \_c_3,\ \_F3_{y,y} = -\_F3_y{}^2 + \frac{\_c_1{}^3}{\_c_3}\right\}\right]$$

```
> build(ans);
```

$$\_F1(x,y,z) = \_c_1\, x + \_C1 - \ln(2) + \ln\left(-\mathbf{e}^{\left(2\frac{\_c_1{}^{\left(\frac{3}{2}\right)}\, y}{\sqrt{\_c_3}}\right)}\, \_C2 + \_C3\right) + \frac{1}{2}\ln(\_c_3) - \frac{3}{2}\ln(\_c_1) - \frac{\_c_1{}^{\left(\frac{3}{2}\right)}\, y}{\sqrt{\_c_3}} + \_c_3\, z + \_C4$$

noF- работает, когда полученное УЧП- однородная функция от неопределенной функции и производных. В этих случаях **mapde**представляет неопределенную функцию, равную экспоненте от другой (вспомогательной) функции и преобразует уравнение
ccoeff - УЧП с постоянными коэффициентами для уравнений с производными высокого порядка
canom - каноническая форма с только одной смешанной производной
canop каноническая форма с только повторными (не смешанными) производными

Решение исходного уравнения (см. рисунок сверху) можно получить, возведя экспоненту в степень, равную полученной функции **_F1(x,y,z).**

Команда **separability** определяет, при каких условиях для заданного уравнения в частных производных возможно получение полного решения при разделении переменных в виде суммы (по умолчанию) или произведения (указывается явно в обратных кавычках при обращении к команде). Результатом работы функции в случае положительного ответа (возможности разделения переменных) является нуль. Если разделение переменных невозможно - повторяется само уравнение.

Синтаксис команды
separability(PDE,F(x,y,...))
separability(PDE,F(x,y,...),`*`)

PDE, дифференциальное уравнение в частных производных
F(x,y,...)- функция, которую требуется представить разделенной на функции, каждая из которых зависит только от одной переменной
`*` - дополнительный параметр, указывающий на то, что искомая функция должна быть представлена в виде произведения некоторых функций

```
> pde3 := diff(u(x,y),x,x) + diff(u(x,y),y,y) + w^2*u(x,y)=0;
```

$$pde3 := u_{x,x} + u_{y,y} + w^2 u(x,y) = 0$$

```
> separability(pde3,u(x,y));
  # тест на разделение переменных с помощью суммы
```

$$0$$

```
> separability(pde3,u(x,y),`*`);
  # тест на разделение переменных с помощью произведения
```

$$0$$

```
> pde2 := (y-z)*diff(u(x,y,z),x,x) + (z-x)*diff(u(x,y,z),y,y) +
  (x-y)*diff(u(x,y,z),z,z)=E;
```

$$pde2 := (y-z)\,u_{x,x} + (z-x)\,u_{y,y} + (x-y)\,u_{z,z} = E$$

```
> separability(pde2,u(x,y,z));
  # невозможно разделить переменные
```

$$u_{x,x}\,y - u_{z,z}\,y - u_{x,x}\,z + u_{y,y}\,z - u_{y,y}\,x + u_{z,z}\,x$$

Команда **splitsys** - Разбивает систему алгебраических или дифференциальных уравнений на независимые между собой системы уравнений.

Синтаксис команды
splitsys(sys,funcs)

sys - множество алгебраических или диф. уравнений
funcs - множество неизвестных функций

```
PDE_sys := {diff(f(x,y),x) = g(x,y)+f(x,y),
diff(g(x,y),x)^2 = g(x,y)-f(x,y),
diff(h(x,y,z),x) = z*k(x,y)};
```

$$PDE\_sys := \{g_x{}^2 = g(x,y) - f(x,y),\ h_x = z\,k(x,y),\ f_x = g(x,y) + f(x,y)\}$$

```
splitsys(PDE_sys,{f(x,y),g(x,y),h(x,y,z),k(x,y)});
```

$$\{h_x = z\,k(x,y)\},\ \{g_x{}^2 = g(x,y) - f(x,y),\ f_x = g(x,y) + f(x,y)\}$$

Команда **dsubs** последовательно подставляет производные, указанные в качестве первого параметра , в **expr** , причем пытается продолжить подстановку до тех пор, пока из **expr** не будут исключены все производные, определяемые в передаваемых команде уравнениях.

Синтаксис команды
dsubs(deriv=a, expr)
dsubs(deriv1=a,`...`, expr)

```
> eq2 := diff(f(x,y,z),x,z) = diff(f(x,y,z),y),
  diff(f(x,y,z),y,y)=diff(f(x,y,z),x,x);
```

$$eq2 := f_{x,z} = f_y, f_{y,y} = f_{x,x}$$

```
> dsubs(eq2, diff(f(x,y,z),x,y,z));
```

$$f_{x,x}$$

```
> subs(eq2, diff(f(x,y,z),x,y,z));
```

$$f_{x,y,z}$$

где параметры **a,expr** - выражения,
deriv - производная для замены в **expr**
deriv1,`...`,dsn - уравнения или последовательность

Обратите внимание на различие в работе функции **dsubs()** дифференциальной подстановки и функции **subs()** обычной подстановки.

Для более компактной записи уравнений и результатов их преобразований в поле вывода может быть использована команда **declare()**. Она позволяет заменить символы производных по определенной переменной на штрих для обыкновенных производных (функции одной переменной) или на подстрочные индексы, указывающие, по каким переменным берется производная от функции многих переменных.

Синтаксис команды:
declare(fncs,prime=..)

fnct - список функций, для которых используется новое обозначение производной
prime=t независимая переменная (после ключевого слова prime), производные по которой заменяются на штрих

```
> declare( [ y(t)], prime=t);
```

y(t), will now be displayed as, y
derivatives with respect to:, t,
of functions of one variable will now be displayed with '

```
> ON;ode[1] := diff(y(t),t,t)^2 -
  4*y(t)^2*diff(y(t),t)^3;
```

$$ode_1 := y''^2 - 4y^2 y'^3$$

```
> OFF;ode[1];
```

$$\left(\frac{\partial^2}{\partial t^2} y(t)\right)^2 - 4\,y(t)^2 \left(\frac{\partial}{\partial t} y(t)\right)^3$$

Для "выключения" режима укороченного воспроизведения выражения с производными используется команда **OFF** , для включения вновь - команда **ON** . Если необходимо отменить укороченную запись для всех функций, для которых она была определена, используется команда **undeclare(all)** .

Команда **PDEplot()** рассматривается в следующем разделе .

Команда **PDEplot** строит график решения дифференциального уравнения в частных производных ПЕРВОГО порядка (как линейного, так и нелинейного; с постоянными и переменными коэффициентами).

Синтаксис команды **PDEplot(PDE, inits, srange, options)**

PDE - диф. уравнение 1-го порядка в частных производных n независимых переменных
inits - начальные данные; список из n+1 выражений или равенств, представленный в параметрической форме от n-1 параметра
srange - область (области) изменения параметров в начальных данных
options - опции.

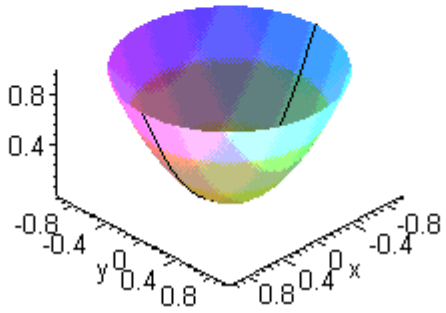
```
> eqq:=y*diff(z(x,y),x)-x*diff(z(x,y),y)=0;
```

$$eqq := y z_x - x z_y = 0$$

```
> inits:=[s,0,s^2];#начальные условия: при x=0
  z=x^2
```

$$inits := [s, 0, s^2]$$

```
> PDEplot(eqq,inits,s=-1..1);
```



Опции команды **PDEplot()**	
iterations	Определяет количество шагов интегрирования между отображаемыми точками решения (по умолчанию - единица). Данная опция полезна, когда для увеличения точности решения приходится уменьшать шаг интегрирования. Увеличение ее значения уменьшает объемы хранимой информации, т.к. в памяти хранятся только отображаемые точки решения.
ctepsize	Задает расстояние между вычисляемыми точками вдоль кождой из характеристик, которое может принимать вещественные значения. Значение по умолчанию, оно же и максимальное, равно 0.25.
numsteps	Определяет количество отображаемых точек вдоль характеристической линии в каждом направлении. Ее значения в этом случае задаются в виде двухэлементного списка целых чисел со знаком. Знак определяет направление. Если задано просто целое со знаком, то характеристическая кривая отображается только в соответствующем направлении, определяемом знаком числа. По умолчанию эта опция имеет значение [-10,10].
numchar	Гиперповерхность решения строится из характеристических линий уровня, каждая из которых проходит через определенную точку начального многообразия. Эта опция определяет количество таких точек, задавая дискретное изменение каждого параметра, используемого в описании многообразия начальных значений. Задается в форме списка или одного целого которое относится ко всем параметрам. Значение по умолчанию равно 20.
scene	Определяет, что выводится на трехмерном графике решения. Задается в виде трехэлементного списка искомых функций или независимых переменных. Например, scene=[x1,x2,u(x1,x2)] означает, что по осям x и y графика отображаются переменные x1 и x2 , а по вертикальной оси z функция решения (это значение по умолчанию).
xi=xmin..xmax, u(x1,x2,..xn)= u_min..u_max	Можно определять диапазон изменения отображаемых величин, указанных в опции scene . Если какая-то из отображаемых величин вышла за пределы заданного для нее диапазона изменения, то процесс отображеия решения на этом прекращается.
obsrange	Установка этой опции, равной булеву значению TRUE (по умолчанию),

	останавливает процесс интегрирования вдоль характеристической кривой решения дифференциального уравнения, как только значение какой-либо отображаемой переменной выходит за заданный для нее диапазон изменения. Значение FALSE отменяет это действие.
method	Определяет численный метод интегрирования вдоль характеристик. По умолчанию используется внутренний метод Рунге-Кутта. Можно устанавливать любой метод, котороый используется в команде dsolve() при численном интегрировании обыкновенных дифференциальных уравнений, но следует учитывать, что использование метода, отличного от внутреннего, приводит к существенному увеличению времени расчета.
animate	В общем случае дифференциального уравнения, в котором неизвестная функция зависит от n прерменных, решение представляет собой n-мерную гиперповерхность в (n+1)-мерном пространстве. Для n>2 ее? естественно, сложно отобразить в трехмерном пространстве. Можно моздать анимационную картинку, отображающую последовательность многообразий, которые все вместе составляют поверхность решения. Для этого следует установить значение опции animate равной true . При значении опции animate=false отображается гиперповерхность решения с выделенным черным цветом начальным условием. Если animate=only, то поверхность решения никак не отображается, а вместо нее отображается последовательность начальных многообразий, из которой можно составить представление о поверхности решения. По умолчанию animate=true при n=2 и animate=false при n>2.
ic_assumptions	Определяет список возможных начальных условий для первых производных неизвестных функций в случае нелинейного уравнения первого порядка в частных производных.
basechar	Значение этой опции определяет, будут ли отображаться базовые кривые характеристик (проекции начальных условий на плоскость x-y). Если ее значение равно true , то отображаются, если равно false, то не отображаются. При значении, равном only, будут отображаться только базовые характеристики и кривая начальных данных - поверхность решения отображаться не будет. Значение по умолчанию - false.
color	Определяет используемый для закрашивания поверхности цвет.
initcolor	Определяет цвет, используемый для закрашивания гиперповерхностей начальных условий.

Остальные опции соответствуют опциям команды **plot3d()** (кроме grid, gridstyle, numpoints) и **dsolve[numeric]**, страницы справки которых можно отобразить командами **?plot3d[options]** и**?dsolve[numeric]** .

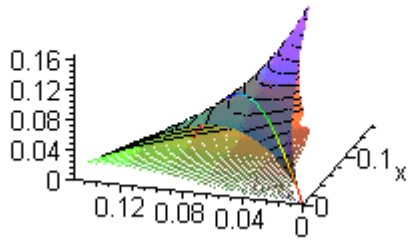
В примере ниже с помощью опций задано отображение наряду с поверхностью решения и базовых характеристик, причем для начальной кривой определена функциональная (*cos(t)*t*) окраска в зависимости от параметра *t*

.

```
> pde3 := (y^2+z(x,y)^2+x^2)*diff(z(x,y),x) -
  2*x*y*diff(z(x,y),y)- 2*z(x,y)*x = 0;
```

$$pde3 := (y^2 + z(x,y)^2 + x^2)\,z_x - 2\,x\,y\,z_y - 2\,z(x,y)\,x = 0$$

```
> PDEplot(pde3, [t,t,sin(Pi*t/0.1)/10], t=0..0.1
  numchar=40,orientation=[-163,56],
  basechar=true, numsteps=[20,20],
  stepsize=.15,initcolour=cos(t)*t,
  animate=false, style=PATCHCONTOUR);
```

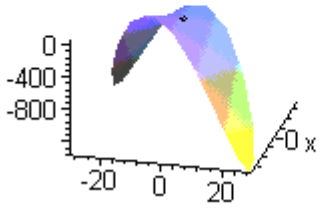


Для нелинейного уравнения необходимо задать начальные условия на производную `ic_assumptions`.

```
> pde1 :=
  diff(u(x,y),x)*diff(u(x,y),y)-x*y+u(x,y)=0;
```

$$pde1 := u_x\,u_y - x\,y + u(x,y) = 0$$

```
> PDEplot(pde1, [cos(t),sin(t),0], t=-Pi..Pi,
  ic_assumptions=[diff(u(x,y),x) = -cos(t)]);
```



Задание 6.

1. Найдите общее решение дифференциального уравнения:
$x(x-1)\,y'+y^3=xy$
Проверьте правильность 1-го из полученных решений, используя разные способы: команды *assign, subs* и *odetest*

2. Решите задачу Коши для обыкновенного дифференциального уравнения,
$yy''=2xy'^2$, $y(2)=2$, $y'(2)=0.5$.
Протестируйте уравнение с помощью функции *odeadvisor*

3. Решите краевую задачу для обыкновенного дифференциального уравнения
$y''+y=1$, $y(0)=0$, $y(1)=1$.

4. Найдите общее решение системы дифференциальных уравнений
$x''=3x+4y$, $y''=-x-y$.
Присвойте полученные решения функциям $x(t)$ и $y(t)$.

5. Найдите общее решение дифференциального уравнения в частных производных 1-го порядка:
$xz_x+yz_y=z-xy$

.

6. Используя функцию **DEplot()**, постройте решение уравнения из п.5 с начальными условиями: при $x=2\ z=y^2+1$

7. Изобразите поверхность $f(x,y)=xy^3$.

8. Создайте анимационную картинку поверхности
$F(x,y)=cos(t\ x)sin(t\ y)$, где $x=-1..1,\ y=-1..1,\ t=1..2$.