Г.В.Прохоров, М.А.Леденев, В.В.Колбеев

ПАКЕТ СИМВОЛЬНЫХ ВЫЧИСЛЕНИЙ MAPLE V

Впервые на российском рынке издана книга, в которой рассматривается один из немногих современных пакетов символьных вычислений Maple V. Подробно описаны интерфейс, правила общения с пакетом и внутренний язык. Более 400 примеров иллюстрируют возможности пакета для решения задач из таких областей математики, как линейная и нелинейная алгебра, дифференциальное и интегральное исчисление, геометрия, операции над графами и группами, комбинаторика и многих других.

Книга предназначена для студентов, инженеров, научных работников и всех тех, кто занимается решением математических задач в общем виде и численным анализом моделей.

СОДЕРЖАНИЕ

Список терминов Maple V, использованных в книге

A

abs - абсолютное значение числа;
addedge - добавление ребра в граф;
addvertex - добавление вершины в граф;
adjacency - генерация матрицы смежности для графа;
alias - определение сокращения;
allvalues - вычисление всех возможных значений в выражении с RootOf;
and - логическое "И";
angle - определяет процедуру - оператор в нотации "<...!..>";
animate - анимация графиков;
antisymmetric - индексная функция - несимметричная;
appendto - дозапись результатов в существующий файл;
array - создание массива;
arrow - определяет процедуру - оператор в нотации "->";

B

binary - двоичный тип данных;
break - выход из цикла;
builtin - определяет функцию как встроенную;
by - приращение счетчика цикла;

C

C - перевод выражения в код языка C;
cat - вырезать выражение;
charpoly - нахождение характеристического многочлена матрицы;
close - закрытие файла;
complete - создает полный граф;
cond - число обусловленности матрицы;
conformal - построение комплексной функции;
constant - тип данных - константы;
convert - преобразование типов;
Copyright - авторские права на процедуру;

D

D - оператор дифференцирования;
Dchangevar - подстановка новых переменных в уравнение;
define - определение оператора и его свойств;
delete - удаление вершин и ребер из графа;
DEplot - построение решений дифференциальных уравнений и систем;
DEplot1 - построение решения

уравнения первого порядка;

DEplot2 - построение решения системы из двух уравнений;

description - секция описания процедуры;

det - вычисление определителя матрицы;

DEtools - графическая библиотека;

dfieldplot - построение поля решения дифференциального уравнения;

diagonal - индексная функция - диагональная;

diff - дифференцирование;

display3d - построение графика 3D по специальной структуре данных;

do - начало тела цикла;

draw - построение чертежа графа;

dsolve - решение дифференциальных уравнений;

duplicate - создание копии графа;

E

edges - список ребер графа;

eigenvals - вычисление собственных значений матрицы;

eigenvects - вычисление собственных векторов матрицы;

elif - конструкция "else - if;

else - конструкция "иначе";

end - завершение описания тела процедуры;

ends - список "хвостов" ребер графа;

entries - значения таблицы;

ERROR - завершение выполнения процедуры с сообщением об ошибке;

eval - точное вычисление выраженеия;

evalb - вычисление логического выражения;

evalf - вычисление выражения в числах с плавающей запятой;

evalm - вычисление матричного выражения;

eweight - нахождение весов ребер графа;

F

factorial - вычисление факториала;

FAIL - прерывание процедуры в случае невычисляемого выражения;

false - "ложь" - зарезервированная константа;

fi - окончание конструкции ветвления if;

float - тип данных - число с плавающей запятой;

flow - нахождение максимального потока в графе;

for - конструкция цикла "для";

fortran - трансляция выражения в код языка Fortran;

fraction - тип данных - дробь;

from - начальное значение

счетчика цикла;

fsolve - решение уравнения в числах с плавающей запятой;

H

hastype - проверка на указанный тип;

head - нахождение "голов" ребер графа;

I

I - мнимая единица (зарезервированная константа);

identity - индексная функция - единичная;

if - конструкция ветвления "если";

igcd - наибольший общий делитель;

ilcm - наименьший общий множитель;

in - конструкция цикла для перечисляемых типов данных;

incidence - определение матрицы инцидентности графа;

infinity - "бесконечность" - зарезервированная константа;

int - вычисление интеграла;

interface - установка интерфейсных переменных;

intersect - пересечение множеств;

inverse - нахождение обратной матрицы;

iquo - частное;

irem - остаток;

isplanar - проверка планарности графа;

isqrt - квадратный корень (целочисленное приближение);

J

jacobian - вычисление якобиана от вектора функций;

L

latex - вывод выражения в редактор LATEX;

length - определение длины выражения;

lhs - выделение левой части выражения;

limit - вычисление предела;

linalg - библиотека линейной алгебры;

local - секция описания локальных переменных процедуры;

M

macro - определение макрообозначений;

map - задание операции над всеми элементами выражения;

matrix - задание матрицы;

max - нахождение максимального элемента;

member - принадлежность элемента множеству;

min - нахождение минимального элемента;

minor - распечатка минора матрицы;

minus - вычитание множеств;

mod - остаток от деления;

multiply - умножение матриц;

N

new - создание пустого графа;
nops - подсчет количества элементов;
0
op - извлечение элементов из выражения;
open - открытие файла;
P
petersen - создание графа Петерсена;
phaseportrait - построение фазового портрета;
plot - построение графика;
plot3d - построение графиков 3D;
print - распечатка содержимого;
printlevel - глобальная переменная, используется для отладки процедур;
proc - задание процедуры;
R
rank - нахождение ранга матрицы;
random - генерация случайного графа;
replot - перерисовать график;
restart - очистка значений переменных;
read - чтение из файла;
readlib - чтение библиотечной функции;
rhs - выделение правой части выражения;
rsolve - решение рекурентных выражений;

S

save - запись выражений в файл;
seq - генерация последовательности;
SearchText - поиск текста в строке;
show - возвращает таблицу, содержащую всю информацию о графе;
shortpathtree - нахождение дерева кратчайших путей в графе;
signum - знак числа;
solve - решение уравнений и систем уравнений в символьном виде;
Sum - распечатка суммы ряда;
sum - нахождение суммы ряда;
subs - подстановка в выражение;
T
taylor - разложение в ряд Тейлора;
table - создание таблицы;
tail - возвращает имя "хвоста" графа;
type - проверка принадлежности типу;
U
union - объединение множеств;
V
vector - задание вектора;
vertices - просмотр узлов графа;
void - создает граф без ребер;
vweight - находит вес вершины;

W

whattype - определение типов;
with - подключение библиотеки;
writeto - запись в новый файл;
write - запись в файл;
writeln - запись строки в файл.

Алфавитный указатель

Вместо введения

Мысль о том, что инженерная деятельность – это всегда вычисления – не нова. Однако «старая » школа инженеров вкладывала в это совсем другой смысл. Подразумевались, в первую очередь, символьные вычисления, а на последнем этапе численные оценки.

Прогресс в познании мира потребовал решения задач либо очень сложных, либо не решаемых аналитически. В вузах стало модным слушать такие дисциплины, как «Численные методы», «Численный анализ» и т.п. Развитие вычислительной техники способствовало созданию мегабайтных программ, "взламывающих" сложные задачи. За методами потянулись алгоритмы и программы, на создание или поиск которых растрачивалась уйма «золотого» времени. К тому же специалисты, втянутые в эту гонку, отмечают, что за массой чисел, таблиц и графиков легко потерять понимание реальности результатов решения.

В последнее десятилетие «программное зло» породило «добро» – стали появляться универсальные пакеты для численных методов : MatLab, MathCAD, электронные таблицы и многие другие. Все они снимали с инженера половину «головной боли» – поиск алгоритмов и программ.

Первым откровением для специалистов стал пакет символьных вычислений DERIVE, который выполнял упрощения довольно сложных алгебраических выражений, оперировал с символьными матрицами, дифференцировал и интегрировал выражения в общем виде. Вслед за ним в России появился знаменитый пакет «Mathematica», рассчитанный на математиков-профессионалов. К сожалению, из–за сложности интерфейса и, главное, из–за отсутствия справочной литературы на русском языке он так и остался невостребованным, несмотря на свои фантастические возможности в области символьных вычислений.

Совершенно другая судьба складывается у пакета Maple, разработанного в университете Ватерлоо (Канада). Его версия, модифицированная под ОС Windows, поистине стала ярким бриллиантом среди подобного класса программных пакетов. С одной стороны, его интерфейс интуитивно понятен, а правила работы предельно просты, с другой же стороны возможности внушительны. Появившись в России, Maple V стал любимым средством для решения задач у математиков–профессионалов, инженеров. В студенческой же среде он вызывает эйфорию, так как легко справляется с труднейшими преобразованиями и вычислениями при дипломном проектировании и в курсовых заданиях. Этот факт беспокоит преподавателей:«Как бы не разучились думать...».

Досадным на сегодняшний день является отсутствие в нашей стране книг по этому пакету, «говорящему на английском языке», что и привело меня в университет города Хельсинки в библиотеку факультета информатики. Читая литературу по этому замечательному и, как оказалось, очень непростому пакету, у меня возникла мысль о необходимости выпустить справочное пособие, помогающее специалистам и студентам работать с Maple и даже программировать.

Прошло более чем полгода, и задуманный справочник появился. Позади самое интересное – открытие нового, многократное перекраивание текста, редактирование, проблемы с изданием. Надеемся, что наш совместный труд будет Вам полезен при общении с пакетом XXI века Maple.

Доцент кафедры «Системы автоматического управления»
КФ МГТУ им.Н.Э.Баумана, канд. техн. наук

Геннадий Прохоров

Посвящается светлой памяти
выдающегося математика
Павла Петровича Коровкина

1. ОБЩИЕ СВЕДЕНИЯ. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

Maple – это программный пакет для автоматизации символьных, численных и графических вычислений. Он может решать как простые, так и довольно сложные задачи. Ну а если вы находились в "каменном веке" и решали ваши математические и инженерные проблемы с помощью языков программирования, то у вас появился реальный шанс попасть сразу в век XXI – в среду Maple, которая стала предвестником возврата математики и инженерной деятельности человека к эпохе романтизма символьных вычислений прошлых веков, когда гениальные открытия совершали "на кончике пера". Широта функциональных возможностей Maple поражает – она охватывает такие разделы, как линейная алгебра, дифференциальные вычисления, геометрия, статистика и многое, многое другое. По каждому разделу написано большое количество процедур и функций, которыми можно воспользоваться, набрав имя одной из них в командной строке Maple.

Символьные вычисления

Maple выдает ответ в самой точной форме — символьной, более точной, чем любой из численных методов. Однако, если вы хотите получить ответ в виде числа с плавающей точкой, то он будет найден в конце символьных вычислений. Таким образом, погрешность метода – это лишь погрешность округления!

Решения получаются компактными, можно сказать – изящными. Посмотрите, как по команде solve (решить) выдается решение для системы из трех алгебраических уравнений:

>solve({a*x+b*y=1, 2*c*x+y=3, z – y=5}, {x,y,z});

$$\left\{y = \frac{3a - 2c}{ad - 2cb}, x = -\frac{3b - d}{ad - 2cb}\right\}$$

Численные вычисления

Численные вычисления – альтернативный путь нахождения решения в тех случаях, когда символьный метод слишком медленно работает над данной задачей или решение в символьном виде вообще не существует. Maple поддерживает почти все существующие численные методы. Все символьные константы могут быть приближены с точностью до любого знака, так как среда Maple имеет "бесконечную точность".

Примеры численных вычислений:

>evalf(Pi,25); # Представление числа "пи" до двадцать пятого знака

3.1415926535897932384626433

>0.5*(1/3);

1666666667

>sum((– 1)^i/i!,i=0..20);

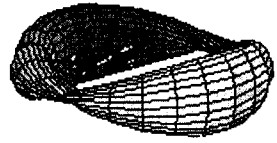
$$\frac{4282366656425369}{11640679464960000}$$

Графика

Возможности среды Maple строить двух- и трехмерные изображения обеспечивает вас мощным орудием визуализации научных вычислений. Функции одной и двух переменных, а также параметрические выражения могут быть представлены в графическом виде. Доступна анимация для создания эффекта движения.

Пример построения трехмерного графика:

>plot3d([cos(t)*(1+.2*sin(u)),sin(t)*(1+.2*sin(u)),.2*sin(t)*cos(u)],t=0..2*Pi, u= – Pi..Pi);



Внутренняя структура среды Maple

Среда Maple состоит из трех компонентов: ядра, библиотеки и экранного интерфейса. Ядро – это "математическое орудие", выполняющее все виды вычислений. Оно написано и откомпилировано на языке программирования С и выполняет основную часть вычислений, производимых системой.

Библиотеки включают встроенные процедуры, процедуры, написанные в среде Maple на его собственном языке программирования и сохраняемые в отдельном файле. Текст команд и программ, написанный в Maple, не компилируется, а интерпретируется, что позволяет вам создавать собственные процедуры интерактивно в пределах среды.

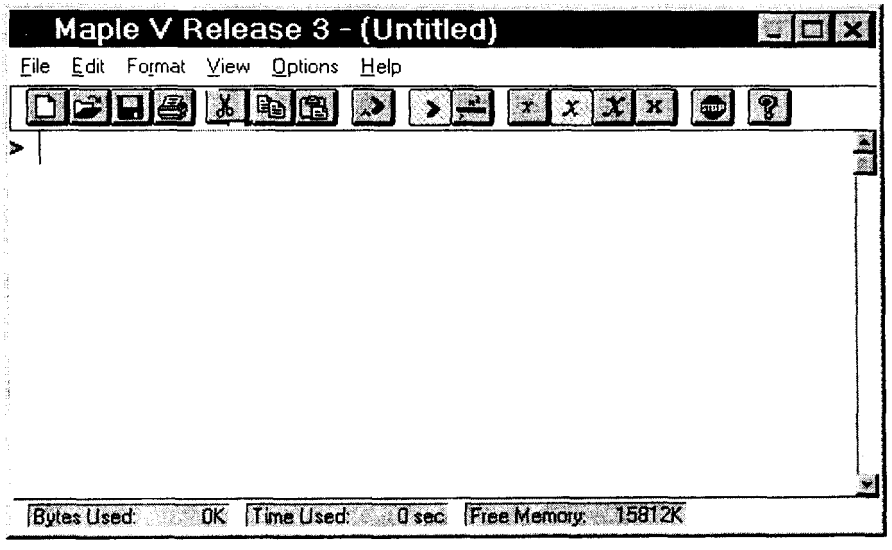
Интерфейс Maple – это вид среды и ее глаза в мир. Он зависит от качества вашего дисплея, версии приобретенной программы. Версия под WINDOWS преобразила интерфейс, сделав его действительно "дружественным" и интуитивно понятным пользователю, работающему с этой удивительной и романтичной средой.

ЭКРАННЫЙ ИНТЕРФЕЙС MAPLE

Maple V присущи все особенности windows–приложений и все прелести работы с ними. Его экранный интерфейс включает следующие составные части:

- строку меню команд;
- строку пиктограмм;

- рабочее окно, где священнодействует пользователь, выполняя сложнейшие математические преобразования. Обратите внимание на отсутствие горизонтальной полосы скроллинга. Maple всегда форматирует содержимое в пределах ширины рабочего окна;
- расположенная в нижней части "строка статуса", где размещаются ресурсная информация об использованной и свободной памяти, временные характеристики и т.п.



Меню команд

Меню включает шесть команд, обращение к которым вызывает выпадающие подменю.

File – отвечает сложившемуся стандарту и включает следующие подкоманды:

New	– открытие нового документа;
Open...	– открытие документа, сохраненного на диске;
Save	– сохранение текущего документа;
Save as...	– сохранение документа под новым именем;

Save settings – сохранение установок текущего сеанса работы;
Print... – вывод документа на принтер;
Printer setup... – изменить установки принтера;
Page Margins... – изменить установки формата текста на листе;
Exit – выход из программы.

Edit – редактирование документа (в скобках показаны "быстрые клавиши"):
Cut (Ctrl + X) – вырезать выделенный фрагмент в буфер;
Copy(Ctrl + C) – скопировать выделенный фрагмент в буфер;
Paste(Ctrl + V) – вставить содержимое буфера начиная от текущего курсора;
Delete (Del) – удалить выделенный фрагмент;
Copy to – сохранить выделенный фрагмент в файле.

Format – изменение структуры или вида рабочего документа:
Text Region(F5) – переход в текстовой режим работы и обратно;
Split Group (F3) – расщепить объединенную группу строк;
Join Group (F4) – объединить в группу указанную строку;
Insert Page Break – вставить принудительное разделение (страницы);
Insert New Region – вставить строку ввода:
Above (Ctrl+O) – вставить над текущей строкой;
Below (Ctrl+I) – вставить под текущей строкой;
Remove All – удалить из документа все:
Input – удалить строки ввода;
Output – удалить строки вывода;
Text – удалить текстовые регионы;
Graphica – удалить графики;
Fonts – изменить шрифт:
Input – изменить шрифт строк ввода;
Output – изменить шрифт строк вывода (только для псевдографики);
Text – изменить шрифт текстовых регионов;
Math Style – требования к представлению вычисляемых математических выражений:

Large – большими символами;
Medium – средними символами;
Small – маленькими символами;
Character – представление формул через псевдографику;
Execute Worksheet – последовательное выполнение всех команд в рабочем окне.

View – операции контроля вида представления окон:
Separator Lines (F9) – вывод на дисплей и бумагу разделительных линий;
Status Bar (F2) – убрать строку статуса;
Tool Bar – убрать строку пиктограмм;
Plot Tool Bar – убрать строку пиктограмм в графических окнах.

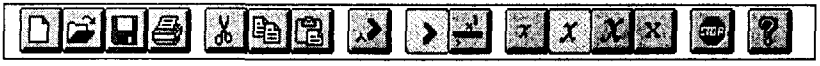
Options – опции контроля режимов работы:
Save Kernel State – контроль автоматического сохранения результатов в файле;
Confirmation Checks – подтверждение записи документа;
Fast Graphics Redraw – быстрая перерисовка графиков;
Automatic Save Settings – автоматическое сохранение установок;
Replace mode – переключение режима замены на вставку в строке вывода;
Continuous mode – после пересчета не вставляется новая строка ввода;
Insert Mode – после пересчета вставляется новая строка ввода;
Insert at end mode – после пересчета вставляется строка ввода в конец текста.

Строка пиктограмм

16 пиктограмм разбиты на семь групп. Ряд из них дублируют некоторые команды меню.

Первая группа из четырех пиктограмм отражает общепринятое обозначение стандартных процедур: открыть новый файл, загрузить файл с диска, быстрое его сохранение и вывод на принтер.

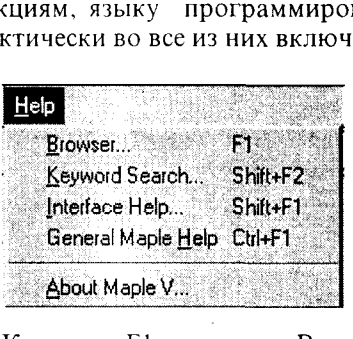
Следующие три – соответственно вырезать, скопировать и вставить фрагмент, используя промежуточный буфер Windows.



Восьмая пиктограмма вставляет строки ввода. Девятая включает и выключает знак ">" в начале строки ввода, а десятая – разделительные линии. Пиктограммы с 11 по 14 управляют качеством шрифтов. Пятнадцатая – позволяет прервать вычислительные процессы системы. Ну, а традиционный знак вопроса вызывает "Browser" навигатор встроенного справочника.

Справочная система Maple

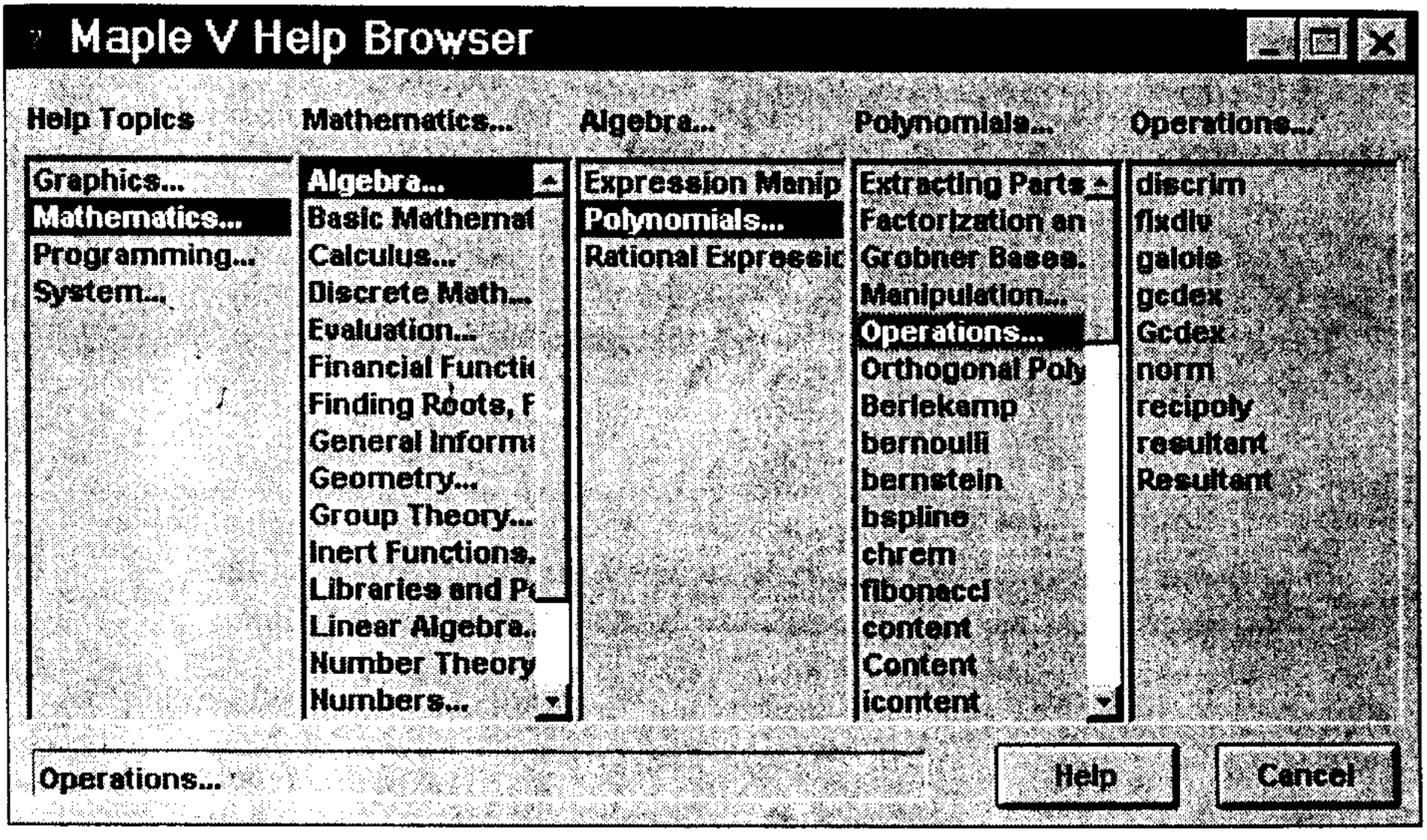
Справочная система Maple удовлетворит самого взыскательного пользователя. В ней помещены бесчисленные подробные справки по всем функциям, языку программирования, командам управления ресурсами. Практически во все из них включены многочисленные примеры.



Команда HELP вызывает выпадающее меню (см. рисунок). Кроме того, возможны поиск по ключевому слову (Shift+F2), получение справки о командах и пиктограммах интерфейса системы (Shift+F1) и, главное, получение справки по словоформе, на которой стоит курсор (Ctrl +F1).

Клавиша F1 вызывает Browser, с древовидным представлением информации по семи уровням. Пользователь может получить более 2200 подробнейших справочных статей, представляющих собой фрагменты гипертекстового файла с перекрестными ссылками и "живыми примерами",

которые можно копировать в рабочую среду и проигрывать со своими данными. Ниже показан вид описываемого справочного навигатора. У каждого уровня может быть полоса скроллинга. Выбор осуществляется мышью или клавишами курсора.



Названия тем в Browser, за которыми следуют три точки, имеют подразделы.

2. РЕЖИМЫ РАБОТЫ С ЧИСЛЕННЫМИ ВЫРАЖЕНИЯМИ

Любое введенное выражение в Maple может заканчиваться символом ":" либо символом ";". В первом случае результат не будет выводиться на экран, во втором – будет.

Пример:

```
> x:=0.5*(2/3):
```

```
> x:=0.5*(2/3);
```

$$x := .3333333334$$

Если мы хотим использовать выражение, введенное ранее, то необходимо воспользоваться символом " " ".

Причем:

" – воспользоваться выражением, введенным перед данным;
"" – воспользоваться введенным за одно выражение ранее;
""" – воспользоваться введенным за два выражения ранее.

Пример:

```
> x:=1/3:
```

```
> y:=2/3:
```

```
> z:=4/9:
```

```
> f:="+""+""";
```

$$f := \frac{13}{9}$$

Чтобы приблизить выражение, полученное в символьном виде числом с плавающей точкой, служит команда evalf.

Пример:

```
> evalf(");
```

1.444444444

Изменить точность представления числа можно при помощи глобальной переменной Digits. Например, распечатаем число "пи" с точностью 30 знаков.

> Digits:=30;

$$Digits := 30$$

> evalf(Pi);

$$3.14159265358979323846264338328$$

3. ПРИМЕРЫ ВЫЧИСЛЕНИЙ В СРЕДЕ MAPLE V

3.1. Дифференцирование

В среде Maple V можно легко найти как стандартные дифференциалы, так и дифференциалы от любых функций, а также частные производные.

Для нахождения производных от выражений существует команда diff:

```
>f:=sin(x)^2:  diff(f,x);
```

$$2\sin(x)\cos(x)$$

Найдем производные от функции двух переменных:

```
> Ffunction:=cos(x)*y+sin(y)*x;
```

$$Ffunction := \cos(x)\,y + \sin(y)\,x$$

Производная по x:

```
> diff(Ffunction,x);
```

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

Производная по y:

```
> diff(Ffunction,y);
```

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

Для нахождения производных более высокого порядка достаточно указать после аргумента дифференцирования знак $ и порядок производной:

```
> diff(Ffunction,x$2);
```

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

Существует особый оператор D, применимый для дифференцирования операторов и задания начальных условий при решении дифференциальных уравнений.

К примеру, найдем производную от операторов cos и tan:

> D(cos); D(tan);

$$-\sin$$

$$1+\tan^2$$

Оператор D можно использовать для дифференцирования пользовательских функций. Например, зададим функцию f=x^2 и вычислим ее производную:

> f:=x -> x^2: D(f);

$$x \to 2x$$

Зададим функцию двух переменных:

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

$$f := (x,y) \to e^{(xy)}$$

Найдем производную от функции f по переменной x. Для этого в квадратных скобках вслед за оператором D необходимо указать, по какому элементу множества неизвестных следует находить производную:

> D[1](f);

$$(x,y) \to y\, e^{(xy)}$$

Производная по y:

> D[2](f);

$$(x,y) \to x\, e^{(xy)}$$

Следующая запись означает, что сначала берется производная по x, а затем по y:

> D[1,2](f);

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

3.2. Интегрирование

С помощью Maple можно находить как определенные, так и неопределенные интегралы, используя команду int. К примеру, найдем неопределенный интеграл от функции **x**:

> int(x,x);

$$\frac{1}{2}x^2$$

Значение того же интеграла на промежутке от 0 до 5 определяется следующим образом:

> int(x,x=0..5.);

$$12.50000000$$

Богатый аппарат встроенных математических функций позволяет найти значения интегралов, не выражающиеся в аналитической форме:

> var3:=int(cos(x^2),x);

$$var3 := \frac{1}{2}\sqrt{2}\,\sqrt{\pi}\ \mathrm{FresnelC}\left(\frac{\sqrt{2}\,x}{\sqrt{\pi}}\right)$$

Здесь функция FresnelC определена в среде Maple.

Найдем значение интеграла от cos(x^2) на отрезке от 0 до 1:

> var3:=evalf(int(cos(x^2),x=0..1));

$$var3 := .9045242375$$

3.3. Пределы

Среду Maple можно использовать для нахождения пределов от различных функций.

Определим функцию f:

```
> f:=1/x;
```

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

Найдем предел функции f при x, стремящемся к бесконечности (бесконечность в Maple обозначается как *infinity*):

```
> limit(f,x=infinity);
```

$$0$$

Если указать заглавную букву, то предел не будет вычислен, а лишь записан:

```
> Limit(f,x=infinity);
```

$$\lim_{x \to \infty} \frac{1}{x}$$

Можно найти пределы справа (right) и слева (left) в определенной точке. Предел функции f при x стремящемся к нулю слева:

```
> limit(f,x=0,left);
```

$$-\infty$$

3.4. Ряды

В Maple можно легко задать любой ряд, а также вычислить его сумму. При этом команда Sum лишь распечатает ряд, а команда sum найдет его сумму:

> Sum(1/n!,n=1..infinity);

$$\sum_{n=1}^{\infty} \frac{1}{n!}$$

> sum(1/n!,n=1..infinity);

$$e - 1$$

Возможны следующие операции для нахождения сумм:

> sum(i^2,i=0..6);

$$91$$

> sum(i^2,i);

$$\frac{1}{3}i^3 - \frac{1}{2}i^2 + \frac{1}{6}i$$

> sum(a[i],i=1..4);

$$a_1 + a_2 + a_3 + a_4$$

> sum(i/(i+1),i=0..n);

$$n + 1 - \Psi(n+2) - \gamma$$

Maple выразил значение суммы i/(i+1) при i=0..n через значения встроенных математических функций PSI и GAMMA.

Определим предел сходимости ряда, когда n меняется от 1 до бесконечности:

> sum(n^2/3^n,n=1..infinity);

$$\frac{3}{2}$$

Возможности Maple для разложения функций в ряд огромны. Включено разложение по ортогональным полиномам, в ряд Тейлора и т.д. Например, разложим в ряд Тейлора функцию exp(x) относительно точки x=0 и возьмем первые четыре члена ряда:

> taylor(exp(x), x=0, 4);

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

Возьмем неопределенный интеграл:

> int(exp(x^3), x);

$$\int e^{(x^3)}\,dx$$

Среда Maple не смогла найти значение интеграла от exp(x^3), но можно попробовать приблизить значение интеграла рядом Тейлора:

> taylor(", x=0);

$$x + \frac{1}{4}x^4 + O(x^7)$$

При разложении в ряд Тейлора Maple добавляет к результату разложения остаточный член некоторого порядка. Чтобы выполнить вычисления с полученным рядом, его надо преобразовать в многочлен, при этом остаточный член будет отброшен:

> convert(",polynom);

$$x + \frac{1}{4}x^4$$

4. СТРУКТУРЫ ДАННЫХ В MAPLE V

4.1. *Последовательности, множества и списки*

Последовательность – это набор элементов, разделенных запятыми, без скобок. Например:

> S:=1,2,3,4,5,6;

$$S := 1, 2, 3, 4, 5, 6$$

Для генерации последовательности в среде Maple служит команда seq. Синтаксис вызова данной команды:

seq(y, i = m..n)
seq(y, i = x)

Параметры: y – любое выражение; i – имя; m, n – численные параметры; x – выражение.

Наиболее распространенный вызов: seq(f(i), i = 1..n) , который генерирует последовательность f(1), f(2), ..., f(n). Менее употребимый – seq(f(i), i = m..n), создающий последовательность f(m), f(m+1), f(m+2), ..., f(n). Здесь m и n могут быть не только типа integer.

> seq(sin(Pi*i/6), i = 0..3);

$$0, \frac{1}{2}, \frac{1}{2}\sqrt{3}, 1$$

> L := seq(i, i = 0..6);

$$L := 0, 1, 2, 3, 4, 5, 6$$

Построим последовательность, состоящую из остатков от деления элементов списка L на 7:

> seq(i^2 mod 7, i = L);

$$0, 1, 4, 2, 2, 4, 1$$

Множества принято обозначать фигурными скобками. Им присущи все правила преобразования, принятые в классической математике.

> set1:={sin,cos,tan,cos};

$$set1 = \{\cos, \sin, \tan\}$$

Используются следующие операции преобразования множеств:

op	– извлечение элементов;
nops	– подсчет количества элементов;
union	– объединение множеств;
intersect	– пересечение множеств;
minus	– вычитание множеств;
member	– принадлежность элемента множеству;
seq	– генерация последовательности;
convert	– преобразование множества в другие структуры.

Извлечение элементов с первого по второй из множества set1:

> op(1..2,set1);

$$\cos, \sin$$

Определение количества элементов в множестве set1:

> nops(set1);

$$3$$

Объединение двух множеств {a,b} и {b,c}:

> {a,b} union {b,c};

$$\{a, b, c\}$$

Пересечение:

> {a,b} intersect {b,c};

$$\{b\}$$

Вычитание:

```
> {a,b} minus {b,c};
```

$$\{a\}$$

Принадлежность элементов 'a' и 'cos' множеству set1:

```
> member(a,set1);
```

false

```
> member(cos,set1);
```

true

Генерация последовательности 'p' и множества L:

```
> p:=seq(x,x=1..4);
```

$$p := 1, 2, 3, 4$$

```
> L := {seq(y[i],i=1..4)};
```

$$L := \{y_3, y_4, y_1, y_2\}$$

Добавление элемента к множеству L:

```
> L := {op(L),z[5]};
```

$$L := \{y_1, y_2, y_3, y_4, z_5\}$$

Удаление второго элемента из множества L:

```
> L := subsop(2=NULL,L);
```

$$L := \{y_1, y_3, y_4, z_5\}$$

Список принято обозначать квадратными скобками.

> list1:=[sin,cos,tan,cos];

$$list1 := [\sin, \cos, \tan, \cos]$$

Со списками можно производить математические операции, например дифференцирование:

> D(list1);

$$[\cos, -\sin, 1 + \tan^2, -\sin]$$

Приведем некоторые операции над списками:

op	– извлечение элементов из списка;
nops	– подсчет количества элементов в списке;
member	– принадлежность списку;
convert	– преобразование в другие структуры.

Извлечение двух элементов из списка list1:

> op(1..2,list1);

sin, cos

Определим количество элементов в списке list1:

> nops(list1);

4

Проверим принадлежность элемента 'a' списку list1:

> member(a,list1);

false

Преобразование списка list1 во множество:

> convert(list1,set);

$$\{\cos, \sin, \tan\}$$

По отношению к спискам и множествам допустимы операции присваивания:

> list2:=list1;

$$list2 := [\sin, \cos, \tan, \cos]$$

4.2. Массивы и таблицы

Массив – конечномерный список с целочисленными индексами. Операции, применяющиеся к массивам:

array – создание массива;
print – распечатка содержимого массива;
map – задание операции над всеми элементами массива;
op – извлечение элементов (уточнение задания массива).

Создадим массив v:

> v := array(1..4);

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

Заполним этот массив элементами и распечатаем его содержимое:

> for i to 4 do v[i]:=i od:

> print(v);

$$[1 \quad 2 \quad 3 \quad 4]$$

Создадим одномерный массив 's' с нулевыми значениями:

> s:=array(1..2,[0,0]);

$$s := [0 \quad 0]$$

Создадим двумерный массив 'm':

> m:=array(symmetric,1..2,1..2,[[cos(y),0],[0,sin(y)]]);

$$m := \begin{bmatrix} \cos(y) & 0 \\ 0 & \sin(y) \end{bmatrix}$$

Зададим операцию дифференцирования над всеми элементами массива 'm':

> map(diff,m,y);

$$\begin{bmatrix} -\sin(y) & 0 \\ 0 & \cos(y) \end{bmatrix}$$

Определим характеристики массива 'm':

тип массива

> op(1,eval(m));

symmetric

размер массива

> op(2,eval(m));

$$1 .. 2, 1 .. 2$$

элементы массива

> op(3,eval(m));

$$[(1, 2) = 0, (2, 2) = \sin(y), (1, 1) = \cos(y)]$$

В отличие от массивов, где индексы – целочисленные значения, расположенные по порядку номеров, индексы у таблиц – любые значения. Таблица задается указанием слова "table":

> table( );

```
table([
      ])
```

Команда "table( )" создала таблицу с неопределенными значениями. Определим значения таблицы, выполнив команду:

> table([22,42]);

```
table([
      1 = 22
      2 = 42
      ])
```

Так как индексы таблицы не были определены, то программа сама присвоила им целочисленные значения, расположенные по порядку. Зададим индексы таблицы следующим образом:

> S := table([(red)=45,(green)=61]);

$$S := \text{table}([green = 61, red = 45])$$

Обратимся к элементам таблицы:

> S[1],S[red];

$$S_1, 45$$

Над таблицами возможны различные операции. Зададим таблицу F, элементами которой являются операторы:

> F:=table([y=(x –> x^2),cos= –sin]):

Распечатаем таблицу:

> print(F);

table([

cos = -sin

$y = \left(x \to x^2\right)$

])

Вычислим значение элемента таблицы 'y' от аргумента, равного 3:

> F[y](3);

9

Таким образом множества, списки, массивы, таблицы – это гибкая основа для создания более сложных структур данных.

5. МАТРИЧНЫЕ И ВЕКТОРНЫЕ ВЫЧИСЛЕНИЯ

Среда Maple позволяет выполнять все стандартные операции, определенные в линейной алгебре. Они становятся доступными при подключении библиотеки "linalg". Библиотеки подключаются через команду with с указанием ее имени. В нашем случае запишем:

> with(linalg):

Приведем некоторые из 107 доступных операций:

vector	– задание вектора;
matrix	– задание матрицы;
minor	– распечатка минора матрицы;
rank	– нахождение ранга матрицы;
transpose	– транспонирование матрицы;
det	– вычисление определителя;
cond	– число обусловленности матрицы;
inverse	– нахождение обратной матрицы;
eigenvals	– вычисление собственных значений матрицы;
eigenvects	– вычисление собственных векторов матрицы;
charpoly	– характеристический многочлен матрицы;
multiply	– умножение матриц;
jacobian	– вычисление якобиана от вектора функций.

Определим вектор 'a':

> a:=vector([2,x^2,4,5.3,alpha]);

$$a := \left[2 \quad x^2 \quad 4 \quad 5.3 \quad \alpha\right]$$

Зададим матрицу:

> A:=matrix([[1,1,1],[4,1,6],[7,1,9]]);

$$A := \begin{bmatrix} 1 & 1 & 1 \\ 4 & 1 & 6 \\ 7 & 1 & 9 \end{bmatrix}$$

Распечатаем минор матрицы A, отвечающий элементу, стоящему во второй строке первого столбца:

> minor(A, 2,1);

$$\begin{bmatrix} 1 & 1 \\ 1 & 9 \end{bmatrix}$$

Найдем ранг матрицы A:

> rank(A);

$$3$$

Транспонирование матрицы A:

> AT:=transpose(A);

$$AT := \begin{bmatrix} 1 & 4 & 7 \\ 1 & 1 & 1 \\ 1 & 6 & 9 \end{bmatrix}$$

Найдем определитель матрицы A:

> det(A);

$$6$$

Число обусловленности матрицы A (вычисленное по L – норме):

> cond(A,1);

$$\frac{128}{3}$$

Отыскание обратной матрицы:

> A1:=inverse(A);

$$A1 := \begin{bmatrix} \frac{1}{2} & \frac{-4}{3} & \frac{5}{6} \\ 1 & \frac{1}{3} & \frac{-1}{3} \\ \frac{-1}{2} & 1 & \frac{-1}{2} \end{bmatrix}$$

Перемножение матриц A и A1 (оператор &* используется для умножения матриц):

> E1:=evalm(A&*A1);

$$E1 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Преобразуем матрицу A, после чего найдем ее собственные значения. При этом используем команду array(identity,1..3,1..3) для создания единичной матрицы 3x3.

> B:=evalm(A – lambda*array(identity,1..3,1..3));

$$B := \begin{bmatrix} 1-\lambda & 1 & 1 \\ 4 & 1-\lambda & 6 \\ 7 & 1 & 9-\lambda \end{bmatrix}$$

> f:=det(B);

$$f := 6 - 2\,\lambda + 11\,\lambda^2 - \lambda^3$$

> res:=evalf(solve(f,lambda),4);

$$res := 10.87,\ .067 + .7395\,I,\ .067 - .7395\,I$$

Результат, полученный в ходе выполнения трех предыдущих команд, можно найти, проделав лишь одну операцию над матрицей A:

> evalf(eigenvals(A),4);

$$10.87, .067 + .7395\,I, .067 - .7395\,I$$

Найдем характеристический полином матрицы A:
> charpoly(A,lambda);

$$\lambda^3 - 11\,\lambda^2 + 2\,\lambda - 6$$

Собственные векторы матрицы A можно найти с помощью следующей команды:

> evalf(eigenvects(A,'radical'),4);

$$[10.87, 1., \{[1. \quad 3.978 \quad 5.893]\}],$$
$$[.067 + .7395\,I, 1., \{[1. \quad -.1833 + .9026\,I \quad -.7487 - .1631\,I]\}],$$
$$[.067 - .7395\,I, 1., \{[1. \quad -.1833 - .9026\,I \quad -.7487 + .1631\,I]\}]$$

Результат выдается в форме:
[num,r,{vect}],
где num – собственное значение матрицы;
r – кратность собственного значения;
vect – собственный вектор;
'*radical*' – ключ, определяющий режим нахождения всех собственных значений.

Создание случайной матрицы через генератор случайных чисел rand:
> G := array(1..3,1..3):

> r:=rand(100):

> for i to 3 do for j to 3 do G[i,j] := r() od od:

> evalm(G);

$$\begin{bmatrix} 81 & 70 & 97 \\ 63 & 76 & 38 \\ 85 & 68 & 21 \end{bmatrix}$$

Перемножим матрицу G на саму себя:

> multiply(G,G);

$$\begin{bmatrix} 19216 & 17586 & 12554 \\ 13121 & 12770 & 9797 \\ 12954 & 12546 & 11270 \end{bmatrix}$$

Вычислим якобиан от вектора A:

> A := vector([x^2, x*y, x*z]);

$$A := \left[x^2 \quad x\,y \quad x\,z \right]$$

> jacobian(A, [x,y,z]);

$$\begin{bmatrix} 2\,x & 0 & 0 \\ y & x & 0 \\ z & 0 & x \end{bmatrix}$$

6. РЕШЕНИЕ ЛИНЕЙНЫХ И НЕЛИНЕЙНЫХ УРАВНЕНИЙ И СИСТЕМ

6.1. Обыкновенные выражения

Для решения линейных и нелинейных уравнений и систем служит команда solve.

Например, зададим нелинейное уравнение:

> eqn1:=x^3 – 54*x^2+972*x – 5839=0;

$$eqn1 := x^3 - 54\,x^2 + 972\,x - 5839 = 0$$

Далее решим его относительно переменной x:

> solve(eqn1,x);

$$7^{1/3} + 18,\ -\frac{1}{2}7^{1/3} + 18 + \frac{1}{2}I\sqrt{3}\,7^{1/3},\ -\frac{1}{2}7^{1/3} + 18 - \frac{1}{2}I\sqrt{3}\,7^{1/3}$$

Получим приближенное решение до пятого значащего знака:

> evalf(",5);

$$19.913,\ 17.044 + 1.6567\,I,\ 17.044 - 1.6567\,I$$

Найдем решение нелинейного уравнения ln(x) – 2x(ln(x) – 2)=0:

> eqn3:=ln(x) – 2*x*(ln(x) – 1)=0;

$$eqn3 := \ln(x) - 2\,x\,(\ln(x) - 1) = 0$$

> a3:=solve(eqn3,x);

$$a3 :=$$

Следовательно, среде Maple не удалось найти решение в символьном виде, но можно попробовать найти решение в численном виде. Для этого служит команда fsolve:

```
> a3:=fsolve(eqn3,x);
```

$$a3 := 3.258475747$$

С помощью Maple можно находить решения линейных и нелинейных уравнений и систем. Например, рассмотрим систему из трех линейных уравнений.

```
> eqs1:= { x + 2*y + 3*z  = 6,
>          5*x + 5*y + 4*z  = 1,
>                3*y + 4*z  = 1};
```

$$eqs1 := \{5x + 5y + 4z = 1, x + 2y + 3z = 6, 3y + 4z = 1\}$$

```
> a1:=solve(eqs1, {x,y,z});
```

$$a1 := \left\{y = \frac{-105}{13}, z = \frac{82}{13}, x = \frac{42}{13}\right\}$$

```
> evalf(a1,4);
```

$$\{x = 3.231, z = 6.308, y = -8.077\}$$

Усложним пример, рассмотрев систему из трех линейных уравнений с параметром.

```
> eqs2:= { a*x + 2*y + 3*z  = 6,
>      5*x +a^2* 5*y + 4*z  = 1,
>                3*y + 4*z  = 1};
```

$$eqs2 := \left\{a x + 2y + 3z = 6, 5x + 5a^2 y + 4z = 1, 3y + 4z = 1\right\}$$

> a2:=solve(eqs2,{x,y,z});

$$a2 := \left\{ x = 21\,\frac{5\,a^2 - 3}{-12\,a + 20\,a^3 + 5},\ z = \frac{5\,a^3 - 3\,a + 80}{-12\,a + 20\,a^3 + 5},\ y = -105\,\frac{1}{-12\,a + 20\,a^3 + 5} \right\}$$

Рассмотрим решение систем нелинейных уравнений.

Пример 1 (среда выдает одно решение):

> eqs5 := {2*x*y = 1, x + z = 0, 2*x – 3*z = 2};

$$eqs5 := \{2\,x - 3\,z = 2,\ x + z = 0,\ 2\,x\,y = 1\}$$

> solve(eqs5,{x,y,z});

$$\left\{ x = \frac{2}{5},\ z = \frac{-2}{5},\ y = \frac{5}{4} \right\}$$

Проверим, правильно ли мы решили данную систему методом подстановки, при помощи команды subs. Здесь мы используем принятые обозначения "– последнее и ""– предпоследнее.

> subs(","");

$$\{2 = 2,\ 0 = 0,\ 1 = 1\}$$

Пример 2 (решений много):

> eqs6:={ x+ y+ z = 6,

> x^2 + y^2 + z^2 = 14,

> x^3 + y^3 + z^3 = 36};

$$eqs6 := \left\{ x^2 + y^2 + z^2 = 14,\ x^3 + y^3 + z^3 = 36,\ x + y + z = 6 \right\}$$

> solve(eqs6,{x,y,z});

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

6.2. *Дифференциальные выражения*

Maple эффективен для задач с различными типами дифференциальных уравнений и систем. Если это необходимо, то среда даст приближенное численное решение. Ко всему прочему, существует возможность использования преобразования Лапласа, численных методов Рунге – Кутта 2, 3, 4, 5 и даже 7 и 8-го порядка. Особо следует подчеркнуть возможность решения систем обыкновенных дифференциальных уравнений.

Например, возьмем простейшее дифференциальное уравнение $y'=y$:

> dsolve(diff(y(x),x) – y(x)=0,y(x));

$$y(x) = e^{x} \_C1$$

В ответе присутствует _C1 – произвольная константа, так как начальные условия не заданы. Если задать начальные условия, например y(2)=5 , то

> dsolve({diff(y(x),x) – y(x)=0, y(2)=5}, y(x));

$$y(x) = 5\frac{e^{x}}{e^{2}}$$

Возьмем пример с дифференциальным уравнением второго порядка:

> dsolve(diff(y(x),x$2) – y(x) = 1, y(x));

$$y(x) = -1 + e^{x} \_C1 + \_C2\, e^{(-x)}$$

Попросим Maple для уравнения $y''+5y'+6y=0$ с начальными условиями $y(0)=0$, $y'(0)=1$ применить метод, основанный на преобразовании Лапла-

са, т.е. включим в команду условие в виде дополнительного указания *method=laplace*:

> de1 := diff(y(t),t$2) + 5*diff(y(t),t) + 6*y(t) = 0;

$$de1 := \left(\frac{\partial^2}{\partial t^2} y(t)\right) + 5\left(\frac{\partial}{\partial t} y(t)\right) + 6\, y(t) = 0$$

> dsolve({de1, y(0)=0, D(y)(0)=1}, y(t),method=laplace);

$$y(t) = -e^{(-3t)} + e^{(-2t)}$$

Решим систему: $y'=z$, $z'=y$ с начальными условиями $y(0)=0$, $z(0)=1$ относительно y(x) и z(x):

> sys := diff(y(x),x)=z(x), diff(z(x),x)=y(x): fcns := {y(x), z(x)};

$$fcns := \{y(x), z(x)\}$$

> dsolve({sys,y(0)=0,z(0)=1}, fcns);

$$\left\{y(x) = \frac{1}{2}e^{x} - \frac{1}{2}e^{(-x)},\ z(x) = \frac{1}{2}e^{x} + \frac{1}{2}e^{(-x)}\right\}$$

К той же системе применим метод приближения функций рядами (series):

> dsolve({sys,y(0)=0,z(0)=1}, fcns, type=series);

$$\left\{z(x) = 1 + \frac{1}{2}x^2 + \frac{1}{24}x^4 + O\left(x^6\right),\ y(x) = x + \frac{1}{6}x^3 + \frac{1}{120}x^5 + O\left(x^6\right)\right\}$$

Очень изящен метод численного интегрирования. Например, для той же системы sys:

> F := dsolve({sys,y(0)=0,z(0)=1}, fcns, type=numeric):

Теперь можно найти значения решения в конкретных точках:

> F(0);

$$\left[x = 0, y(x) = .2890728569016066\ 10^{-7}, z(x) = .999999881386018\right]$$

> F(1);

$$[x = 1, y(x) = 1.175201093665006, z(x) = 1.543080478651408]$$

6.3. Рекуррентные выражения

Для рекуррентных выражений используется команда rsolve.

> rsolve(f(n) = – 3*f(n – 1) – 2*f(n – 2), f(k));

$$(2\,f(0) + f(1))\,(-1)^k + (-f(0) - f(1))\,(-2)^k$$

> rsolve(f(n) = 3*f(n/2) + 5*n, f(n));

$$f(1)\,n^{\left(\frac{\ln(3)}{\ln(2)}\right)} + n^{\left(\frac{\ln(3)}{\ln(2)}\right)}\left(-15\left(\frac{2}{3}\right)^{\left(\frac{\ln(n)}{\ln(2)} + 1\right)} + 10\right)$$

> rsolve(t(b*n) = a*t(n) + n, t(m));

$$t(1)\,m^{\left(\frac{\ln(a)}{\ln(b)}\right)} + m^{\left(\frac{\ln(a)}{\ln(b)}\right)}\left(-\frac{\left(\frac{b}{a}\right)^{\left(\frac{\ln(m)}{\ln(b)} + 1\right)} a}{b\,(-b + a)} + \frac{1}{-b + a}\right)$$

7. РЕШЕНИЕ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ С ВИЗУАЛИЗАЦИЕЙ РЕЗУЛЬТАТОВ

Графическая библиотека DEtools содержит средства для построения решений обыкновенных дифференциальных уравнений и некоторых типов уравнений в частных производных, включая поля решений и фазовый портрет. Данная библиотека также содержит средства визуализации решений. Подключение библиотеки происходит после выполнения команды:

> with(DEtools):

В состав библиотеки DEtools входят следующие семь функций:

Deplot	– построение решений дифференциальных уравнений и систем;
Deplot1	– построение решения уравнения первого порядка;
Deplot2	– построение решения системы из двух уравнений;
Dchangevar	– подстановка новых переменных в уравнение;
PDEplot	– построение решения квазилинейного уравнения первого порядка в частных производных;
dfieldplot	– построение поля решения;
phaseportrait	– построение фазового портрета.

Функция Deplot

Формат вызова:

DEplot(diffeq, vars, trange, inits, <options>)
DEplot(diffeq, vars, trange, xrange, yrange, <options>)

Параметры:

diffeq	– система дифференциальных уравнений;
vars	– имена переменных;
trange	– область определения независимых переменных;
inits	– начальные условия;
xrange	– область построения первой независимой переменной;

yrange – область построения второй независимой переменной;
options – опции.

Параметр *inits* – множество начальных условий, задаваемых в одной из двух форм:

{[t0,x0,y0], [t1,x1,y1], [t2,x2,y2], ...} или
{[x(t0)=x0, y(t0)=y0], [x(t1)=x1, y(t1)=y1], [x(t2)=x2, y(t2)=y2], ...} .

В случае, когда имеет место система из n уравнений
[t0,x0,x'0,x"0,...x(n)0], неопределенные начальные условия считаются нулевыми. Например, если заданы начальные условия для системы из четырех уравнений в виде [1,2] , то будет использовано значение [1,2,0,0].

Опишем подробнее параметр <options> (перечисленные ниже параметры употребимы для следующих функций из библиотеки: DEplot, DEplot1, DEplot2) .

Все параметры раздела <options> указываются в следующем виде: параметр=значение. Например, области изменения переменных x = c1..d1 или y = c2..d2

stepsize = h
Значение stepsize используется для задания шага в выбранном методе решения поставленной задачи. По умолчанию используется stepsize = (b - a)/20, где a..b– область определения независимой переменной (т. е. параметр trange).

iterations = <integer>
Число шагов итерации между двумя точками. По умолчанию используется iterations равно 1. Эта опция может быть полезной для решения неустойчивых задач.

limitrange = true
Прекратить интегрирование, если интегральные кривые выходят за пределы указанного промежутка для x и y. По умолчанию используется limitrange = false. Если не заданы промежутки для x или y, то этот параметр игнорируется.

scene = [<name>,<name>,<name>]

Указывает вид построения. Например, scene = [x,y] показывает, что будет построен двухмерный график y как функции от x. Если добавить переменную t, то scene = [t,x,y] укажет среде Maple построить трехмерное изображение всех переменных. Параметр также можно использовать для изменения положения осей (например [x,y,t] показывает, что t – вертикальная ось).

method = <scheme>

Определяет метод решения поставленной задачи. Параметр <scheme> может быть: *'euler', 'backeuler', 'impeuler'* или *'rk4'*. По умолчанию используется метод Рунге–Кутта 4-го порядка.

'euler' – метод Эйлера (формула метода: y = y + h*f(t,y));
'backeuler'– формула метода y = y + h*f(t+h,y+h*f(t,y));
'impeuler' – усовершенствованный метод Эйлера (формула метода : y = y + h/2*(f(t,y) + f(t+h,y+h*f(t,y))));
'rk4' – метод Рунге–Кутта 4-го порядка.

Можно также написать свою собственную процедуру, реализующую метод, отличный от перечисленных выше. Для этого нужно указать, что процедура будет являться параметром:

method = My_Integration_Scheme
где My_Integration_Scheme – имя этой процедуры.

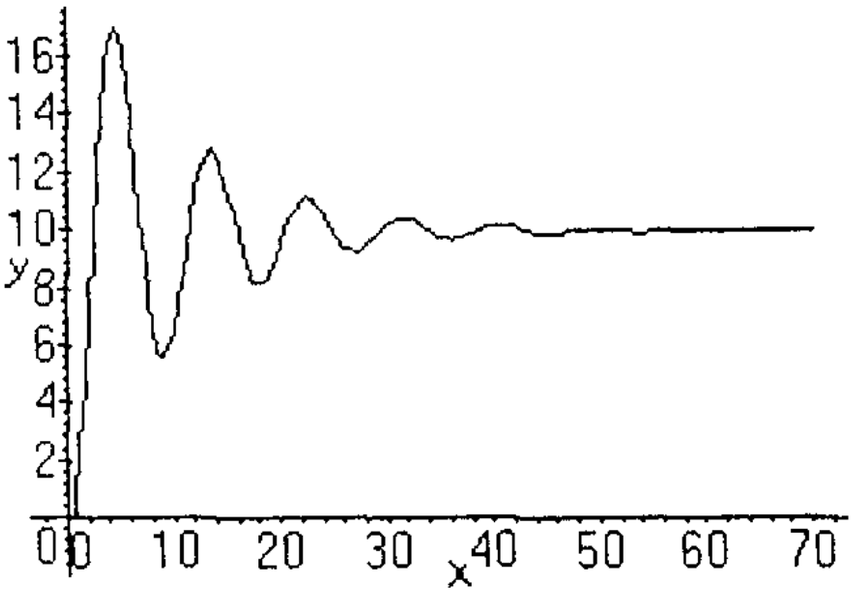
Для примера можно рассмотреть процедуру 'rk4', реализующую метод Рунге–Кутта. Для этого следует выполнить такие команды:

```
readlib(`DEtools/DEplot`):
interface(verboseproc=2):
print(`DEtools/DEplot/rk4`);
```

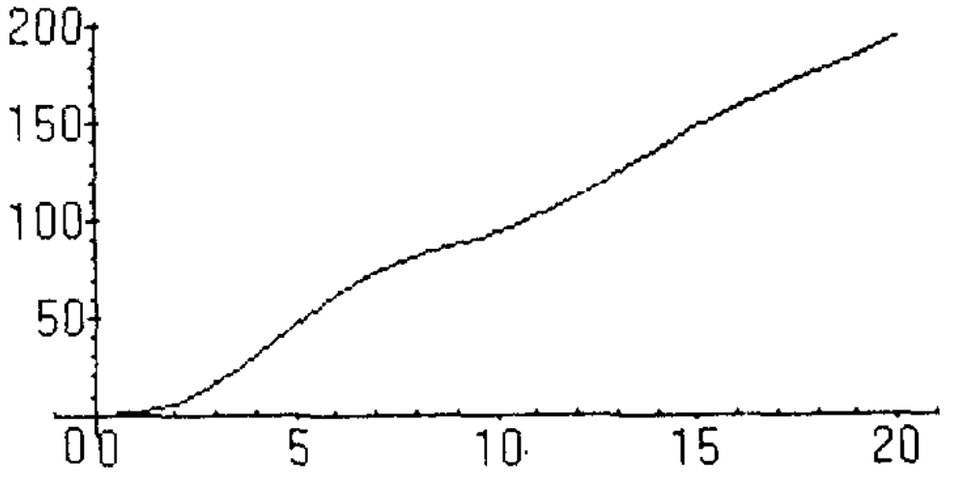
Для заданной системы дифференциальных уравнений первого порядка вида x' = f1(t,x,y), y' = f2(t,x,y), а также для дифференциальных уравнений более высокого порядка вида diff(y(x),x$n) = f(x,y) и множества начальных условий функция DEplot строит кривую решения.

Примеры:

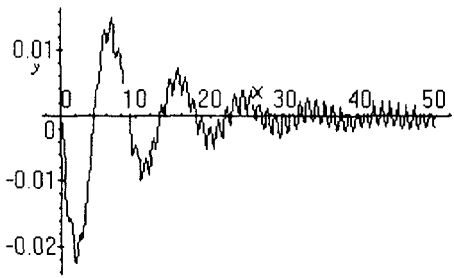
```
> s:=4*diff(y(x),x$2)+0.8*diff(y(x),x)+2*y(x)=20:
> DEplot(s,[x,y],0..70,{[0, – 1,0]},stepsize=.4);
```




```
> s:=4*diff(y(x),x$2)+0.8*diff(y(x),x)+2*y(x)=20*x:
> DEplot(s,[x,y],0..20,{[0,0,0]},stepsize=.4);
```



```
> s:=4*diff(y(x),x$2)+0.8*diff(y(x),x)+2*y(x)=sin(100*x):
> DEplot(s,[x,y],0..50,{[0,0,0]},stepsize=.2);
```



Функция Deplot1

Формат вызова:

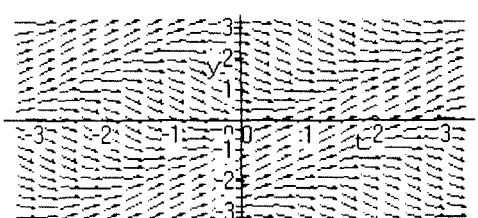
DEplot1(diffeq, vars, trange, inits, <options>)
DEplot1(diffeq, vars, trange, xrange, yrange, <options>)

Указанные в формате вызова параметры аналогичны описанным параметрам функции DEplot.

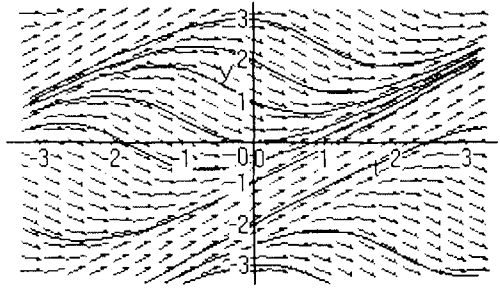
Функция DEplot1 строит график решения для заданного дифференциального уравнения первого порядка вида y' = f(t,y) функция.

Примеры:

```
> DEplot1(diff(y(t),t)=sin(t - y), y(t), t= - Pi..Pi, y= - Pi..Pi);
```



> DEplot1(sin(t – y), [t,y], t= Pi..Pi, {[0,3],[0,2],[0,1],[0,0],[0 , – 2],[0, – 1],[0, – 0.5],[0, – 2.5],[0, – 3]}, y= – Pi..Pi);



Функция DEplot2

Формат вызова:

DEplot2(diffeq, vars, trange, inits, <options>)

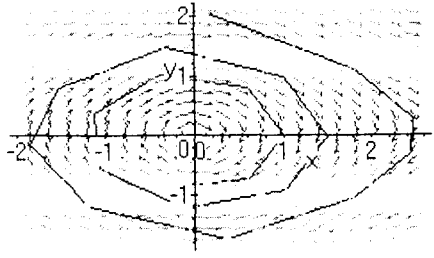
DEplot2(diffeq, vars, trange, xrange, yrange, <options>)

Указанные в формате вызова параметры аналогичны описанным параметрам функции DEplot.

DEplot2 строит кривую решения для двух заданных уравнений в виде x' = f1(t,x,y), y' = f2(t,x,y) и для множества начальных условий.

Пример:

> DEplot2([y, – sin(x) – y/10],[x,y], – 10..10,{[0,1,1]},color = x^2+y^2);



Функция Dchangevar

Функция Dchangevar – переход к другим переменным в дифференциальном уравнении.

Формат вызова:

Dchangevar(eqns, diffeq, <constants>)

Параметры:

eqns	– множество уравнений для подстановки в одно уравнение;
diffeq	– дифференциальное уравнение;
<constants>	– необязательное множество констант в уравнении.

Для заданного уравнения n-го порядка и множества из выражений Dchangevar делает подстановку этого множества в исходное уравнение с вычислением требуемых дифференциалов.

Первый аргумент, *eqns* – множество выражений, содержащее новые значения переменных.

Второй аргумент, *diffeq* – уравнение, в котором требуется провести подстановку. Оно может быть следующего вида:

diff(y(t),t) = f(t,y) или

diff(y(t),t) – f(t,y) = 0.

Примеры:

> Dchangevar({t = T*tau,x(t) = L*y(tau)},diff(x(t),t$3) = tan(t),{T,L});

$$\frac{L\left(\frac{\partial^3}{\partial \tau^3} y(\tau)\right)}{T^3} = \tan(T\tau)$$

```
> a:= diff(x(t),t$2)= - g+R/m*(sqrt(v0^2 -2*g*x(t)) -diff(x(t),t))^2 / sqrt(v0^2
- 2*g*x(t));
```

$$a := \frac{\partial^2}{\partial t^2} x(t) = -g + \frac{R\left(\sqrt{v0^2 - 2\,g\,x(t)} - \left(\frac{\partial}{\partial t} x(t)\right)\right)^2}{m\sqrt{v0^2 - 2\,g\,x(t)}}$$

```
> eqns := {x(t)=L*y(tau),t=T*tau};
```

$$eqns := \{x(t) = L\,y(\tau),\ t = T\,\tau\}$$

```
> cnstnts := {L,T};
```

$$cnstnts := \{T, L\}$$

```
> Dchangevar(eqns,a,cnstnts);
```

$$\frac{L\left(\frac{\partial^2}{\partial \tau^2} y(\tau)\right)}{T^2} = -g + \frac{R\left(\sqrt{v0^2 - 2\,g\,L\,y(\tau)} - \frac{L\left(\frac{\partial}{\partial \tau} y(\tau)\right)}{T}\right)^2}{m\sqrt{v0^2 - 2\,g\,L\,y(\tau)}}$$

Функция PDEplot.

PDEplot(pdiffeq, vars, inits, srange, <options>)

Параметры:

pdiffeq	– квазилинейное уравнение в частных производных первого рода, записанное в определенных терминах;
vars	– имена переменных;
inits	– начальные данные;
srange	– область параметра для начального данного;
<options>	– необязательный аргумент, описываемый ниже;

Нахождение решения уравнения вида

$$P(x,y,u) * D[1](u)(x,y) + Q(x,y,u) * D[2](u)(x,y) = R(x,y\ u),$$

где P, Q и R зависят от x, y и u .

Первый аргумент, *pdiffeq*, должен быть списком, состоящим из P, Q и R. Они могут быть заданы или как выражения, или как функции (операторы). Например, формы [(x,y,z) –› x^2^y, (x,y,u) –› u^3/x, (x,y,u) –› sin(x*y)*u] и [x^2*y, u^3/x, sin(x*y)*u] эквивалентны и образуют дифференциальное уравнение в частных производных:
x^2*y*D[1](u)(x,y) + u^3/x*D[2](u)(x,y) = sin(x*y)*u. В аргументе *pdiffeq* нельзя употреблять имена констант, а можно только имена переменных.

Второй аргумент, *vars*, может быть задан только в следующем виде: [x,y,u] или u(x,y). Это означает, что x и y – имена независимых переменных, а u – имя зависимой переменной.

Третий аргумент, *inits*, должен быть списком, состоящим из трех элементов, которые определяют параметрическую форму трехмерной кривой в декартовых координатах. Элементы списка могут быть или выражениями, или функциями; если выражения, то они должны содержать одно символьное имя (параметр), а если функции – то только от одного переменного.

Четвертый аргумент, *srange*, должен быть областью изменения параметра начального условия. Указывается в виде s = a..b, или просто a..b.

Пятый аргумент необязательный, он может состоять из следующих параметров:

x = c1..d1
y = c2..d2 – области изменения аргументов x и y;

stepsize = h – шаг, используемый в данном методе интегрирования. По умолчанию stepsize = 0.2;

numsteps = <integer> – число шагов интегрирования. Знак перед числом показывает, что шаги будут выполнены вперед

(знак `+`) или назад (знак `–`);

numsteps = [<integer>, <integer>] – список показывает, что интегрирование ведется в двух направлениях. Первое число – шаги назад, второе – вперед;

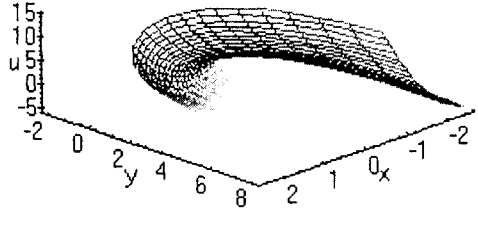
iterations = <integer> – число шагов интегрирования между двумя соседними построенными точками. По умолчанию этот аргумент равен единице;

title = <string> – выводит заголовок графика;

limitrange = true – остановить интегрирование, если результат вышел за область определения для x и y.

Пример:

> PDEplot([1,2*x,y],[x,y,u],[0,s,1+s^2],s= – 2..2,numchar=20);



Функция dfieldplot

Функция dfieldplot предназначена для построение поля решения системы дифференциальных уравнений.

Формат вызова:

dfieldplot(diffeq, vars, trange, <options>)

Параметры:

diffeq	– уравнение или система дифференциальных уравнений;
vars	– имена переменных;
trange	– область изменения независимой переменной;
<options>	– аналогичны аргументам для функции Deplot.

Первый аргумент должен содержать систему из одного или двух дифференциальных уравнений и может быть записан в одной из двух эквивалентных форм:

```
diff(y(t),t) = f(t,y)
diff(y(t),t) – f(t,y) = 0
```

или для системы из двух дифференциальных уравнений:

```
[diff(x(t),t) = f1(t,x,y), diff(y(t),t) = f2(t,x,y)]
[diff(x(t),t) – f1(t,x,y) = 0, diff(y(t),t) – f2(t,x,y) = 0]
[diff(x(t),t) – f1(t,x,y), diff(y(t),t) – f2(t,x,y)]
```

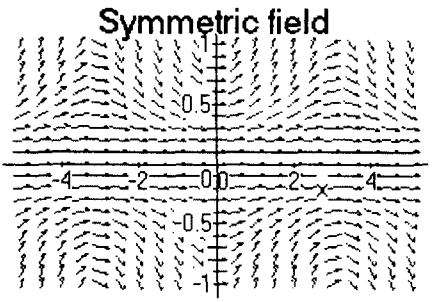
Нельзя употреблять имена констант; t, x и y – имена, употребляемые для аргумента diffeq, где t, x и y описаны во втором аргументе – vars.

Второй аргумент, vars, указывает имена переменных, используемых в дифференциальных уравнениях, причем следует отделять зависимые и независимые переменные. Для автономной системы имена переменных могут быть заданы в следующем виде: [x,y]. Если система не автономная, аргумент vars должен быть записан следующим образом: [t,x,y] или [x(t),y(t)].

Третий аргумент, trange, может быть записан двумя способами: a..b или t = a..b, где t – независимая переменная.

Пример:

> dfieldplot(y^2*sin(x),[x,y], – 5..5,title=`Symmetric field`);



Функция phaseportrait

Функция phaseportrait – построение фазового портрета (интегральной кривой) для системы дифференциальных уравнений.

Формат вызова:

phaseportrait(diffeq, vars, trange, inits, <options>)

Параметры:

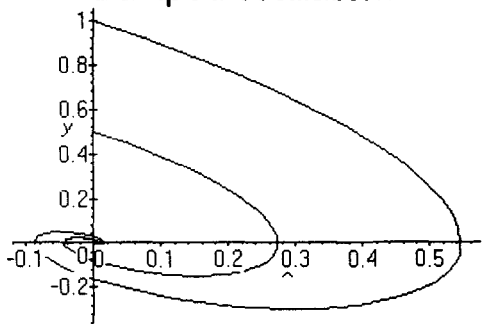
diffeq	– уравнение или система из двух уравнений;
vars	– имена переменных;
trange	– область изменения независимой переменной;
inits	– начальные условия;
<options>	– неоднократно описано выше.

Для заданных одного или двух дифференциальных уравнений в виде x' = f1(t,x,y), y' = f2(t,x,y) и для множества начальных условий phaseportrait строит интегральную кривую для данного уравнения согласно заданным начальным условиям.

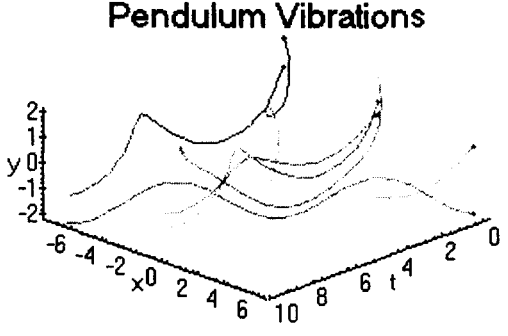
Задание аргументов vars и trange аналогично функции dfieldplot.

Примеры:

>phaseportrait([y,–x–y],[t,x,y],0..10,{[0,0,1],[0,0,.5]},scene=[x,y], stepsize=.1, title=`Damped oscillations`);



> phaseportrait([y, –sin(x)],[t,x,y],0..10,{[0,0,.5],[0,0,1],[0,0,1.8],[0, –2*Pi,1],[0,2*Pi,.5],[0, –2*Pi,2.1],[0,2*Pi, –2.1]}, stepsize=.2,title=`Pendulum Vibrations`);



8. РЕШЕНИЕ ЗАДАЧ ИЗ ТЕОРИИ ГРАФОВ С ВИЗУАЛИЗАЦИЕЙ РЕЗУЛЬТАТОВ

Maple включает в свой состав библиотеку для работы с графами. Имя этой библиотеки networks. Подключается эта библиотека следующей командой:

> with(networks):

Список функций данной библиотеки включает 75 команд и процедур, некоторые из них будут описаны далее.

Граф в Maple представляется процедурой, которая имеет тип GRAPH. Создается граф с помощью команды new( ); можно также использовать команды *complete( ), cycle( ), petersen( ).*

Команда new

Формат команды new следующий:

new(G);
G:=new( ); (G – имя графа)

> new(G):

Команда new создает пустой граф – граф, в котором нет ни ребер, ни узлов.

Узлы графа можно просмотреть с помощью команды vertices:

> vertices(G);

{ }

Ребра графа можно просмотреть с помощью команды edges:

> edges(G);

{ }

Формат команды edges имеет три вида:

edges(G);
edges(P,G);
edges(P,G,'all');

где G – граф, P – один из следующих вариантов: {u,v} или [u,v].

Команда первого вида возвращает в форме множества все имена ребер графа.

Команда второго вида используется, когда надо просмотреть только определенные имена ребер.

Если P – это {u,v}, то возвращаются имена всех ребер между u и v (вне зависимости от их направленности). Если P – это [u,v] , то возвращаются имена ребер, направленые от u к v (u–хвост, v–голова).

> new(i):

Добавим в граф три вершины:

> addvertex(1,2,3,i);

2, 3, 1

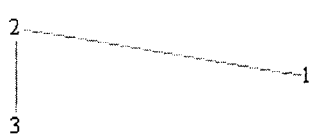
Добавим два направленных ребра 1–2 и 2–3:

> addedge([1,2],i); addedge([2,3],i);

e1

e2

> draw(i);



Просмотрим все ребра:

> edges(i);

{*e1*, *e2*}

Просмотрим ребра, направленные от вершины 3 к вершине 2:

> edges([3,2],i);

{ }

А теперь просмотрим ребра, направленные от вершины 2 к вершине 3:

> edges([2,3],i);

{*e2*}

Следующая команда просматривает все ребра между вершинами 2 и 3:

> edges([3,2],i,`all`);

{*e2*}

Команда complete

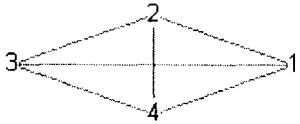
Complete создает полный граф. В полном графе каждая вершина одной части графа соединена с каждой вершиной другой части графа.

Формат команды следующий:

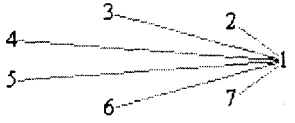
complete(n);
complete(m,n);
complete(m1,...,mk);
complete(vset);

где n, m – число узлов в отдельной части;
m1, ... ,mk – целое число, определяющее количество узлов в каждой части;
vset – множество имен узлов.
Число аргументов определяет число частей графа.

> g:=complete(4): draw(g);



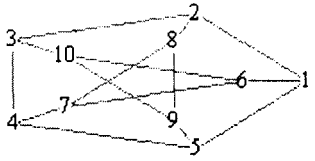
> g1:=complete(1,6): draw(g1);



Команда petersen

Эта команда создает граф Петерсена.

> h:=petersen( ): draw(h);



Команда draw

Эта команда рисует граф.

Формат команды:

draw(G);
draw(Concentric(L),G);
draw(Linear(L),G);

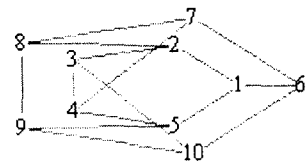
где G – имя графа; L – список вершин.

Первый вариант команды рисует граф, в котором все вершины находятся на равном расстоянии друг от друга и соединены прямыми линиями. Причем располагаются вершины на окружности.

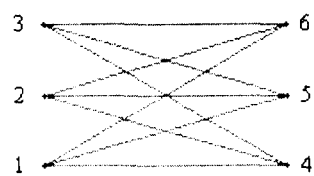
Во втором случае вершины располагаются на концентрических окружностях.

В третьем случае случае команды вершины графа располагаются на линии.

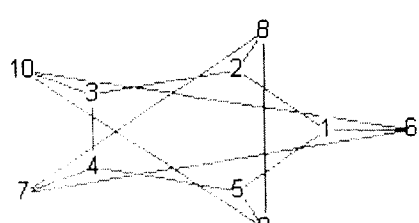
> G:=petersen( ): draw(Concentric([1,2,3,4,5]),G);



> draw(Linear([1,2,3]),complete(3,3));



> draw(Concentric([1,2,3,4,5],[6,8,10,7,9]),G);



Команда ends

Эта команда возвращает имена вершин ребра.
Формат команды:

ends(G) – возвращает имена всех вершин графа G;
ends(e, G) – возвращает имена вершин ребра e.

> new(w): addvertex(a,b,c,d,w);

$$a, c, b, d$$

Добавим в граф два ребра: a–b (направленное) и c–b (ненаправленное):

> addedge([a,b],w): addedge({b,c},w):

> ends(w);

$$\{\{c, b\}, [a, b]\}$$

> edges(w);

$$\{e2, e1\}$$

> ends(e1,w);

$$[a, b]$$

Результат последней команды показывает, что ребро e1 – это направленное ребро a–b. Первый элемент списка – хвост ребра, второй элемент – голова ребра.

В Maple существует правило обозначения ребер графа: ненаправленное ребро обозначается в виде множества, а направленное – в виде списка (первый элемент списка – хвост ребра, второй элемент – голова).

Команда addvertex

Эта команда добавляет вершину или множество вершин в граф.

Формат команды:

addvertex(v1,G);
addvertex(v1,v2,G);
addvertex(v1,weights=w,G);
addvertex([v1,v2],weights=[w1,w2],G);

где v1, v2 – имена вершин графа G;
w – вес вершины (по умолчанию 0).

> new(j): addvertex({a1,a2,a3,a4},j);

a1, a2, a4, a3

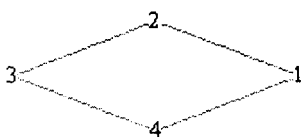
Команда addedge

Эта команда добавляет в граф новое ребро.
Формат команды:

```
addedge({v1,v2},G);
addedge([v1,v2],G);
addedge({v1,v2},names=edg1,weights=w,G);
addedge(Cycle(v1,...,vn),G);
addedge(Path(v1,...,vn),G);
```

где
- v1, v2 – имена вершин графа G;
- edg1 – имя, присваемое ребру (по умолчанию e1, e2 и т.д.);
- w – вес ребра (по умолчанию 1);
- Path – определение пути через ребра v1, ... ,vn;
- Cycle – определение петли через ребра v1,...,vn;

> new(G): addvertex({1,2,3,4},G): addedge(Cycle(1,2,3,4),G): draw(G);



Если надо добавить несколько ребер, то они должны быть переданы в команду addedge в форме списка или множества. В случае списка ребрам можно присвоить имена и веса, которые также должны быть представлены в форме списка.

```
> new(g):
> addvertex({a,b,c},g):
> addedge([{a,b},{a,c}],names=[way1,way2], weights=[2,6],g);
```

$$way1, way2$$

Команда delete

Эта команда позволяет удалять из графа ребра и вершины.

Формат команды:
delete(Eset,G);
delete(Vset,G);

где G – имя графа; Vset, Eset – имена множеств, содержащих вершины или ребра, предназначенные для удаления.

```
> G:=complete(5):
> delete({3,4},G):
> vertices(G), ends(G);
```

$$\{1, 2, 5\}, \{\{1, 5\}, \{2, 5\}, \{1, 2\}\}$$

В предыдущих командах было удалено ребро {3,4}.
В следующих командах удаляются ребра, инцидентные вершине 5.

```
> delete(incident(5,G),G):
> vertices(G), ends(G);
```

$$\{1, 2, 5\}, \{\{1, 2\}\}$$

Команды head и tail

Эти команды возвращают имена вершин, которые являются головой ребер (в случае команды head) или хвостом (в случае команды tail).

Формат команд:

tail(G);
tail(e,G);
tail(eset,G);

где G – граф, e – ребро, eset – множество ребер.

Для команды head формат вызова аналогичный.

> new(k):

> addvertex({a,b},k);

$$a, b$$

> addedge([a,b],k);

$$e1$$

> tail(e1,k);

$$a$$

> head(e1,k);

$$b$$

> addedge([b,a],k);

$$e2$$

> head(k);

table([
$e1 = b$
$e2 = a$
])

Последняя команда вывела все вершины–головы графа.

Команда void

Эта команда генерирует пустой граф, т.е. граф без ребер.

Формат команды:

void(n);
void(vset);

где n – число, определяющее количество вершин;
vset – множество имен вершин.

```
> g:=void(10):
> edges(g);
```

{ }

```
> vertices(g);
```

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Команда show

Эта команда возвращает таблицу, содержащую всю информацию о графе.

```
> G:=complete(3):
> T:=show(G);
```

```
T := TABLE([
_Emaxname = 3
_Countcuts = _Countcuts
_Neighbors = TABLE([
1 = {2, 3}
2 = {1, 3}
3 = {1, 2}])
_Vweight = TABLE(sparse,[
])
```

```
_Econnectivity = _Econnectivity
_Vertices = {1, 2, 3}
_EdgeIndex = TABLE(symmetric,[
(1, 2) = {e1}
(2, 3) = {e3}
(1, 3) = {e2}])
_Edges = {e2, e1, e3}
_Head = TABLE([
])
_Eweight = TABLE([
e1 = 1
e2 = 1
e3 = 1
])
_Tail = TABLE([
])
_Bicomponents = _Bicomponents
_Ends = TABLE([
e1 = {1, 2}
e2 = {1, 3}
e3 = {2, 3}
])
_Counttrees = _Counttrees
_Status = {SIMPLE, COMPLETE}])
```

Команда incidence

Эта команда генерирует матрицу инцидентности.

```
> G:=cycle(4):
> addedge([1,3],G):
> T:=incidence(G);
```

$$T := \begin{bmatrix} 1 & 0 & 0 & 1 & -1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Команда adjacency

Эта команда генерирует матрицу смежности.

```
> with(networks):
> G:=cycle(4):
> addedge([1,3],G):
> T:=adjacency(G);
```

$$T := \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Команды vweight и eweight

Эти команды возвращают веса вершин и ребер соответственно.

Формат команды vweight:

vweight(G);
vweight(v,G);
vweight(vlist,G);

где G – граф; v – имя вершины; vlist – список вершин.

Формат команды *eweight*:

eweight(G);
eweight(e,G);
eweight(elist,G);

```
> new(G):
> addvertex([1,2,3],weights=[1,2,3],G):
> vweight(G);
```

```
table(sparse, [
        1 = 1
        2 = 2
        3 = 3
        ])
```

```
> vweight(1,G);
```

1

```
> new(G):
> addvertex(1,2,3,G):
> addedge([{1,2},{2,3},{1,3}],weights=[3,4,6],G);
```

e1, *e2*, *e3*

```
> eweight(G);
```

table([
e1 = 3
e2 = 4
e3 = 6
])

```
> eweight(e1,G);
```

3

Команда duplicate

С помощью этой команды можно создать копию графа.

```
> G:=complete(4):
> H:=duplicate(G):
```

Последние команды создали два идентичных графа: G и H.

Команда random

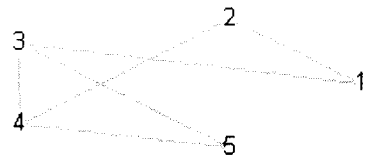
Эта команда создает случайный граф.
Формат команды:
G:=random(n);
G:=random(n,m);

G:=random(n,'prob'=x);

где n – число вершин; m – число ребер; x – действительное число 0 < x <1.

Если используется первый вариант команды, то создается граф с n вершинами, а ребра определяются с вероятностью 50 % (т.е. ребро между данными вершинами выбирается случайным образом).

> h:=random(5): draw(h);



При вызове random с несколькими аргументами строится случайный граф с n вершинами и с m ребрами, причем вероятность построения ребра определяется числом x.

> G:=random(4,prob=1): draw(G);

В теории графов широко распространены задачи отыскания наибольшего потока через заданную сеть. Подобные задачи решаются не только для определения потока, но и для определения максимального паросочетания. Для решения таких задач обычно используется алгоритм Форда – Фалкерсона.

Maple также предоставляет возможность решения задачи отыскания максимального потока в сети. Эту возможность реализует функция *flow*.

Формат команды flow:

flow(G,s,t);
flow(G,s,t,'maxflow'=n);
flow(G,s,t,eset,comp);
flow(G,s,t,eset,comp,'maxflow'=n);

Эта команда находит максимальный поток в сети G от вершины s к вершине t (от источника к стоку).

eset – имя множества, в котором будут содержаться ребра, по которым проходит максимальный поток.

comp – имя множества, в котором будут содержаться имена вершин, через которые проходит поток.

Параметр maxflow определяет, что необходимо найти поток n. Если n больше максимально возможного потока, то возвращается значение максимально возможного потока.

В следующем примере создается полный граф,у которого вес всех ребер равен еденице. Затем находится максимальный поток от вершины 1 к вершине 2.

```
> G:=complete(3,2): flow(G,1,2,eset,comp);
```

$$2$$

```
> eset;
```

$$\{\{1,4\},\{1,5\},\{2,4\},\{2,5\}\}$$

```
> comp;
```

$$\{1,2,5\}$$

```
> flow(G,1,2,set1,set2,'maxflow'=1);
```

$$1$$

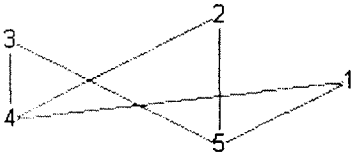
```
> set1;
```

$$\{\{1,4\},\{2,4\}\}$$

Последняя команда flow выдала 1 – поток, который требовался опцией maxflow (хотя максимальный поток в данной сети равен 2).

Граф, в котором находился поток, показан ниже:

> draw(G);



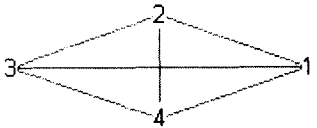
Широко распространены задачи нахождения кратчайшего пути. В таких задачах задаются граф (сеть дорог) и начальная вершина (пункт отправления). Каждому ребру можно присвоить вес – длину дороги; кроме этого ребра могут быть как ориентированными, так и не ориентированными. Задача нахождения кратчайшего пути решается с помощью алгоритма Дейкстры. Результат работы алгоритма – дерево с началом в начальной вершине, причем ко всем остальным вершинам идут кратчайшие пути. В Maple задачу нахождения кратчайшего пути можно решить с помощью команды *shortpathtree*.

Формат команды:

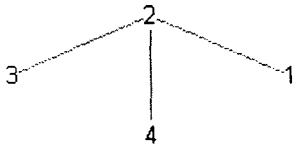
shortpathtree(G,v),
где G – граф; v – начальная вершина.

В следующем примере создадим полный граф с четырьмя вершинами, причем вес каждого ребра равен единице.

> g:=complete(4): draw(g);



> g1:=shortpathtree(g,2): draw(g1);



Таким образом, g1 – дерево, полученное в результате работы алгоритма Дейкстры.

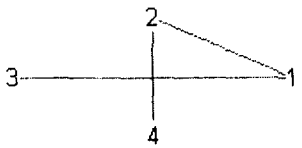
Теперь изменим исходный граф g. Ребру, которое соединяет вершины 2 и 3, присвоим вес, равный 100.

> edges({2,3},g);

$$\{e4\}$$

> delete(e4,g): addedge({2,3},weights=100,g):

> t:=shortpathtree(g,2): draw(t);



Таким образом, в измененном графе найдены новые кратчайшие пути ко всем вершинам.

Команда shortpathtree также присваивает длины кратчайших путей весам вершин.

> vweight(t);

table(*sparse*, [
1 = 1
2 = 0
3 = 2
4 = 1
])

Из приведенной таблицы следует, что, для того чтобы попасть из вершины 2 в вершину 3, требуется пройти расстояние, равное двум.

В теории графов существует понятие *планарности* графа. Граф называется планарным, если его можно изобразить на плоскости без самопересечений. Задача определения планарности встречается при разводке печатных плат, где ребра графа – печатные проводники, вершины – контактные площадки.

В Maple V проверить планарность графа можно с помощью команды *isplanar*, которая возвращает true, если граф планарный, и false – в противном случае. Вначале *isplanar* упрощает граф, т.е. удаляет циклы и повторяющиеся ребра, а затем граф проверяется на планарность (упростить граф можно с помощью gsimp).

```
> g:=complete(5): draw(g);
> isplanar(g);
```

false

```
> g1:=complete(3): draw(g1);
> isplanar(g1);
```

true

В этом разделе приведены лишь краткие сведения о библиотеке Networks, но перечисленных команд будет достаточно для решения элементарных задач теории графов. При необходимости найти дополнительные сведения о данной библиотеке можно воспользоваться справочной системой Maple.

9. ПОСТРОЕНИЕ ГРАФИКОВ ПО РЕЗУЛЬТАТАМ МАТЕМАТИЧЕСКИХ ВЫЧИСЛЕНИЙ

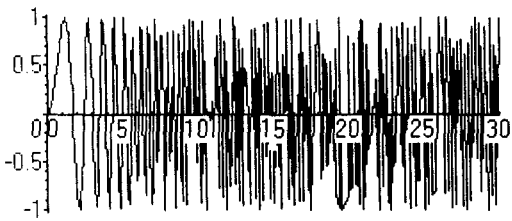
9.1. Общие сведения

Maple является прекрасным инструментом для визуализации информации о исследуемой функции. Используя устройство типа "мышь", можно просмотреть локальные минимумы и максимумы, нулевые точки.

9.1.1. Ограничения

Если функция имеет большой интервал между двумя точками (превышающий масштаб осей), то график не может быть выведен. Например:

> plot (sin(x^2),x=0..30);



В Windows – версии Maple – не рекомендуется устанавливать командой interface устройство вывода, так как не удастся восстановить вывод в окно Windows.

9.1.2. Устройства вывода

Установить устройство вывода можно с помощью команды
interface(plotdevice = x),
где x – параметр.

Параметр может принимать следующие значения:

bmp, canon, cps (или postscriptc), deskjet, epson24, epson9, epson9hi, hpgl, gif, i300, ibm, ibm_mono, ibmpro, ibmquiet, laserjet, ln03, oki92, paintjet, pcx, pic, ps, (или postscript), tiff, toshiba, unix

Если в качестве устройства вывода установлен один из типов принтеров, то вывод производится на терминал, указанный в команде *interface(plotoutput)*, т. е. на PRN, LPT1, LPT2, COM1 или COM2.

В противном случае вывод производится в файл, указанный в команде *interface(plotoutput)*.

Если устройство – *char*, то вывод в зависимости от установки plotoutput осуществляется на следующие устройства:

- на экран, если plotoutput=ibm;
- на принтер, если *plotoutput=PRN, LPT1, LPT2, COM1* или COM2;
- в файл (plotoutput=имя файла).

Режим *char* не рекомендуется для вывода 3D-графиков.

> interface(plotdevice=epson9);

> interface(plotoutput=PRN);

Эти команды установили режим вывода на девятиигольчатый EPSON–совместимый принтер.

> interface(plotdevice=ibm);

Последняя команда восстанавливает вывод на экран. В Windows – версии Maple – эта команда не работает!

9.1.3. Терминальные установки

Многие терминалы и терминальные программы работают обычно в текстовом режиме, но могут быть переведены в графический режим. Maple имеет две интерфейсные переменные: preplot и postplot, которые могут быть использованы для переключения текстового режима в графический.

Например, команда interface(preplot=[27,91,63,51,56,104]); генерирует стринговскую последовательность перед каждым обращением к plot.

Postplot посылает последовательность после plot для возвращения в исходный режим. Терминал вывода устанавливается командой *interface(plotoutput=x)*, где x – параметр: PRN, LPT1, LPT2, COM1 или COM2 (имена портов).

```
> interface(plotoutput=PRN);
```

Эта команда устанавливает вывод на терминал принтера. При этом plotdevice должен быть установлен на определенный тип принтера.

9.1.4. Твердая копия

Твердую копию (распечатку) можно получить двумя способами:

- через установку командой interface устройства вывода;
- выбором в меню окна Plot for Maple соответствующего пункта.

При необходимости копия экрана может быть сохранена в файле:

```
> interface(plotdevice=gif,plotoutput=`my.gif`);
> plot(sin(y),y=0..Pi);
```

Построенное изображение вывелось в файл my.gif в графическом формате GIF.

9.2. *Построение графиков 2D*

9.2.1. Задание областей

Область – это окно декартовой системы координат, в котором строится график.

Синтаксис определения области:

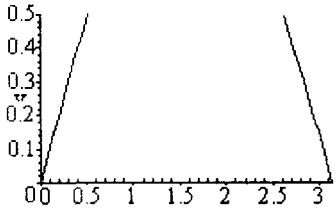
x=*нижняя граница..верхняя граница.*

Числа, определяющие границы, должны быть действительными.

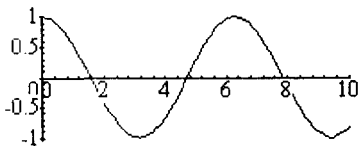
plot(f,x=low..hi,y=low..hi) – пример задания.

Области можно задавать с использованием констант. Например: infinity, Pi, exp(8) и т. д. По умолчанию выбирается диапазон -10..10 для оси абсцисс. Если указан один диапазон, то считается, что он для оси абсцисс, а для оси ординат область изменения выбирается автоматически.

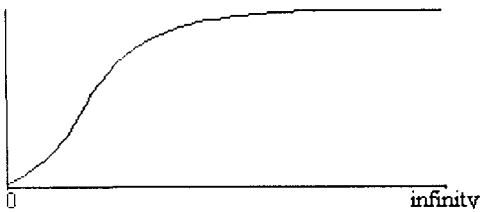
> plot(sin(x),x=0..Pi,y=0..0.5);



> plot(cos,0..10);



> plot(exp,0..infinity);



Maple выполняет преобразования (–infinity,infinity) в (–1,1), используя аппроксимацию через *arctan*.

9.2.2. Стили

При построении графиков можно выбрать стиль (тип) интерполирования. Задается стиль с помощью ключевого слова *style*:

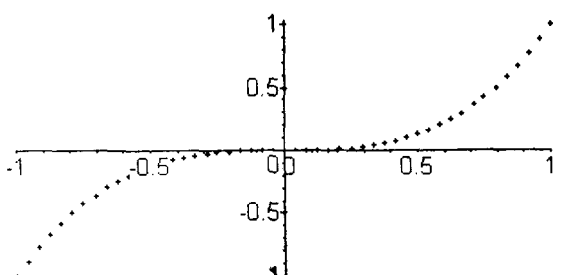
plot(f,h,v,style=x).

Существуют три стиля:

POINT – построение по точкам;
LINE – линейная интерполяция;
PATCH – стиль для многоугольников.

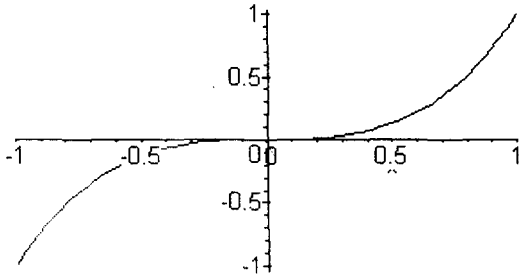
Стиль point – график будет строиться по точкам. Точки могут быть заданы парами в виде списка: [x1, y1, x2, y2, ..., xn, yn] или [[x1, y1], [x2, y2], ..., [xn, yn]]}

> plot(x^3,x= – 1..1,style=point);



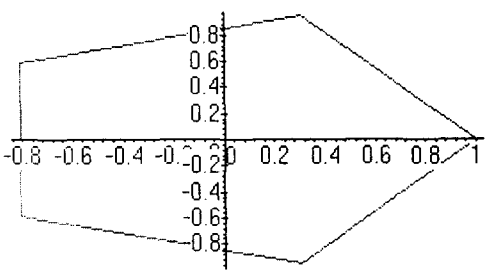
Стиль line – точки графика будут соединяться прямыми. Данный стиль выбирается по умолчанию.

> plot(x^3,x= – 1..1,style=line);



Стиль patch – будут раскрашиваться многоугольники.

> plot([seq([cos(2*Pi*i/5), sin(2*Pi*i/5)], i = 1..5),[cos(2*Pi/5), sin(2*Pi/5)]], style=patch, color=red);



9.2.3. Параметры

Параметры перечисляются в команде plot после указания областей в форме

< имя параметра > = <значение>.

Параметры могут быть следующими:

scaling – управляет масштабированием ((CONSTRAINED (сдавленный) или

UNCONSTRAINED (несдавленный)). По умолчанию – UNCONSTRAINED;

axes – определяет тип координатных осей: FRAME, BOXED, NORMAL, NONE;

coords=polar – показывает, что построение ведется в полярных координатах. Первый параметр – радиус, второй – угол;

numpoints=n – определяет минимальное количество точек (по умолчанию n = 49).
Примечание: построение ведется по адаптивному алгоритму, поэтому обычно генерируются большее количество точек, чем указано в numpoints;

resolution=n – устанавливает горизонтальное разрешение устройства вывода (по умолчанию n = 200). Этот параметр используется, когда отключен адаптивный алгоритм построения;

color=n – определяет цвет построения;

xtickmarks=n – определяет, что на оси X должно быть показано не меньше n отметок значений координат. По оси Y действует команда ytickmarks=n;

style=s – определяет режим интерполяции (POINT, LINE, PATCH);

discont=s – только для построения выражений.
s = {true,false}. Определяет промежутки непрерывности выражения и разбивает ось X на промежутки, где выражение непрерывно;

title=t – определяет заголовок чертежа, t = string;

thickness=n – определяет толщину линий, n = 0, 1, 2, 3 (0 – по умолчанию);

linestyle=n – стиль линии. 0 и 1 – сплошная линия, n > 1 – различные шаблоны заполнения точками;

symbol=s – символ для точек чертежа, s – это одно из выражений: BOX, CROSS,CIRCLE, POINT, DIAMOND;

font=l – шрифт для текстовых объектов чертежа. l – список [family, style, size], где family – это одно из выражений TIMES, COURIER, HELVETICA, SYMBOL. Для TIMES стиль может быть одним из выражений: ROMAN, BOLD, ITALIC, BOLDITALIC. Для HELVETICA и COURIER стиль может быть выбран из BOLD, OBLIQUE или BOLDOBLIQUE. Для SYMBOL стиль не указывается;

Size – определяет размер точек шрифта;

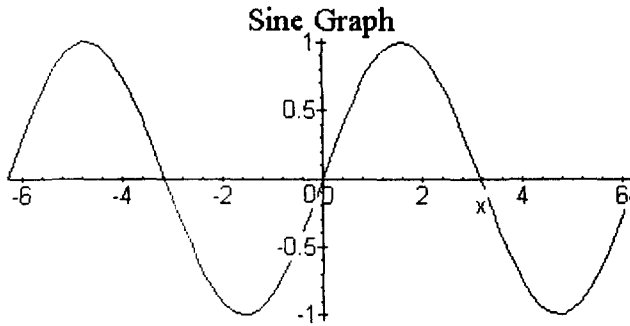
titlefont=l – определяет шрифт для заголовка (так же как и для font);

axesfont=l – определяет шрифт для координатных меток осей координат (так же как и для font);

labelfont=l – определяет шрифт для меток (labels) на осях координат (так же как и для font);

view=[A, B] – определяет максимальные и минимальные координаты, в пределах которых график будет отображаться на экране. A = [xmin..xmax], B=[ymin..ymax]. По умолчанию отображается вся кривая.

> plot(sin(x),x= – 2*Pi..2*Pi,title=`Sine Graph`,color=blue);

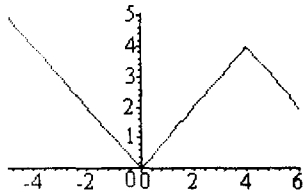


9.2.4. Кусочные функции

Для построения кусочной функции надо просто определить процедуру, описывающую кусочную функцию, а затем как обычно воспользоваться командой plot.

> w:=proc(x) if x<0 then – x elif (x>0) and (x<4) then x else – x+8 fi end:

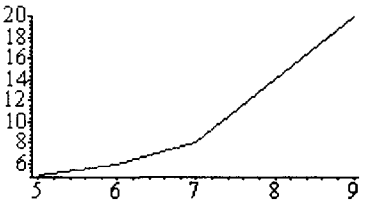
> plot(w, – 5..6,color=red);



9.2.5. Построение по данным

В этом случае необходимо определить набор данных – точки графика.

> plot([5,5,6,6,7,8,9,20],color=blue);



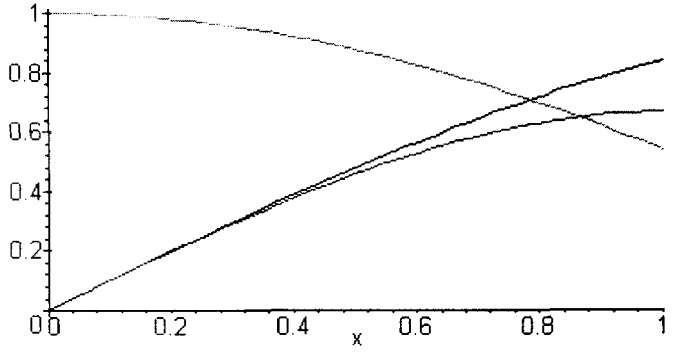
> plot([[5,5],[6,6],[7,8],[9,20]]):

Последняя команда эквивалентна предыдущей.

9.2.6. Совмещение графиков

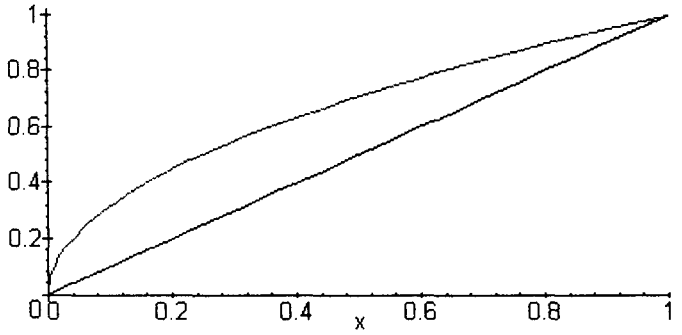
Можно построить несколько графиков на одной координатной плоскости. При таком построении Maple автоматически выбирает разные цвета для графиков функций.

> plot({x – x^3/3,sin(x),cos(x)},x=0..1);



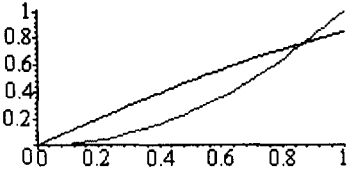
Допустимо совмещать обычную и параметрическую графику.

> plot({x,[x^2,x,x=0..1]},x=0..1);



На одном графике могут быть представлены функции, заданные через функциональные операторы и встроенные функции. Например:

> plot({sin,x – >x^2},0..1);

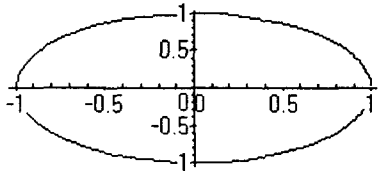


9.2.7. Параметрическая графика

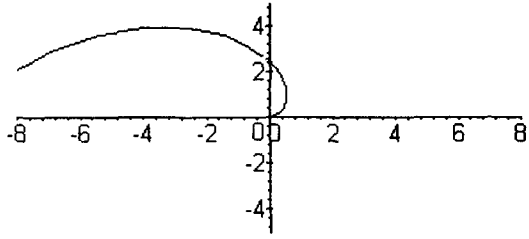
При построении параметрических функций используется следующий синтаксис команды plot:

plot([x(t),y(t),t=(диапазон изменения t)],h,v,options)

> plot([(t^2 – 1)/(t^2+1),2*t/(t^2+1), t= – infinity..infinity]);



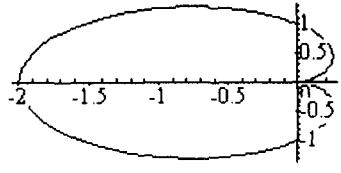
> plot([x^2,x,x=0..3*Pi], – 8..8, – 5..5,coords=polar);



9.2.8. Построение в полярных координатах

При построении функции в полярных координатах используют параметр coords=polar. Формат команды plot в этом случае такой:

> plot ([1 – cos(t),t,t=0..2*Pi],coords=polar);

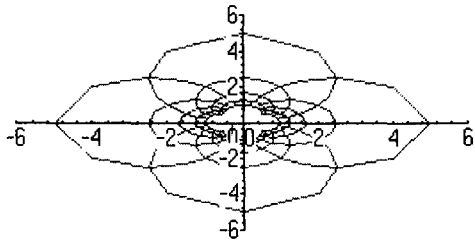


9.2.9. Дополнительные возможности

Перед построением графики в комплексной плоскости следует подключить библиотеку plots.

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

```
> conformal(1/z,z= – 1 – I..1+I, – 6 – 6*I..6+6*I, color=magenta);
```



В Maple возможна анимация двумерных графиков.

```
> with(plots):
> animate( {x – x^3/u,sin(u*x)}, x=0..Pi/2,u=1..16 ,color= red);
```

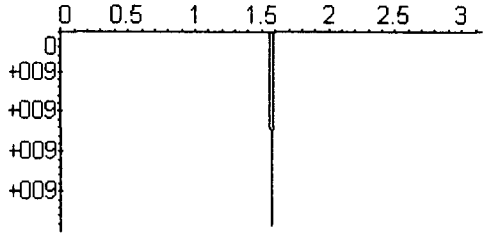
Синтаксис команды animate:
animate(F, x, t,...).
Здесь F=F(x,t) – функция двух переменных; x, t – диапазоны изменения x и t.

При анимации происходит следующее: изменяются значения t и при фиксированных значениях t строится график F(x,t). Количество выводимых кадров можно устанавливать параметром frames (по умолчанию frames =16).

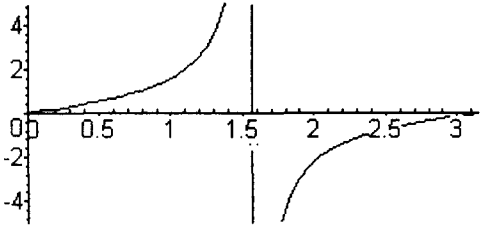
```
> animate(t*x,x=0..1,t=0..100,frames=30);
```

При изменении параметров рисования графика (не самой функции!) график можно перерисовать командой replot.

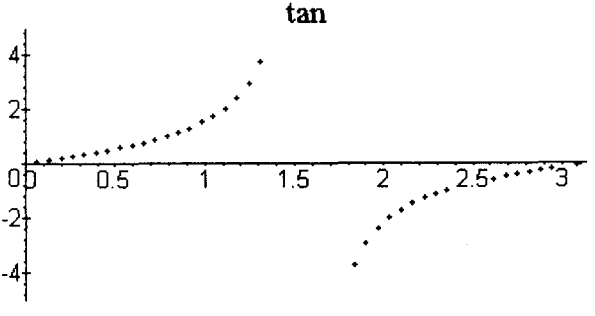
```
> with(plots):
> plot(tan(x),x=0..Pi);
```




```
> replot(",view=[0..Pi, – 5..5]);
```




```
> replot(",style=POINT,title=tan,scaling=unconstrained);
```



9.3. Построение графиков 3D

9.3.1. Описание функций для построения

Для построения графиков 3D используется функция *plot3d.*

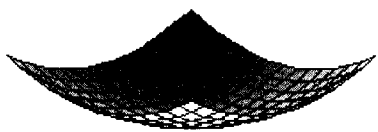
Синтаксис:

plot3d(expr1, x=a..b, y=c..d)
plot3d(f, a..b, c..d)
plot3d([f,g,h], a..b, c..d)

f,g,h	– отображаемые функции (или функция);
expr1	– выражение, зависящее от x и y (функция двух переменных);
exprf,exprg,exprh	– выражения, зависящие от s и t;
a,b	– действительные константы;
c,d	– действительные константы, процедуры или выражения (зависящие от x);
x,y	– имена.

Построить трехмерный график в Maple достаточно просто.

```
> plot3d(x^2+y^2,x= – 1..1,y= – 1..1);
```



Можно построить несколько графиков в одной системе координат.

```
> c1:= [cos(x) – 2*cos(0.4*y),sin(x) – 2*sin(0.4*y),y]:
> c2:= [cos(x)+2*cos(0.4*y),sin(x)+2*sin(0.4*y),y]:
> c3:= [cos(x)+2*sin(0.4*y),sin(x) – 2*cos(0.4*y),y]:
> c4:= [cos(x) – 2*sin(0.4*y),sin(x)+2*cos(0.4*y),y]:
> plot3d( {c1,c2,c3,c4}, x=0..2*Pi, y=0..10, grid=[25,15], style=patch, color=sin(x));
```

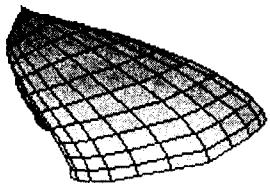
9.3.2 Параметрическое построение

Приведем синтаксис команды *plot3d* для построения параметрических поверхностей:

plot3d([expr1,expr2,expr3],s=a..b,t=c..d).

Здесь expr1, expr2, expr3 – функции, зависящие от s и t;
x(t,s)=expr1, y(t,s)=expr2, z(t,s)=expr3

> plot3d([s*sin(s)*cos(t),s*cos(t)*cos(t),s*sin(t)],s=0..2*Pi,t=0..Pi);



9.3.3. Стили

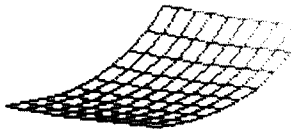
Стили определяют, каким образом будет рисоваться график, и устанавливаются параметром

style=s,

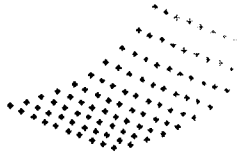
где s – одно из выражений: POINT, HIDDEN, PATCH, WIREFRAME, CONTOUR, PATCHNOGRID, PATCHCONTOUR, LINE. По умолчанию устанавливается HIDDEN.

> f:=proc(x,y) x^2+y^3 end:

> plot3d(f(x,y),x=0..10,y=0..10,style= WIREFRAME);



> plot3d (f(x,y),x=0..10,y=0..10,style= POINT);



9.3.4. Цвет

Цвет графика устанавливается параметром *color=c* (colour=c), где c – предопределенные цвета:

aquamarine	black	blue	navy
coral	cyan	brown	gold
green	gray	grey	khaki
magenta	maroon	orange	pink
plum	red	sienna	turquoise
violet	wheat	white	yellow

Задать цвет можно двумя способами:

1. *color = f(x,y)* – цвет определяется по HUE-алгоритму;
2. *color = [expr1,expr2,expr3]* – цвет определяется по RGB-алгоритму.

Пример:

color=x*y – по HUE-алгоритму;

color=[sin(x*y),cos(x*y),tan(x*y)]) – по RGB-алгоритму.

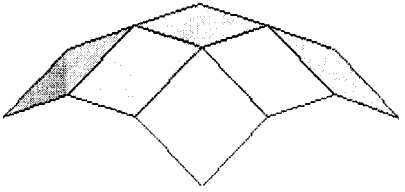
> plot3d(x^4+y^4,x= – 2..2,y= – 2..2,color=x^2+y^2);

> plot3d(x^4+y^4,x= – 2..2,y= – 2..2,color=[sin(x*y),cos(x*y),tan(x*y)]);

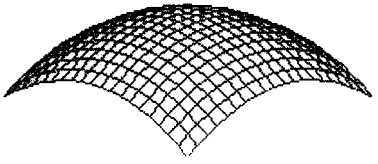
9.3.5. Нанесение сетки

Для наглядности чертежа на графики 3D накладывают сетку. Размер ячеек сетки меняется с помощью параметра *grid*, который описывается в форме *grid=[m,n]* и определяет количество четырехугольников – элементов сетки. По оси X будет уложено (m–1), а по оси У (n–1) четырехугольников.

> plot3d((– x^2 – y^2),x= – 2..2,y= – 2..2,grid=[4,4]);



> plot3d((– x^2 – y^2),x= – 2..2,y= – 2..2,grid=[20,20]);



9.3.6. Координаты системы

Пользователь по своему желанию может выбирать следующие типы системы координат:

cartesian – декартова;
spherical – сферическая;

cylindrical – цилиндрическая.

Устанавливается тип координатной системы параметром *coords*. Например: *coords=spherical*. По умолчанию устанавливается декартова система координат.

Если выбрана декартова система координат, то вертикальная координата z выражается как функция координат x и y, т.е. *plot3d(z(x,y), x=a..b, y=c..d);*

Если выбрана сферическая система координат, то команда plot3d должна быть записана в следующей форме:

plot3d(r(theta,phi), theta=a..b, phi=c..d, coords=spherical);

Здесь theta – угол, измеряемый от x – оси в плоскости XY; phi – угол, измеряемый от положительной полуоси z; r(theta,phi) – модуль радиуса-вектора.

Соотношение между декартовыми координатами и сферическими координатами выражается формулами:

```
x = r * sin(phi) * cos(theta)
y = r * sin(phi) * sin(theta)
z = r * cos(phi)
```

Если выбрана цилиндрическая система координат, то команда plot3d должна быть записана в следующей форме:

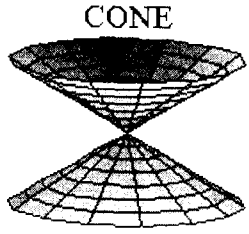
plot3d(r(theta,z), theta=a..b, z=c..d, coords=cylindrical);

Здесь theta – угол, измеряемый от положительной полуоси x; z – координата (высота); r(theta,z) – модуль радиуса-вектора; theta может изменяться от 0 до 8*Pi.

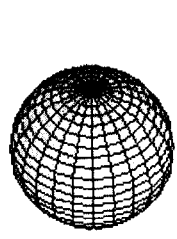
Соотношение цилиндрических и декартовых координат выражается формулами:

```
x = r * cos(theta)
y = r * sin(theta)
z = z
```

> plot3d(height, angle=0..2*Pi, height= – 5..5, coords=cylindrical, title=`CONE`);



> plot3d(1,t=0..2*Pi, p=0..Pi, coords=spherical, scaling=CONSTRAINED, title=`SPHERE`);



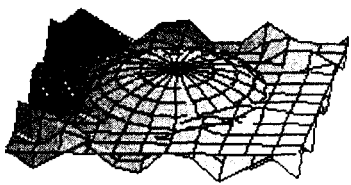
9.3.7. Редеринг

При изменении некоторых параметров построения графиков (но не самой функции!) чертеж можно быстро перерисовать командой *display*.

> with(plots):

```
> F:=plot3d(sin(x*y),x= – Pi..Pi,y= – Pi..Pi): G:=plot3d(.2*x +.2* y,x= –
Pi..Pi, y= – Pi..Pi):
> H:=plot3d([2*sin(t)*cos(s),2*cos(t)*cos(s),2*sin(s)],s=0..Pi,t= – Pi..Pi):
> display3d({F,G,H});
```

```
> display3d({F,G,H},orientation=[10,20]);
```

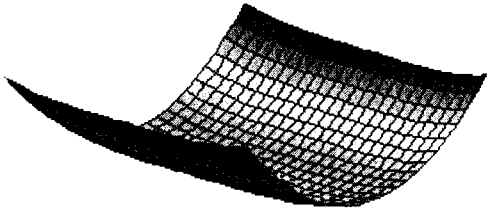


9.3.8. Масштабирование осей

Оси графиков масштабируются параметром *scaling=s*. Здесь s может принимать значения UNCONSTRAINED или CONSTRAINED (несдавленный и сдавленный соответственно).

> plot3d(5*x^2+.6*y^4,x= – 1..1, y= – 1..1,scaling=CONSTRAINED, color=x);

> plot3d(5*x^2+.6*y^4, x= – 1..1, y= – 1..1, scaling=UNCONSTRAINED, color=x);



9.3.9. Оформление графиков

Оформить график можно с помощью разнообразных параметров функции plot3d.

Параметр *title=t* определяет заголовок для чертежа. Здесь t – строка, по умолчанию t – пустая строка.

Параметр *labels=[x,y,z]* определяет метки для осей. Здесь x, y, z – строки (по умолчанию нет меток).

Параметр *axes=f*, где f – одно из выражений BOXED, NORMAL, FRAME, NONE; определяет, как будут изображаться оси координат. По умолчанию устанавливается NONE.

Параметр *contours = n*, где n – целое положительное число или список контурных значений (по умолчанию n = 10).

Параметр *projection=r*, где r – действительное число от 0 до1, определяет точку взгляда (1 – определяет точку взгляда, с которой поверхность видна под прямым углом; 0 – определяет широкоугольную перспективу). Также r может быть одним из зарезервированных имен: 'FISHEYE', 'NORMAL', 'ORTHOGONAL', которые соответствуют значениям 0, 0.5 и 1 соответственно. По умолчанию устанавливается ORTHOGONAL.

Параметр *orientation=[theta,phi]* определяет углы theta и phi точки трехмерного пространства, в которой будет находиться наблюдатель. Точка взгляда описана в сферических координатах, где theta и phi – углы в градусах (по умолчанию оба угла равны 45 градусам).

Параметр *view*=zmin..zmax или [xmin..xmax, ymin..ymax, zmin..zmax] показывает минимальные и максимальные координаты плоскости, в пределах которых поверхность будет отображаться на экране. По умолчанию отображается вся поверхность.

Параметр *shading=s*, где s – одно из выражений: XYZ, XY, Z, Z_GREYSCALE, Z_HUE, NONE; определяет, как будет окрашена поверхность. По умолчанию окраска выбирается в зависимости от типа устройства вывода.

Параметр *ambientlight=[r,g,b]* устанавливает соотношение красного, зеленого и синего цветов для освещения поверхности (освещение определяется пользователем). Здесь r, g, b – действительные числа от 0 до 1.

Параметр *light=[phi,theta,r,g,b]* добавляет источник света, помещенный на прямой, определяемой углами phi и theta сферических координат. Цвет источника определяется значениями r, g, b (действительных

чисел от 0 до 1), которые устанавливают соотношение красного, зеленого и синего цветов.

Параметр *thickness=n* определяет толщину линий чертежа; n должно быть равно 0, 1, 2 или 3 (0 – по умолчанию).

Параметр *linestyle=n* определяет точечный шаблон для линий чертежа. Если n = 0 или 1, то линии будут сплошными.

Параметр *symbol=s* определяет символ для точек чертежа. Здесь s – одно из следующих выражений: BOX, CROSS, CIRCLE, POINT, DIAMOND.

Параметр *font=l* определяет шрифт для текстовых объектов чертежа. Здесь l – список [family, style, size], где family – одно из выражений TIMES, COURIER, HELVETICA, SYMBOL. Для TIMES стиль может быть одним из ROMAN, BOLD, ITALIC или BOLDITALIC. Для HELVETICA и COURIER стиль может быть выбран из BOLD, OBLIQUE, BOLDOBLIQUE. Для SYMBOL стиль не указывается. Size – размер точек шрифта.

Параметр *titlefont=l* определяет шрифт для заголовка (так же как и для font).

Параметр *axesfont=l* определяет шрифт для координатных меток осей координат (так же как и для font).

Параметр *labelfont=l* определяет шрифт для меток (labels) на осях координат (так же как и для font).

Для параметров *axes, style, projection, shading и scaling* значения могут быть напечатаны как прописными, так и строчными буквами. Например, axes=BOXED эквивалентно axes=boxed.

9.3.10. Анимация

Как и для двухмерных графиков, для графиков 3D можно применить анимацию.

Синтаксис команды:

animate3d(F, x, y,t);

где F = F(x,y,t); x,y,t – диапазоны изменения величин.

```
> with(plots):
> animate3d(cos(t*x)*sin(t*y),x= – Pi..Pi, y= – Pi..Pi,t=1..2);
```

Количество кадров можно регулировать, используя параметр frames. Аналогичный результат можно получить с помощью команды display3d при установке *insequence=true.*

```
> with(plots):
> P := animate3d(x – k*y+1,x= – 10..10,y= – 10..10,k= – 10..0,frames=4):
> Q := animate3d(x – k*y+1,x= – 10..10,y= – 10..10,k=0..10,frames=4):
> display([P,Q], insequence=true);
```

9.4. Сохранение графиков

Графики в Maple можно определить как структуру данных, в которой будет отражена вся информация о чертеже. С такими данными можно производить действия так же, как и над обычными выражениями, т.е. их можно распечатать, сохранить в файле и т. п.

В Maple существуют два типа данных, которые предназначены для сохранения информации о чертежах:

PLOT - для графиков 2D;

PLOT3D - для графиков 3D.

```
> gr2d:=plot(sin(x),x=0..Pi,color=blue):
> type(gr2d,PLOT);
```

true

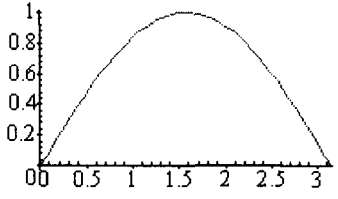
> gr3d := plot3d (sin(x) * cos(y), x=0..Pi, y=0..Pi):

> type(gr3d,PLOT3D);

true

Построить графики по уже определенным переменным нетрудно.

> eval(gr2d);



Переменные, содержащие информацию о чертежах, можно сохранить в файле командой save.

> save gr2d,gr3d,`graph.txt`:

Очистим память Maple-системы:

> restart:

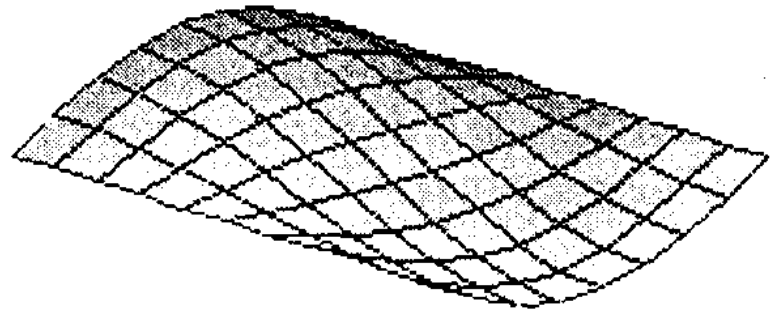
Переменные уничтожены, и график построить по переменным нельзя.

> eval(gr2d);

gr2d

Но переменные gr2d и gr3d можно считать из файла командой read.

```
> read `graph.txt`:
> eval (gr3d);
```



9.5 Графические библиотеки

Графические библиотеки предназначены для расширения графических возможностей среды Maple. Существуют три основные графические библиотеки: plots, stats, DEtools.

Библиотека plots предназначена для построения графических объектов, которые нельзя построить обычными функциями plot и plot3d.

Графическая библиотека stats предназначена для обработки статистической информации и построения гистограмм.

Графическая библиотека DEtools предназначена для построения графиков решения системы дифференциальных уравнений.

10. СТАТИСТИЧЕСКИЕ ВЫЧИСЛЕНИЯ

Maple V содержит мощную библиотеку stats, поддерживающую разнообразные статистические вычисления и генерирующую реализации случайных последовательностей, с заданными законами распределения.

Библиотека включает следующие подбиблиотеки:

- describe – для вычислений статистических характеристик данных;
- fit – для регрессионного анализа (аппроксимации данных заданными зависимостями);
- transform – преобразования данных;
- random – для генерирования случайных чисел с заданными свойствами;
- statevalf – для получения численных оценок массивов данных;
- statplots – для графического представления данных.

Подключение библиотеки выполняется командой:

>with(stats):

10.1 Подбиблиотека DESCRIBE

Эта подбиблиотека позволяет вычислять широкий спектр статистических характеристик, назначение которых не всегда можно отыскать даже в специализированной справочной литературе.

Формат вызова команд:

stats[describe, <function>](args)

или

describe[<function>](args),

где (*args*) – массив данных, а вместо <*function*> может быть использовано одно из следующих ключевых слов:

- coefficientofvariation – усредненное отклонение;
- count – подсчитывает число элементов в массиве данных;
- decile – делит диапазон изменения данных на 10 частей;

- geometricmean – геометрическое среднее;
- harmonicmean – гармоническое среднее;
- kurtosis – коэффициент Куртосиса;
- linearcorrelation – линейная корреляция;
- mean – среднее арифметическое (начальный момент 1-го порядка);
- median – медиана распределения;
- mode – мода распределения;
- moment – начальные и центральные моменты k-го порядка;
- quadraticmean – среднее квадратическое;
- range – размах (диапазон изменения данных);
- standarddeviation – стандартное отклонение;
- variance – дисперсия (центральный момент 2-го порядка).

Поясним использование этих функций на примерах.

Зададим два массива чисел data1 и data2:

```
> data1:=[1.,2.,3.,4.,3.,5.,2.,7.,6.,2.,5.]:
> data2:=[1,2,4,4,3.,7.,3.,8,5,3,7]:
```

Покажем на примерах, как можно воспользоваться подбиблиотекой DESCRIBE:

```
> describe[ coefficientofvariation](data1);
```

.5012484414

```
> describe[count](data1);
```

11

```
> describe[countmissing](data1);
```

0

```
> describe[covariance](data1,data2);
```

3.553719010

```
> describe[geometricmean](data1);
```

3.149449927

```
> describe[harmonicmean]([1,3,4,5]);
```

240/107

```
> evalf(describe[kurtosis](data2));
```

1.979718020

```
> evalf(describe[linearcorrelation](data1,data2));
```

.9128215626

```
> describe[mean](data1);
```

3.636363636

```
> describe[meandeviation](data1);
```

1.603305785

```
> describe[median](data1);
```

3.

```
> describe[mode](data1);
```

2.

Вычислим начальные моменты 1-го и 8-го порядка относительно нуля:

```
> describe[moment[1,0]]([1,2,3,4,5])
```

3

```
> evalf(describe[moment[8,0]]([1,2,3,4,5]));
```

92595.80000

Вычислим центральный момент 2-го порядка. Обратите внимание на возможность использования статистических функций как вложенных.

```
> describe[moment[2,mean]]([1,2,3,4,5]);
```

2

```
> describe[percentile[50]](data1);
```

3.

```
> describe[quadraticmean](data1);
```

4.067610423

```
> evalf(describe[quartile[3] ](data2));
```

5.500000000

Вычислим диапазон, в котором лежат данные массива data1:

```
> describe[range](data1);
```

1. .. 7.

```
> describe[skewness](data1);
```

.3662496971

```
> describe[standarddeviation](data1);
```

1.822721605

```
> describe[variance](data1);
```

3.322314050

Maple V позволяет создавать собственную универсальную функцию, с помощью которой далее можно вычислять несколько показателей одновременно. Например, создадим функцию Gen, которая сразу определяет: количество элементов в данных, среднее значение и дисперсию.

> Gen:=[describe[count],describe[mean],describe[variance]]:

> Gen(data1);

[11, 3.636363636, 3.322314050]

10.2. Подбиблиотека FIT

Предназначена для нахождения корреляционных отношений и аппроксимации статистических данных выбранными зависимостями.

Формат вызова:

stats[fit, leastsquare[vars, eqn, parms]](data)

или

fit[leastsquare[vars, eqn, parms]](data),

где *data* – список данных; *vars* – список переменных, в порядке представления данных; *eqn* – аппроксимирующее уравнение (по умолчанию линейное); *parms* – множество параметров, которые будут заменены вычисленными значениями.

Ниже аппроксимируются три массива. В качестве переменных выбраны x, y, z. Поскольку вид аппроксимирующего выражения не оговаривается, то по умолчанию система выбирает его линейным.

> z1:=fit[leastsquare[[x,y,z]]]([[1,2,3,5],[2,4,6,8],[3,5,7,10]]);

$$z1 := z = 1 + x + \frac{1}{2}y$$

При описании данных удобно пользоваться функцией Weight (элемент, число повторений):

> fit [leastsquare[[x,y,z]]]([[1,2,3,5,5],[2,4,6,8,8],[3,5,7,10,Weight(15,2)]]);

$$z = 1 + \frac{13}{3}x - \frac{7}{6}y$$

Возьмем два массива X0 и Y0, состоящих каждый из четырех элементов, и аппроксимируем эти данные уравнением 2-го порядка. На показанных ниже трех примерах легко проследить, как можно менять формат команды.

> X0:=[1,2,3,4]: Y0:=[0,6,14,24]:

> Ur_1:= fit[leastsquare[[x,y], y=a*x^2+b*x+c, {a,b,c}]]([X0, Y0]);

$$Ur\_1 := y = x^2 - \frac{2}{5}x - \frac{8}{5}$$

>Ur_2:= fit[leastsquare[[x,y], y=a*x^2+b*x+c]]([X0, Y0]);

$$Ur\_2 := y = x^2 - \frac{2}{5}x - \frac{8}{5}$$

>Ur_3:= fit[leastsquare[[x,y], y=a*x^2+b*x+c, {a,b}]]([X1, Y1]);

$$Ur\_3 := y = \left(\frac{55}{659}c + \frac{747}{659}\right)x^2 + \left(-\frac{399}{659}c - \frac{902}{659}\right)x + c$$

Трансформируем полученный результат в процедуру:

> My_function:=unapply(rhs(Ur_3),x,c);

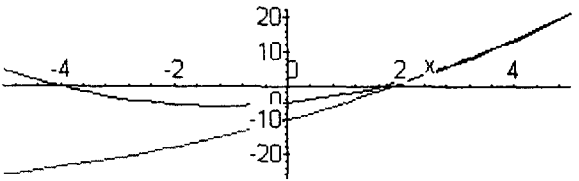
$$My\_function := (x, c) \rightarrow \left(\frac{55}{659}c + \frac{747}{659}\right)x^2 + \left(-\frac{399}{659}c - \frac{902}{659}\right)x + c$$

Далее этой функцией можно воспользоваться при вычислениях или построениях графиков, например:

> My_function(1,3);

$$\frac{790}{659}$$

> Plot({My_function(x, – 10), My_function(x, – 5),x= – 5..5);



10.3. Подбиблиотека TRANSFORM

Подбиблиотека содержит богатые возможности выполнения преобразований над данными, что видно при рассмотрении ее содержания, приведенного ниже.

Формат вызова команд:

stats[transform, <function>](args)

или

transform[<function>](args),

где вместо *<function>* может быть использовано одно из следующих ключевых слов:

- apply – замена элементов данных новыми, вычисленными по заданной формуле;
- classmark – заменяет классы данных их средними значениями;
- cumulativefrequency – подсчитываются веса элементов данных;
- deletemissing – вычитание пустых элементов из списка данных;
- divideby – деление каждого элемента на число или статистическую функцию;
- frequency – частотность каждого входящего элемента в данных;
- moving – вычисление средних значений последовательных порций элементов;
- multiapply – преобразование по формуле данных, представленных списками;
- remove – вычитание из данных числа или статистической функции;
- scaleweight – умножение весов данных на число;

- split – разбить данные на множество списков с соответствующими весами;
- standardscore – замена элементов данных;
- statsort – сортировка статистических данных;
- statvalue – дать величину каждого элемента данных и элементов множества, которые описаны весами, как единичный элемент;
- tally – подсчет повторяемости каждого элемента;
- tallyinto – подсчет повторяемости каждого элемента в определенном классе.

Рассмотрим некоторые перечисленные возможности на примерах.

Зададим массив данных data3:

```
> data3:=[1,1,2,3,3,4,4,4,5,6,6,9]:
```

Подсчитаем веса, входящих в него элементов:

```
> y45:=transform[tally](data3);
```

y45 := [Weight(3, 2), Weight(4, 3), 5, Weight(6, 2), 9, 2, Weight(1, 2)]

Выделим из полученного списка y45 первый элемент:

```
> op(1,y45);
```

Weight(3, 2)

Подсчитаем, сколько элементов попадает в указанные диапазоны:

```
> y40:= transform[tallyinto](data3,[0..5,5..10,10..15]);
```

y40 := [Weight(0 .. 5, 8), Weight(5 .. 10, 4), Weight(10 .. 15, 0)]

После этого можно определить частотность одинаковых элементов, например, так:

```
>transform[frequency](y40);
```

[8,4,0]

Присвоим идентификатору n3 значение числа элементов, указанного во втором члене полученного списка y40:

> n2:=op(2,op(2,y40));

$$n2 := 4$$

Далее можно воспользоваться переменной n2. Например, вычислим n2* n2-1:

> n2*n2 – 1;

$$15$$

Вычтем из каждого элемента списка data3 число 3:

> transform[remove[3]](data3);

$$[-2, -2, -1, 0, 0, 1, 1, 1, 2, 3, 3, 6]$$

Вычтем из каждого элемента списка data3 среднее значение этого списка:

> transform[remove[mean]](data3);

$$[-3, -3, -2, -1, -1, 0, 0, 0, 1, 2, 2, 5]$$

Разделим элементы списка data3 на 4:

>transform[divideby[2]](data3);

$$\left[\frac{1}{2}, \frac{1}{2}, 1, \frac{3}{2}, \frac{3}{2}, 2, 2, 2, \frac{5}{2}, 3, 3, \frac{9}{2}\right]$$

Разделим элементы списка data3 на их среднее:

>transform[divideby[describe[mean](data3)]](data3);

$$\left[\frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, \frac{3}{4}, 1, 1, 1, \frac{5}{4}, \frac{3}{2}, \frac{3}{2}, \frac{9}{4}\right]$$

Найдем средние значения трех последовательных элементов в списке data3. Очевидно, что число элементов в новом списке будет меньше на 3:

> transform[moving[3]](data3);

[3, 4, 3, 5, 4, 10/3, 5/3, 5/3, 8/3]

Интегральная характеристика накопления весов элементов в данных

> transform[cumulativefrequency]([5,Weight(1..7, 10),2,2,2,2]);

[10, 11, 12, 13, 14]

Если функция определена, то можно выполнить преобразования над списком данных. Например, определим функцию f, возвращающую 3-ю степень аргумента:

> f:=x – >x^3;

$$f := x \rightarrow x^3$$

> Mass_1:=transform[apply[f]]([2,3,4]);

Mass_1 := [8, 27, 64]

Затем найдем разность между соответствующими элементами данных:

> Itog:=transform[multiapply[(x,y) – > x – y]]([[12,40,70], Mass_1]);

Itog := [4, 13, 6]

Зададим список элементов data_0, рассортируем его, определим среднее значение и оценим разброс данных:

> data_0:=[1,3,5,3]: transform[statsort] (data_0); describe[mean](data_0); describe[standarddeviation](data_0);

[1,3,3,5]

3

$\sqrt{2}$

> transform[statvalue]([Weight(3,10),2,2,1,1,1]);

[3, 2, 2, 1, 1, 1]

10.4. Подбиблиотека RANDOM

Формат вызова:

stats[random, distribution](quantity, uniform, method)

или

random[distribution](quantity, uniform, method)

Опишем назначение параметров командной строки:

distribution – описание закона распределения получаемых чисел;

quantity – положительное целое число (определяет, сколько случайных чисел требуется получить) или ключевое слово 'generator' (опция, по умолчанию устанавливается 1);

uniform – процедура, которая генерирует числа равномерного распределения, или ключевое слово 'default' (опция, по умолчанию устанавливается 'default');

method – одно из следующих ключевых слов 'auto', 'inverse' или 'builtin' (опция, по умолчанию устанавливается 'auto').

Библиотека включает следующие дискретные и непрерывные распределения.

Дискретные распределения:

binomiald[n,p] – биномиальное:
binomial(n,x)*p^x*(1–p)^(n–x);

discreteuniform[a,b] – дискретное равномерное:
1/(b–a+1) и равно 0, если x<a или x>b;

empirical[list_prob] – эмпирическое:
равно 0, если x<1 или x>nops(list_prob), иначе равно list_prob[x];

hypergeometric[N1, N2, n] – гипергеометрическое:
binomial(N1,x) * binomial(N2,n–x) / binomial(N1+N2,n);

negativebinomial[n,p] – отрицательное биномиальное:
binomial(n+x–1,x)*p^n*(1–p)^x;

poisson[mu] – Пуассона:
exp(–mu)*mu^x/x!;

Непрерывные распределения:

beta[nu1, nu2] – бета:
1 / Beta(nu1, nu2) * x ^ (nu1–1) * (1–x) ^ (nu2–1);

cauchy[a, b] – Коши:
1/(Pi*b*(1+(x–a)/b)^2), b>0;

chisquare[nu] – χ–квадрат:
x^((nu–2)/2) exp(–x/2) / 2 ^ (nu/2) / GAMMA(nu/2), x>0, nu>0;

exponential[alpha, a] – экспоненциальное:
alpha * exp(–alpha * (x–a)), если x>=a; и равно нулю, если x<a;

fratio[nu1, nu2] – оносительное (Фишера):
GAMMA((nu1+nu2)/2) / GAMMA(nu1/2) / GAMMA(nu2/2) * (nu1/nu2) ^ (nu1/2) * f ^ ((nu1–2)/2) / (1+ (nu1/nu2)*f) ^ ((nu1+nu2)/2), x>0, nu1>0, nu2>0:

gamma[a, b] – гамма:
x^(a–1)*exp(–x/b)/GAMMA(a)/b^a, x>0, a>0, b>0;

laplaced[a, b] – Лапласа:
1/(2*b) * exp(–abs(x–a)/b), b>0.

logistic[a, b] – распределение, которое описывается функцией: exp(– (x–a)/b) / b / (1+ exp(–(x–a)/b))^2, b>0;

lognormal[mu, sigma] – логнормальное:
1/(x*sqrt(2*Pi)*sigma) * exp(– ((ln(x)–mu)/sigma)^2 /2), x>0;

normald[mu, sigma] – нормальное (Гаусса):
exp(–(x–mu)^2/2/sigma^2) / sqrt(2*Pi*sigma^2);

students[nu] – Стюдента:
GAMMA((nu+1) / 2) / GAMMA (nu/2) / sqrt (nu*Pi) / (1+t^2/nu) ^ ((nu+1)/2);

uniform[a, b] – равномерное:
1/(b–a) если a<=x<=b, и 0 в других случаях;

weibull[a, b] – Вейбула:
a*x^(a–1)*exp(–(x/b)^a)/ b^a, x>0, a>0, b>0.

Рассмотрим возможности подбиблиотеки RANDOM на примерах.

Сгенерируем четыре случайных числа из генеральной совокупности, описываемой нормальным законом распределения:

```
> stats[random, normald](4);
```

.7433, .9421, –.8268, –.8957

Аналогично выполненному выше создадим массив из 53 элементов, но с заданными параметрами распределения: со средним значением – 7 и средним квадратическим отклонением – 1.5.

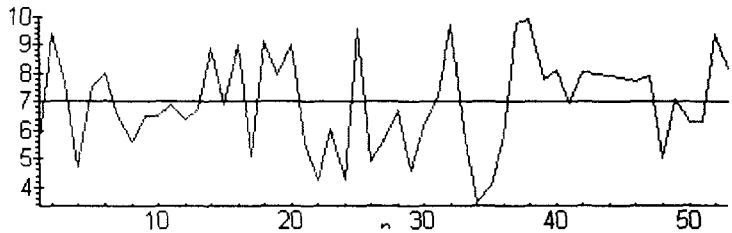
```
> k:=53:

>mas_1:=stats[random, normald[7,1.5]](k):
```

Оценим среднее значение в полученном нами массиве и построим график, на котором покажем последовательность элементов, соединенных линиями, а также линию, равную среднему значению.

```
> describe[mean]([mas_1]);
```

7.076

> plot({describe[mean]([mas_1]), [[n,op(n,[mas_1])]$n=1..k], n=1..k, style=line);



Генерируется четыре числа из gamma распределения:

>random[gamma[3]](4);

2/1077, 1.8891, 2,8672, 5.5985

Назначение идентификатора Gen1 как генератора и получение четырех чисел с точностью представления по умолчанию:

> Gen1:=random[gamma[3]]('generator'):

> 'Gen1( )'$4;

4.322417178, 1.556014201, 1.085076020, 7.213556406

Назначение Gen2 как генератора и получение восьми чисел с точностью представления до четырех:

> Gen2:=random[gamma[3]]('generator[4]'):

> 'Gen2( )'$8;

4.458, 5.016, 3.087, 5.379, 2.430, 1.485, .6150, 3.078

10.5. Подбиблиотека STATEVALF

Формат вызова:

stats[statevalf, <function>, <distribution>](args)

или

statevalf[<function>, <distribution>](args),

где *<function>* – следующие переменные:

для непрерывных функций распределения:

- cdf – функция вероятности (интегральная функция);
- icdf – обратная функция от интегральной функции;
- pdf – функция плотности вероятности (дифференциальная функция);

для дискретных функций распределения:

- dcdf – дискретная функция вероятности;
- icdcf – обратная функция;
- pf – функция плотности вероятности.

Ниже приведены примеры, поясняющие использование подбиблиотеки STATEVALF.

Для компактности выводимых результатов установим точность представления чисел до четырех значащих цифр:

> Digits:=4:

Определим значение функции плотности распределения для распределения Пуассона в точке 2:

> stats[statevalf,pf,poisson[3]](2);

.2241

Определим значение обратной функции от интегральной функции нормального распределения в точке 0.7:

> stats[statevalf,icdf ,normald](0.7);

.5244

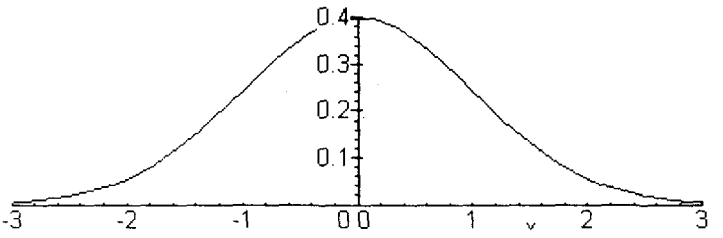
Определим значение функции плотности вероятности нормального распределения в точке 0.5:

> stats[statevalf,pdf,normald](.5);

.3519

Для пояснения сказанного построим функцию

>plot(stats[statevalf,pdf,normald](x),x=–3..3);



Получим восемь случайных чисел для гамма-распределения и оценим их среднее значение:

> y3:=stats[random,gamma[3,1]](8);

y3 := 1.382, 3.726, 3.936, 3.930, 2.006, 5.439, 1.305, .5604

> describe[mean]([y3]);

2.786

Определим значение обратной функции от интегральной функции вероятности вейбуловского распределения в точке 0.5:

> stats[statevalf,icdf,weibull[3, 2]](0.5);

1.770

Получим пять случайных чисел для бета-распределения

> stats[random,beta[1,2]](5);

.4621, .4244, .2967, .4706, .3968

Получим четыре случайных числа для экспоненциального распределения и оценим их среднее значение:

> t1:=stats[random,exponential[3]](4); describe[mean]([t1]);

$$t1 := .01257, .1229, .2387, .04893$$

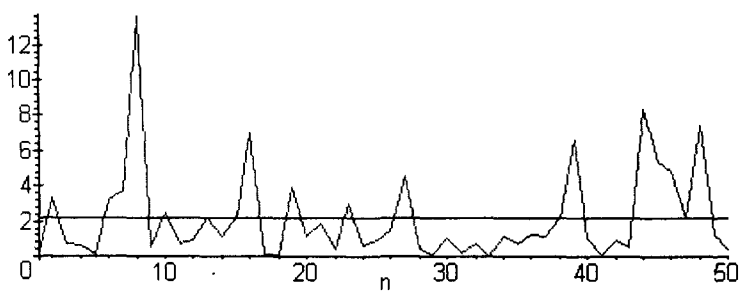
$$.1058$$

Генерирование чисел из распределения χ-квадрат с точностью до третьей значащей цифры. Команда из трех операторов: в первом задаем алгоритм, во втором формируем генератор, а в третьем определяем значение константы k.

> A:=stats[random,chisquare[2]](generator[5]): t4:='A( )'$50: k:=50:

Теперь построим реализацию из случайной последовательности чисел, определяемую вейбуловским распределением с заданными параметрами. Для лучшего представления графика зададим ключ соединения точек линиями (style=line).

> plot({describe[mean]([t4]),[[n,op(n,[t4])]$n=1..k]},n=1..k,style=line);



10.6. Подбиблиотека STATPLOTS

Приведенные здесь примеры показывают возможности системы Maple V визуализировать статистические данные.

Формат вызова:

stats[statplots, <function>](args)

или

statplots[<function>](args),

где в качестве *<function>* выбирается один из следующих видов графического представления данных: boxplot , histogram, notchedbox, quantile, quantile2, scatter1d, scatter2d, symmetry.

Подключим библиотеку STATS командой:

>with(stats):

Далее создадим два массива данных Xdata и Ydata:

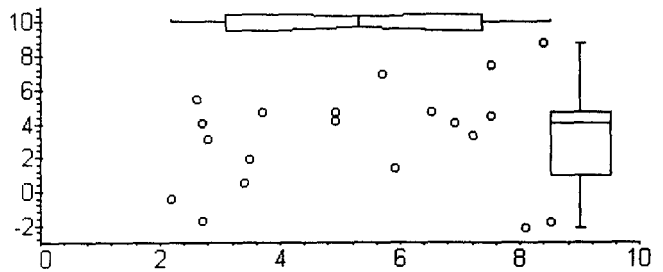
> Xdata := [7.5, 4.9, 8.4, 6.5, 7.5, 2.7, 7.2, 5.9, 6.9, 3.4, 8.5, 5.7, 2.7, 8.1, 3.7, 2.6, 3.5, 2.2, 2.8, 4.9]:

> Ydata:= [4.5, 4.7, 8.7,4.7, 7.5, – 1.7, 3.3, 1.4, 4.1, .5, – 1.8, 6.9, 4.1, – 2.1, 4.7, 5.4, 1.9, – .4, 3.1,4.2]:

> plots[display]({statplots[scatter2d](Xdata,Ydata), statplots[boxplot[9]](Ydata),

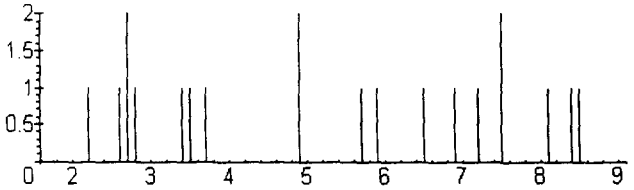
> statplots[xyexchange] (statplots [notchedbox[10]](Xdata))}, view =[0..10, – 3..11],

> axes=FRAME,symbol=circle);



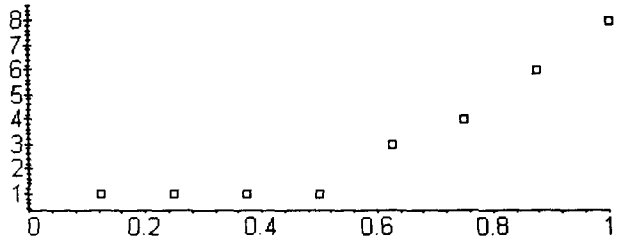
Построим гистограмму распределения для массива Xdata:

> statplots[histogram[– 10..10]](Xdata);



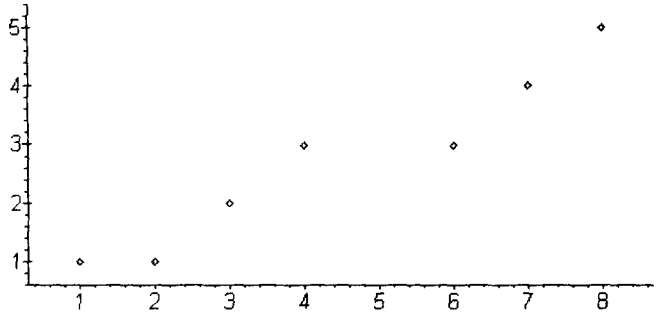
Визуализация данных в отсортированном порядке:

>plots[display](statplots[quantile]([6,3,1,4,8,1,1,1]),symbol=box);



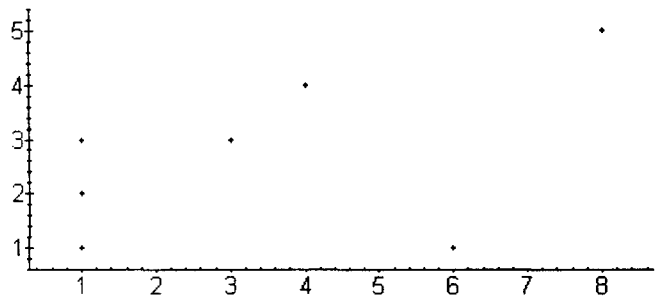
Графическое представление двухмерного массива только для точек, заданных двумя координатами:

>plots[display](statplots[quantile2]([6,3,7,4,8,1,2,1],[1,3,1,4,5,2,3,1]),symbol=diamond);



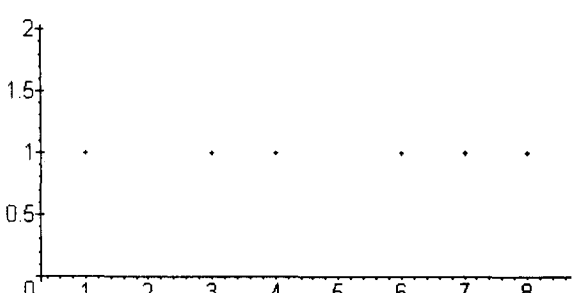
Построение точек двухмерного массива, определенных двумя координатами:

> statplots[scatter2d]([6,3,1,4,8,1,1,1],[1,3,1,4,5,2,3,1]);



Нанесение точек,встречающихся в массиве данных:

> statplots[scatter1d]([6,3,1,4,8,1,7,1]);



11. ВЫВОД ДАННЫХ В ДРУГИЕ СРЕДЫ

11.1. Вывод в редактор TeX

Maple V позволяет автоматизировать набор формул для специалистов, работающих с общепринятым среди математиков и физиков редактором TeX. Вывод выражений в файл формата редактора TeX обеспечивает команда latex.

Формат команды latex:

latex(expr, filename),

где expr – выражение; filename – имя файла (необязательный параметр).

Зададим имя файла:

```
> filename1:=output;
```

$$filename1 := output$$

Зададим выражение для записи:

```
> expr:=x+y^6;
```

$$expr := x + y^6$$

Запись:

```
> latex(expr,filename1);
```

Вывод с предварительным вычислением:

```
> latex(int(x,x)=y);
```

```
{\frac {{x}^{2}}{2}}=y
```

Вывод без вычисления:

```
> latex(Int(x,x)=y);
```

```
\int \!x{dx}=y
```

11.2. Получение кода языков Fortran и C

Maple V может генерировать код C и Fortran с помощью команд: fortran, C. Ниже определим выражение для перевода в код Fortran:

> f:=2^x+x^2;

$$f := 2^x + x^2$$

Тогда команда для генерации кода Fortran и записи его в файл fileout.for будет:

> fortran(f,filename=`fileout.for`);

Формат команды fortran:

fortran(expr, filename=str, optimize),

где expr – выражение; str – строка (имя файла) (если filename=str не указано, то вывод осуществляется на экран); optimize – необязательный параметр, определяющий, что будет сгенерирован оптимизированный код, т. е. вычисления будут производиться по этапам с присвоением значений промежуточным переменным.

Оптимизируем получаемый код:

> fortran(f,optimized);

```
t1 = 2**x
t2 = x**2
t3 = t1+t2
```

Опция mode в команде fortran(f,mode) может принимать следующие значения типов:

single, double, complex, generic.

> fortran(log(x)^3,optimized,mode=complex);

```
t1 = clog(x)
t2 = t1**2
```

```
t3 = t2*t1
```

Трансляция в код С

Подключаем библиотечную функцию С:

> readlib(C):

Формат вызова команды С аналогичен формату вызова команды fortan.

Сгенерируем код С для функции f, определенной выше, и запишем его в файл:

> C(f);

> C(f,filename=cout);

```
t0 = pow(2.0,x)+x*x;
```

Получим оптимизированный код:

> C(f,optimized);

```
t1 = pow(2.0,x);
t2 = x*x;
t3 = t1+t2;
```

Таким образом команды С и fortran позволяют автоматизировать конструирование сложных функций в алгоритмических языках.

12. ЭЛЕМЕНТЫ ЯЗЫКА MAPLE V

12.1. Нотация языка

Нотация – форма записи, принятая в языке. Основные элементы нотации:

:, ; – разделители;
– комментарий;
` – ограничитель строки.

Работа в Maple ведется в диалоговом режиме. В процессе работы среда выдает подсказку ">" – приглашение к вводу строки. В ответ на приглашение пользователь вводит строку. Операторы в строке могут разделяться символами: ":" и ";" . Если используется разделитель ":" , то данная команда исполнится, но результат на экран выведен не будет, в противном случае результат будет выводиться на экран. Комментарий должен начинаться символом "#". Строки заключаются в обратные кавычки: `string...` .

В математике встречается много различных сокращений и условных форм записи. Maple располагает средствами определения сокращенных форм записи (нотаций). Новую нотацию можно ввести с помощью функции alias и macro.

Maple использует 26 прописных и 26 строчных латинских букв, 10 цифр, а также 32 специальных символа (три типа скобок, знаки отношения, арифметических действий и др.).

Имеются пять альтернативных пар символов:
^ и **; [и (|;] и |); { и (*; } и *).

Используются следующие ключевые слова:

by	do	done	elif	else	end	fi	for
from	if	in	local	od	option	options	proc
quit	read	save	stop	then	to	while	

Имеются восемь унарных операторов:

+ – унарный плюс (префикс);
! – факториал (постфикс);
- – унарный минус (префикс);
$ – оператор последовательности (префикс);
not – логическое отрицание (префикс);

%	– метка (префикс);
&string	– нейтральный оператор (префикс);
	– десятичная точка (префикс или постфикс).

Бинарных операторов 27:

+	– сложение;
@	– композиция;
<, <=, >, >=, =, <>	– операторы отношения;
-	– вычитание;
@@	– повторная композиция;
mod	– модуль;
*	– умножение;
and	– логическое "И";
:=	– присвоение;
/	– деление;
or	– логическое "ИЛИ";
union	– объединение множеств;
**	– степень;
.	– конкатенация;
minus	– разность множеств;
^	– степень;
..	– область;
intrsect	– пересечение множеств;
$	– повторение;
,	– разделение выражений;
&*	– некоммутативное умножение;
&string	– нейтральный оператор.

Имена в Maple должны начинаться с буквы и могут включать цифры и знак "_".

Максимальная длина имени – 499 символов. Прописные и строчные буквы считаются различными символами в имени. Например, переменные r34y и r34Y считаются различными.

Имена и выражения связываются через оператор присвоения:

<имя>:=<выражение>

Имеются встроенные константы: Pi, Digits,E,I и т.п. Константы могут быть: целые, простые и десятичные дробные, стринговские.

Допустимо использование пустого оператора: a:=1; ;x:=2;

Завершение работы с Maple происходит при выполнении пользователем по одной из трех команд: quit, done, stop.

Символ "\" может быть использован для обозначения продолжения строки. Например:

```
> x2:= 1234567890\
> 0987654321;
```

$$x2 := 12345678900987654321$$

Аналогично символу "\" конкатенация выражений осуществляется и символом ".", например:

```
> vv:=si.
> n(x);
```

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

12.2. Работа с файлами

Имеются следующие функции для работы с файлами:

writeto	– запись результатов в новый файл;
appendto	– дозапись результатов в существующий файл;
open	– открытие файла;
write	– запись в файл;
writeln	– запись строки в файл;
close	– закрытие файла;
save	– запись выражений в файл;
read	– считывание файла.

Включим режим записи в файл res.txt :

```
> writeto(`rez.txt`);
```

А теперь выполним вычисления, результаты которых будут записаны в файл:

> evalf(Pi,10);

$$3.141592654$$

> 5*234;

$$1170$$

Прекращение записи в файл:

> writeto(terminal);

Дозапись в файл rez.txt возможна при помощи appendto:

> appendto(`rez.txt`);

> f:=x^2+56;

$$f := x^2 + 56$$

> writeto(terminal);

Считать файл rez.txt можно при помощи команды read:

> read(`rez.txt`);

$$3.141592654$$

$$1170$$

$$f := x^2 + 56$$

В файл также можно записывать любые выражения. Для этого сначала надо открыть файл командой open. Затем информацию можно записывать при помощи write и writeln. После окончания записи файл необходимо закрыть командой close.

Примечание: открыт может быть только один файл; перед использованием описанных выше функций их необходимо определить с помощью readlib(write).

Пример:

```
> readlib(write):

> open(out1);
> write(5+4);
> writeln(10*3);
> writeln(int(x,x));

> close( );
```

После выполнения этих команд в файл запишутся результаты вычислений.

Примечание: команда writeto формирует файл, содержащий операторы, а команда write записывает в файл результат вычислений.

С помощью команды save в файл можно записать все или некоторые выражения из сессии Maple.

```
> f:=x^2+45;
```

$$f := x^2 + 45$$

```
> save f,one;
> read one;
```

$$f := x^2 + 45$$

Примечание: если имя файла указано с расширением "m", то выражения автоматически сохраняются во внутреннем формате Maple. Если не указать список переменных, то будут записаны значения всех переменных сессии Maple.

13. ОПЕРАТОРЫ И ВЫРАЖЕНИЯ

13.1. Типы операторов

13.1.1. Составляющие операторов

В Maple существует большое количество различных операторов. Например, операторы математических операций: +, -, /, * . Другие операции реализуются операторами, которые имеют свое собственное имя – идентификатор. Например: evalm – оператор матричного вычисления. Некоторые операторы требуют определенный список аргументов:

```
> D(sin);
```

$$\cos$$

В этом примере D – оператор, sin – аргумент.

Таким образом, основные составляющие операторов – идентификатор и список аргументов.

Функциональный оператор можно определить в двух нотациях:

1. <функция|переменные> – нотация "<..>"
2. (переменные)–>функция – нотация "–>"

```
> oper1:=<x^3 - 2|x>;
```

$$oper1 := \langle x^3 - 2 \mid x \rangle$$

```
> oper2:=<x+y|x,y>;
```

$$oper2 := (x, y) \rightarrow x + y$$

Эти операторы можно переписать в другой нотации (записи эквивалентны):

```
> oper1:=x - >x^3 - 2;
```

$$oper1 := x \rightarrow x^3 - 2$$

> oper2:=(x,y) – >x+y;

$$oper2 := (x, y) \rightarrow x + y$$

Использование функционального оператора:

> oper1(3);

25

> oper2(3,2);

5

13.1.2. Операторы выбора, циклов, переходов и выхода

В Maple V существуют конструкции операторов управления – ветвления и циклов.

Ветвление:

If – then – elif – else.

Синтаксис оператора ветвления следующий:
if <условие> then <последовательность операторов>
| elif <условие> then <последовательность операторов> |
| else <последовательность операторов> |
fi
Условие – булевское выражение.
Строк ELIF может быть неограниченное количество (fi – обязательно!).

Пусть имеем следующие равенства:

> cond:=true: x:=2: y:=3:

Тогда:

> if cond then r:=2 else r:=3 fi;

$$r := 2$$

> if not cond then r:=0 elif y>x then r:=1 elif x>y then r:=2 else r:=3 fi;

$$r := 1$$

Конструкция циклов

Синтаксис описания оператора цикла следующий:
|for <имя>| |from <выражение>| |by <выражение>|
|to <выражение>| |while <выражение>|
do <последовательность операторов> od;
или
|for <имя>| |in <выражение>| |while <выражение>|
do <последовательность операторов> od;

Конструкция in предназначена для работы с массивами, списками и т.д.
Если from не указано, то цикл начнется с 1.
Если by не указано, то приращение равно 1.
Выражение в to и в in вычисляется перед началом цикла.

> a:=array(1..2);

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

> for i to 2 do a[i]:=i od;

$$a_1 := 1$$

$$a_2 := 2$$

> i:=0:

> while i<>2 do i:=i+1: a[i]:=1: od;

$$i := 1$$

$$a_1 := 1$$

$$i := 2$$

$$a_2 := 1$$

```
> s:=0:
> for z in [1,2,3] do s:=s+z od;
```

$$s := 1$$

$$s := 3$$

$$s := 6$$

Можно организовать вложенные циклы:

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

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

```
> for i to 2  do for j to 2  do b[i,j]:=1 od od;

> print(b);
```

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

Оператор перехода next

Если в цикле необходимо пропустить одну итерацию, то можно использовать оператор next. Действие этого оператора заключается в том, что при данном значении счетчика цикла не будет выполнено никаких действий, а начнется выполнение следующей итерации (т. е. изменится значение счетчика цикла). Next можно использовать только в циклах.

```
> for i to 4 do if i=3 then next else ar[i]:=0 fi od;

> print(ar);
```

$$\begin{bmatrix} 0 & 0 & ar_3 & 0 \end{bmatrix}$$

Оператор выхода break

Для прерывания выполнения цикла можно использовать оператор break. Break можно использовать только в циклах:

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

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

```
> for i to 4 do if i=3 then break else ar[i]:=i fi od;
> print(ar);
```

$$\left[1 \quad 2 \quad ar_3 \quad ar_4\right]$$

13.2. Выражения

В Maple выражения строятся по правилам, которые приняты в большинстве языков программирования (например, в Pascal).

```
> f:=((2*x+y)^2)/(35*sin(y)+x^3);
```

$$f := \frac{(2x+y)^2}{35\sin(y)+x^3}$$

```
> g:=not v1 or v2;
```

$$g := \text{not } v1 \text{ or } v2$$

Логические выражения можно строить с использованием символов отношения: >, <, <>, =.

13.2.1. Метки

Метки в Maple представляются выражением %n, где n – натуральное число. Метки используются для красивого вывода решения.

> y:=exp(5*x^2+3*x+45)/(1+x^3);

$$y := \frac{e^{\left(5x^2+3x+45\right)}}{1+x^3}$$

> diff(y,x$2);

$$10\frac{\%1}{1+x^3}+\frac{(10x+3)^2\%1}{1+x^3}-6\frac{(10x+3)\%1\,x^2}{\left(1+x^3\right)^2}+18\frac{\%1\,x^4}{\left(1+x^3\right)^3}-6\frac{\%1\,x}{\left(1+x^3\right)^2}$$

$$\%1 := e^{\left(5x^2+3x+45\right)}$$

Далее этой меткой можно воспользоваться как выражением:

> z:=t+%1;

$$z := t + e^{\left(5x^2+3x+45\right)}$$

13.2.2. Алгебраические и арифметические операции

Список операций:

a + b	– сложение;
a – b	– вычитание;
a * b	– умножение;
a / b	– деление;
a ^ b	– возведение в степень;
a ** b	– возведение в степень;
n !	– факториал;
min(a,b)	– минимум;
iquo(a,b)	– частное;
irem(a,b)	– остаток;

isqrt(n)	– квадратный корень (целочисленное приближение);
igcd(a,b)	– наибольший общий делитель;
ilcm(a,b)	– наименьший общий множитель;
signum(n)	– знак числа;
abs(n)	– абсолютное значение числа;
max(a,b)	– максимум.

Использовать эти функции достаточно просто:

> 6!;

$$720$$

> signum(– 100);

$$-1$$

> max(x,y)*y!+isqrt(z)*x^7;

$$\max(x, y)\, y! + \mathrm{isqrt}(z)\, x^7$$

13.2.3. Последовательности из операторов

Последовательности из операторов легко формируются при помощи символов-разделителей: ":" и ";" .

Различия в действии этих разделителей были уже описаны.

> v1:=123: int(t^4,t); k:=sqrt(4): D(tan);

$$\frac{1}{5}t^5$$

$$1 + \tan^2$$

> z:=v1+k;

$$z := 125$$

14. ТИПЫ ДАННЫХ

14.1. Основные типы

14.1.1. Константы

Константы в Maple бывают целочисленными, числами с плавающей запятой и обыкновенными дробями. Кроме этих типов констант существуют символьные константы – зарезервированные имена.

Например: false, true, infinity, Pi, I, ...

Проверить принадлежность выражения к классу констант можно при помощи функции type:

```
> type(0.004, constant);

                                true

> type(infinity, constant);

                                true

> type(`str`, constant);

                                false

> type(x^6, constant);

                                false
```

14.1.2. Целые

В среде Maple выражение принадлежит к целому типу (тип integer), если оно состоит из последовательности цифр, не разделенных между собой никакими знаками. Длина последовательности ограничена лишь ресурсами системы, но обычно не должна превышать 500 000 цифр. Число типа integer может быть как положительным, так и отрицательным.

Присвоим переменной var1 значение –1234567890:

> var1:= – 123567890;

$$var1 := -123567890$$

Для определения принадлежности выражения определенному типу служит команда type.

Пример использования типа integer: распечатаем значение 10-го числа Фибоначчи.

> combinat[fibonacci](10);

55

Для работы с целыми числами определены следующие операции:

abs	– модуль числа;
min	– минимум из последовательности;
max	– максимум из последовательности;
factorial	– нахождение факториала;
mod	– остаток от деления.

> abs(– 25);

25

> min(1,2,3,4,4,3,2,1);

1

> factorial(5);

120

> 12 mod 7;

5

14.1.3. Дробные

Тип fraction – дробный тип.

Дроби представляются в виде a/b, где a – целое число со знаком; b – целое число без знака.

> – 23/34;

$$\frac{-23}{34}$$

В выражении типа fraction обязательно должны присутствовать два поля – числитель и знаменатель:

> type(5/(– 8),fraction);

true

> type(34/17,fraction);

false

> type(a/b,fraction);

false

Функция op от дроби возвращает два числа – числитель и знаменатель.

> op(2/7);

2, 7

Тип rational включает в себя тип integer и тип fraction:

> type(5/(– 8),rational);

true

> type(34/17,rational);

true

> type(a/b,rational);

false

14.1.4. Числа с плавающей точкой

Тип float – числа с плавающей точкой. Тип float в среде Maple V определен следующим образом:

1. последовательность чисел, разделенных точкой:
   а) <integer>.<integer>
   б) <integer>.
   в) .<integer>
2. число может быть представлено в виде
   Float(mantissa, exponent)
   т.е. <mantissa>*10^<exponent>

Примеры:

```
> type(10.11111,float);
                                true
> type(.1234,float);
                                true
> Float(2,4);
                                20000.
```

Обратное представление числа реализуется функцией op, которая возвращает два числа – мантиссу и экспоненту:

```
> op(0.02234);
                                2234, -5
```

Если дробная или целая часть – нулевая, то нуль можно не писать.

Для приближения чисел с плавающей точкой служит команда evalf:

```
> evalf(Pi,5);
                                3.1416
```

14.1.5. Стринговские

В среде Maple V определены тип string и тип name. Выражение типа string может содержать цифры и буквы, строчные и прописные. Строка с двух сторон должна быть окружена символом ` . Если же в строку требуется вставить символ ` то его надо удвоить.

```
> var:=`It``s a string`;
```

$$var := It`s\ a\ string$$

```
> type(var,string);
```

true

```
> type(var,name);
```

true

```
> type(a*b,string);
```

false

Одиночные символы можно не заключать в кавычки

```
> char:=c;
```

$$char := c$$

```
> char:=`c`;
```

$$char := c$$

Из строки можно выделить подстроку:

```
> substring(abcdefgh,3..5);
```

$$cde$$

Определим длину строки:

```
> length(abcdef);
```

Для того, чтобы склеить несколько строк, служит команда cat:

> cat(123,abc,456);

$$123abc456$$

Можно организовать поиск в строке:

> SearchText(wx,tuvwxyz);

$$4$$

Команда SearchText различает строчные и прописные буквы:

> SearchText(Wx,tuvwxyz);

$$0$$

14.1.6. Индексные переменные

Могут использоваться везде, где возможно использование переменных типа name. Параметр вызова индексной переменной:

<имя> [<ряд выражений>]

Например, выражение

> b[1]:=10;

$$b_1 := 10$$

создает таблицу для b и значению 10 присваивается индекс 1.

Индексные переменные удобно использовать, к примеру, для нахождения сумм:

> a:=b[1]+20+b[11];

$$a := 30 + b_{11}$$

14.1.7. Области

В среде Maple V определен такой тип данных, как области (range). Синтаксис описания: выражение..выражение

Выражение типа range имеет два операнда: левое выражение и правое выражение. Между ними стоит оператор ".." . Но следует отметить, что область 1..3 не эквивалентна 1,2,3.

Примеры:

> sum(i,i=1..3);

$$6$$

> 1..3;

$$1 .. 3$$

> $1..3;

$$1, 2, 3$$

> a.(1..3);

$$a1, a2, a3$$

14.1.8. Отношения

В среде Maple V под отношением понимают выражение, разделенное специальными операторами, называемыми операторами отношения: <, <=, >, >=, =, и <> .

К примеру, в выражении m+n>=k можно вычислить количество операндов, а также извлечь из выражения его составляющие.

> f:=m+n<=k;

$$f := m + n \le k$$

Вычислим количество переменных в правой части данного выражения:

> nops(");

$$2$$

Удобны операторы rhs и lhs для выделения правой и левой частей выражения.

> y = a*x^2 + b;

$$y = a\,x^2 + b$$

> rhs(");

$$a\,x^2 + b$$

> lhs("");

$$y$$

14.1.9. Булевские выражения

Для логических операций в среде Maple предусмотрен специальный тип данных: `boolean`. Также специальные зарезервированные слова true и false используются для работы с булевскими выражениями.

В булевских выражениях можно использовать следующие операторы:
`=`, `<>`, `<`, `<=`, `and`, `or`, `not`
Для работы с булевскими выражениями предусмотрена команда evalb:

> x=x;

$$x = x$$

> evalb(");

true

> ToBe or not ToBe;

true

> Boy:=not Girl;

$$Boy := \text{not } Girl$$

> Girl and Boy;

false

14.2. Поэлементная обработка, подстановки и подвыражения

14.2.1. Поэлементная обработка

Функция map – действие над всеми элементами некоторого выражения.

Вызов функции:
map(fcn, expr, arg_2, ..., arg_n)
Параметры функции:
fcn – процедура или имя;
expr – любое выражение;
arg_i – (необязательный параметр) дополнительные аргументы.

С помощью функции map можно применить процедуру fcn к операндам выражения expr.

К примеру, найдем модуль от всех злементов списка L:

> L:=[– 1,2, – 3, – 4,5];

$$L := [-1, 2, -3, -4, 5]$$

> map(abs,L);

$$[1, 2, 3, 4, 5]$$

Возведем в квадрат все элементы этого списка:

> map(x – >x^2,L);

$$[1, 4, 9, 16, 25]$$

Проинтегрируем все элементы списка L:

> map(integrate,L,x);

$$[-x, 2x, -3x, -4x, 5x]$$

14.2.2. Подстановки

При выполнении различных математических операций возникает необходимость подставить одно выражение в другое, а также проверить полученное решение путем подстановки его в исходное равенство. Для этого служит команда subs.

Вызов:

subs(s_1,s_2,...,s_n,expr)

Параметры:

s_1,... – уравнение, или множество, или список из уравнений;

expr – любое выражение.

При этом s_1,...,s_n подставляются в выражение expr.

Пример:

> subs(x=r^(1/3), 3*x*ln(x^3));

$$3r^{1/3}\ln(r)$$

Другой *пример* использования функции subs – проверка полученного решения:

> eqs := {2*x*y = 1, x + z = 0, 2*x - 3*z = 2};

$$eqs := \{2\,x\,y = 1, x + z = 0, 2\,x - 3\,z = 2\}$$

> f:=solve(eqs,{x,y,z});

$$f := \left\{x = \frac{2}{5}, y = \frac{5}{4}, z = \frac{-2}{5}\right\}$$

Проверим, правильно ли мы решили эту систему:

> subs(f,eqs);

$$\{2 = 2, 1 = 1, 0 = 0\}$$

14.2.3. Подвыражения

В среде Maple V возможны различные операции над выражениями, в том числе и выделение подвыражения. Например, команда op.

Вызов команды:

op(i,e) op(i..j,e) op(e)

Параметры:

i,j – положительные целые числа, определяющие позицию операнда в выражении;

e – любое выражение.

> f:=[x,y,z];

$$f := [x, y, z]$$

> op(3,f);

$$z$$

Для того, чтобы заменить некоторый операнд в выражении, служит команда subsop.

Вызов команды:

subsop(eq1, eq2, ..., eqN, expr)

Параметры:

eqI – выражение (необязательное) вида:
<numI> = <exprI>;

<numI> – положительное целое;
<exprI> – выражение;
expr – выражение.

Заменим в определенном нами ранее списке f=[x,y,z] третий элемент этого списка на v:

> subsop(3=v,f);

$$[x, y, v]$$

Удалим его:

> subsop(3=NULL,f);

$$[x, y]$$

14.3. Определение типов в Maple

Для проверки принадлежности определенного выражения конкретному типу служит функция type.

Вызов функции:

type(x, t)

Параметры:

x – любое выражение;
t – имя типа или множество имен типов.

Функция type – булевская функция, которая возвращает значение true, если x принадлежит типу t,и false – в противном случае.

В том случае, если t – множество типов, то возвращаемое функцией type значение будет true, если x принадлежит хотя бы одному типу из множества.

Примеры:

> type(c+d,polynom);

true

> type(5,{fraction,integer});

true

> type(99/55,float);

false

Для определения типов в Maple служит функция whattype.

Вызов функции:

whattype(expr)

Здесь expr – любое выражение.

Функция возвращает имя типа данных выражения, которое может быть одним из следующих:

*	+	.	..	<
<=	<>	=	^	and
array	exprseq	float	fraction	function
indexed	integer	list	not	or
procedure	series	set	string	table
uneval				

Примеры:

> whattype(0.5);

float

> whattype(f);

string

> whattype([x,y]);

list

Функция hastype – проверка на указанный тип. Вызов функции:

hastype(expr, t);

Параметры:

expr – любое выражение;

t – имя типа данных.

Булевская функция hastype возвращает значение true тогда и только тогда, когда выражение expr имеет подвыражение типа t.

Примеры:

```
> hastype( (x+y),`+` );
```

true

```
> hastype( (x+y)*(1/2*x+y), fraction );
```

true

```
> hastype( x+7*y, fraction );
```

false

14.3.1. Простые типы

К простым типам в Maple можно отнести: системный и процедурный типы, константы.

К системным типам в Maple можно отнести все типы, описанные в п. 10.1. Определение принадлежности к системным типам было рассмотрено в п. 10.3.

Для определения принадлежности выражения к процедурному типу служит функция type/procedure.

Формат вызова функции:

type(expr,procedure)

Здесь expr – любое выражение.

Эта команда проверяет принадлежность выражения expr к процедурному типу.

```
> type(sin,procedure);
```

true

```
> f:=proc(x) x^2 end;
```

$f := proc(x)\ x\hat{}2\ end$

```
> type(f,procedure);
```

true

> type(x^2,procedure);

false

Для того чтобы определить, является ли заданное выражение константой, служит команда type/constant.

Формат вызова команды: type(x,constant);

Здесь x – любое выражение.

Примеры:

> type(7, constant);

true

> type(infinity,constant);

true

> type(x^4,constant);

false

14.3.2. Структурные типы

Для объяснения понятия структурного типа в Maple рассмотрим такой пример. Введем множество L={1,2,3}:

> L:={1,2,3};

$$L := \{1, 2, 3\}$$

Определим, к какому типу принадлежит L:

> whattype(L);

set

Однако множество может состоять не только из целых чисел, но и из дробных и даже из строк. Чтобы узнать, из чего же состоит множество L, воспользуемся такой командой:

```
> type(L,set(integer));
```

true

Теперь мы уверены, что L – множество из целых чисел, а не из каких-либо других:

```
> type(L,set(fraction));
```

false

Рассмотрим другие *примеры*:

```
> type(f(1,2,3),function(integer));
```

true

```
> type(y^(-2),name^integer);
```

true

```
> type([y,1],list(integer));
```

false

```
> type([y,1],list({name,integer}));
```

true

14.4. Преобразование типов

Для преобразования выражения из одного типа в другой служит функция convert.

Вызов функции (в общем виде):

convert(expr, form, arg3, ...)

Параметры:

expr – любое выражение;

form – имя;
arg3, ... – другие аргументы (необязательный параметр).

Примеры:

Преобразуем число 1.2345 в дробь (т.е. в тип fraction):

```
> convert(1.2345,fraction);
```

$$\frac{2469}{2000}$$

Преобразуем обратно:

```
> convert(2469/2000,float);
```

1.234500000

Можно преобразовывать числа в двоичную форму:

```
> convert(12,binary);
```

1100

```
> convert( – 1.234,binary);
```

-1.001110111

Преобразуем двоичное число 1100 в десятичную нотацию:

```
> convert(1100,decimal,binary);
```

12

Для преобразования радиан в градусы служит следующая команда:

```
> convert(Pi/2,degrees);
```

90 *degrees*

Градусы в радианы:

```
> convert(90*degrees,radians);
```

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

Для преобразования в список служит команда convert/list. Причем преобразуемое выражение должно принадлежать одному из следующих типов: таблице, вектору, множеству или выражению.

Вызов:

convert(<table>, list)
convert(<vector>, list)
convert(<expression>, list)

```
> A:= array([x,y,z]):
> convert(A,list);
```

$$[x, y, z]$$

```
> convert(r+s – t,list);
```

$$[r, s, -t]$$

```
> B := table([(3)=123,(7)=456]):
> convert(B,list);
```

$$[123, 456]$$

```
> convert({x,y},list);
```

$$[y, x]$$

Для преобразования в список из списков служит команда convert/listlist. Причем преобразуемое выражение должно быть массивом или списком.

Вызов:

convert(<array>, listlist)
convert(<list>, listlist)

```
> A := array([[1,2],[3,4]]):

> convert(A,listlist);
```

$$[[1, 2], [3, 4]]$$

```
> B := [4=12,2=7,1=8,3=6]:

> convert(B,listlist);
```

$$[8, 7, 6, 12]$$

Преобразование во множество осуществляется командой convert/set. Преобразуемое выражение должно принадлежать одному из следующих типов: таблице, массиву, списку или выражению.

Вызов функции:

convert(<table>, set)
convert(<array>, set)
convert(<expr>, set)

```
> A := array(1..2,1..2,[[1,2],[3,4]]):
> convert(A,set);
```

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

```
> convert(x+y – z,set);
```

$$\{-z, y, x\}$$

Для преобразования в таблицу служит команда convert(object, table), где object – список или массив.

```
> convert([[1,2],[3,4]],table);
```

```
table([
        (1, 1) = 1
        (2, 1) = 3
        (2, 2) = 4
        (1, 2) = 2
        ])
```

> convert(linalg[vector](3,[1,x,x^2]),table);

table([
$2 = x$
$3 = x^2$
$1 = 1$
])

> A := linalg[randmatrix](2,2,symmetric):

> convert(A,table);

table(*symmetric*, [
$(1, 1) = 79$
$(2, 2) = 49$
$(1, 2) = 56$
])

Для преобразования выражения в строку служит команда convert/string.

> convert(sin(x^2),string);

sin((x)^(2))

> convert(a+b+c*d,string);

(a)+(b)+((c)(d))*

15. МАССИВЫ И ТАБЛИЦЫ

15.1. Создание таблиц

Таблица – конечномерный массив, у которого индексы любые значения.

Таблица создается указанием слова "table":

> table( );

```
table([
      ])
```

Команда table( ) создала таблицу с неопределенными значениями.

Определим значения таблицы:

> table([22,red,Pi]);

```
table([
      1 = 22
      2 = red
      3 = π
      ])
```

Так как индексы таблицы не были определены, то программа сама присвоила им целочисленные значения, расположенные по порядку.

Зададим индексы таблицы следующим образом:

> S := table([(2)=45,(4)=61]);

```
S := table([
      2 = 45
      4 = 61
      ])
```

Обратимся к элементам таблицы, один из которых не определен:

> S[1],S[2];

$$S_1, 45$$

Для распечатки индексов и значений таблицы служат функции indises и entries. Например, распечатаем индексы таблицы F:

> indices(F);

$$[def], [abc]$$

Значения таблицы F:

> entries(F);

$$[-\sin], [\cos]$$

15.2. Индексные функции

Можно создавать определенные типы таблиц с помощью индексных функций: symmetric, antisymmetric, sparse, diagonal, identity – по формату:

table(<индексная функция>,<индексная функция...>)

Если <индексная функция> определено как <имя> которое не принадлежит одному из пяти перечисленных, то имя `index/`. <имя> будет использоваться, как имя процедуры, которая определяет индексную функцию.

Пример, приведенный ниже,создает индексную функцию, которая возвращает заначение infinity для всех несвязанных со значениями индексов таблицы (похоже на индексную функцию sparse, которая возвращает нуль).

Пример:

```
> `index/infinity` := proc(indices,table)
>       if (nargs = 2) then
>               if (assigned(table[op(indices)])) then table[op(indices)];
>               else infinity;
>               fi;
>       else table[op(indices)] := args[3..nargs];
>       fi;
> end:

> t := table(infinity):

> t[1,2] := 1:

> t[1,2];

                                1

> t[2,1];

                                ∞

> t := table(symmetric,infinity):

> t[1,2] := 1:

> t[1,2];

                                1

> t[2,1];

                                1

> t[33,44];

                                ∞
```

16. ПРОЦЕДУРЫ

16.1. Определение и вызов процедур

Синтаксис определения процедуры следующий:
proc (<argseq>) local <nseq>; global <nseq>; options <nseq>;
description <string>; <statseq> end

Параметры:

<argseq> – список формальных параметров. Параметры в списке разделяются запятой. Список может быть пустым;

local, global, options – необязательные параметры, которые определяют глобальные и локальные переменные, а также режимы работы;

description – комментарий, описание процедуры.

Просмотреть параметры процедуры можно с помощью функции op.
op(<num>,eval(<имя процедуры>), где num – число (параметр).
1 – возвращает список формальных параметров;
2 – возвращает список локальных переменных;
3 – возвращает список ключей из раздела options;
4 – возвращает таблицу памяти процедуры;
5 – возвращает строку описания description;
6 – возвращает список глобальных переменных.

Любой возвращаемый список может быть пустым.

Пример описания и вызова процедуры :

```
> power:=proc(a,b)
> description `возведение в степень`;
>          a^b
> end;
```

power := proc(a,b) a^b end

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

$$8$$

```
> op(1,eval(power));
```

$$a, b$$

Процедура возвращает значение последнего выполненного оператора или значение, определенное процедурой RETURN.

```
> one:=proc(t)
>     if t<0 then RETURN(0) else RETURN(1) fi
> end:
> one(1); one( – 6);
```

$$1$$

$$0$$

16.2. Локальные переменные

Как и в процедурах Pascal, в Maple существуют локальные и глобальные переменные. Они определяются при описании процедуры после ключевых слов global и local. Локальные переменные видимы только в пределах процедуры, глобальные переменные видимы везде.

Рекомендуется все переменные, используемые процедурой, описывать в секциях local и global. Если тип переменных явно не определен, то существует правило определения типа переменной:

- если переменной внутри процедуры присвоено какое-либо значение, то эта переменная считается локальной;
- переменные, используемые в циклах for и seq, также считаются локальными.
- остальные переменные считаются глобальными.

```
> s:=proc(x)
>    y:=2^x;
>    RETURN(y)
> end:
```

Warning, `y` is implicitly declared local

```
> a:=proc(x)
> global z;
>    x+z
> end:

> z:=0: a(5);
```

5

```
> z:=6: a(5);
```

11

16.3. Расширяющие ключи

Расширяющие ключи записываются в секции options.
Ключи могут быть следующими:

remember	– определяет таблицу памяти для процедуры;
builtin	– определяет функцию как встроенную;
system	– определяет "системную" процедуру;
operator	– объявляет процедуру – функциональный оператор;
arrow	– определяет процедуру – оператор в нотации "->";
angle	– определяет процедуру – оператор в нотации "<..|..>";

trace – определяет, что будет выводиться информация о выполнении;
package – определяет пакетную процедуру;
Copyright – авторские права.

Ключ remember

Определяет, что все результаты обращений к процедуре будут заноситься в таблицу памяти процедуры.

```
> f:=proc(t)
> options remember;
> t^2
> end:

> f(4):
                                16
> f(6);
                                36
```

Просмотреть таблицу памяти можно с помощью следующей команды:

```
> op(4,eval(f));
                        table([
                            6 = 36
                            4 = 16
                            ])
```

Рассмотрим пример процедуры, вычисляющей числа Фибоначчи:

```
> f := proc(n) if n<2 then n else f(n – 1)+f(n – 2) fi end:
> f(14);
```

377

Затраты времени возрастают по экспоненциальной зависимости:

```
> f := proc(n) option remember;
>       if n<2 then n else f(n – 1)+f(n – 2) fi end:

> f(14);
```

377

В последнем случае затраты времени возрастают по линейной зависимости, так как предыдущие значения запоминаются в таблице памяти.

С таблицей памяти можно работать как с обычной таблицей. Допустимо присваивание:

```
> f(0):=0;
```

$$f(0) := 0$$

Ключ builtin

Определяет процедуру как встроенную. Ключ builtin должен быть первым в списке options.

Например, функция type – встроенная функция.

```
> eval(type):
```

proc( ) options builtin; 145 end

Результат выполнения eval показывает, что type – встроенная функция. Число 145 – системный номер функции.

Ключ system.

Определяет "системную" процедуру. "Системная" процедура – это процедура, таблица памяти которой может быть удалена из "мусорного ящика" (garbage collector) системы. Если процедура описана без ключа system, то нельзя будет удалить ее таблицу памяти.

Ключ operator

Определяет, что с процедурой можно работать как с оператором.

Пример функционального оператора:

```
> sq:=<x^2|x>;
```

$$sq := \langle x^2 \mid x \rangle$$

```
> oper:=proc(x,y)
> option operator;
>   x^2 – y
> end;
```

$$oper := \langle x^2 - y \mid x, y \rangle$$

```
> oper(4,1);
```

15

Процедура oper аналогична записи:

```
> oper:=<x^2 – y|x,y>;
```

$$oper := \langle x^2 - y \mid x, y \rangle$$

Ключ arrow

Процедура – оператор будет записана в нотации "->".

Нотация "->" - это форма записи оператора. Ключ Arrow изменяет правило определения глобальных и локальных переменных: все не локальные переменные не будут добавлены в список параметров. Ключ arrow используется вместе с operator.

```
> oper1:=proc(x,y)
> options operator,arrow;
>   x^2 - y
> end;
```

$$oper1 := (x, y) \rightarrow x^2 - y$$

```
> oper1(4,1);
```

15

Oper1 эквивалентно записи:

```
> oper1:=(x,y) - >x^2 - y;
```

$$oper1 := (x, y) \rightarrow x^2 - y$$

Ключ angle

Объявляет процедуру – оператор в нотации "< >". Действует вместе с ключом operator (по умолчанию).

```
> oper2:=proc(x,y)
> options operator,angle;
>   x^2 - y
> end;
```

$$oper2 := \langle x^2 - y \mid x, y \rangle$$

Ключ trace

Определяет, что при выполнении процедуры будет выводиться отладочная информация.

Не подлежат трассировке процедуры, содержащие функции: assigned, eval, evalhf, evalf evaln, traperror, seq, userinfo.

```
> pro:=proc(x,y,z)
> local t,k,i;
> options trace;
>   t:=x*y;
>   k:=y+z;
>   i:=x*z;
>   t+k+i;
> end:

> pro(1,2,3);
   {--> enter pro, args = 1, 2, 3

                                   t := 2
                                   k := 5
                                   i := 3
                                    10
   <-- exit pro (now at top level) = 10}
                                    10
```

Аналогичные результаты дает функция trace(<имена процедур>).

```
> ff:=proc(t)
>   t^3
> end;
   ff := proc(t) t^3 end
```

```
> ff(4);
                                    64
> trace(ff):

> ff(4);
   {--> enter ff, args = 4

                                    64
   <-- exit ff (now at top level) = 64}
                                    64
```

Выключить режим трассировки можно с помощью untrace.

```
> untrace(ff):
> ff(4);
                                    64
```

Ключ package

Создает библиотечную процедуру. Пример приведен ниже.
Примечание: библиотечные процедуры загружаются оператором with.

```
> # A package defined by a procedure
> linalg2 := proc() local s,t; option package;
>     t := linalg2(_PackageTable):
>    eval(t[eval(op(procname),1)](args));
> end:


> linalg2(_PackageTable) := eval(linalg)
```

```
> with(linalg2,add);
```

[*add*]

Ключ Copyright

Защищает процедуру от просмотра. При просмотре защищенной процедуры ее текст выводиться не будет. Отменить действие ключа Copyright можно командой interface(verboseproc=2).

```
> myproc:=proc(y)
> option Copyright;
>   y+4
> end:

> myproc(4);
```

8

```
> eval(myproc);
```

proc(y) ... end

16.4. Присвоение значений параметрам

При описании процедуры указывается список ее формальных параметров. При обращении к процедуре указывается список действительных параметров. В следующем примере в процедуре f формальный параметр – a, действительный параметр – 2. При вызове процедуры действительные параметры присваиваются формальным.

```
> f:=proc(a)
>  a^2
> end:


> f(2);
```

4

При передаче действительных параметров в процедуру проверяется совместимость типов формальных и действительных параметров. В случае их несовпадения генерируется ошибка.

Действительные параметры перед передачей в процедуру предварительно вычисляются.

```
> f(6 – 4);
```

4

Количество передаваемых в процедуру действительных параметров может быть не равным количеству формальных параметров (но не меньше количества формальных параметров).

```
> f(2,34,56,7);
```

4

16.5. Сообщения об ошибках и завершение процессов

Выполнение процедуры можно прервать оператором ERROR. В этом случае будет выдано сообщение "Error, <in имя процедуры> < список параметров ERROR>".

Синтаксис ERROR: ERROR(список параметров).

```
> f := proc (x) if x<0 then ERROR(`invalid x`, x) else x^(1/2) fi end:
```

```
> f(2);
```

$$\sqrt{2}$$

```
> f( – 2);
```

Error, (in f) invalid x, -2

Если вызов процедуры произошел в режиме traperror(..), то список параметров ERROR присваивается переменной lasterror.

```
> traperror(f( – 2)):
> if '' = lasterror then `error occurred` else `no error occurred` fi;
```

error occurred

Процедура после выполнения действий возвращает значение последнего выполненного оператора. Это правило можно изменить при помощи RETURN.

RETURN прерывает выполнение процедуры и возвращает список значений.

Синтаксис: RETURN(<список параметров>).

```
> delta:=proc(t)
>  if t=0 then RETURN(infinity) else RETURN(0) fi
> end:
> delta(1); delta(0);
```

0

∞

С помощью RETURN можно передавать не только численные значения.

```
> g:=proc(t);
>  RETURN('procname(args)')
> end:
```

```
> g(4);
```

g(4)

Ключевое слово FAIL предназначено для прерывания процедуры. FAIL используется для сообщения о том, что выражение нельзя вычислить.

```
> h:=proc(t)
>   FAIL
> end:


> h(5);
```

FAIL

16.6. Булевские процедуры

Булевские процедуры определяются так же, как и обычные.

```
> g:=proc(x,y)
>   x or y
> end:


> g(true,false);
```

true

```
> comp:=proc(x,y)
>     if x>y then true else false fi
> end;
```

comp := proc(x,y) if y < x then true else false fi end

```
> if comp(3, – 2) then 1 else 0 fi;
```

1

16.7. Вызов и сохранение процедур в файлах на диске

Запись процедур в файл:
save <список имен>,<имя файла>
Чтение процедур из файла:
read <имя файла>

```
> f1:=proc(x,y)
>  x^2+y^2
> end:

> f2:=proc(x,y)
>  x^3+y^3
> end:

> save f1,f2,profile;

> read profile;
```

f1 := proc(x,y) x^2+y^2 end
f2 := proc(x,y) x^3+y^3 end

Процедуры можно записать в файл во внутреннем формате Maple. Для этого просто надо добавить к имени файла расширение ".m".

```
> save f1,`pro.m`;
> read `pro.m`;
```

Кроме этих возможностей сохранения процедур Maple имеет большие возможности создания библиотек.

17. ОПЕРАТОРЫ

В Maple операторы - это абстрактный тип данных, описывающий операции преобразования над данными.

17.1. Определение операторов в Maple

Варианты описания операторов:
- неопределенный (f);
- нейтральный (определяемый пользователем) (&);
- процедурный;
- стрелочная нотация;
- нотация угловых скобок;
- композиционный оператор (@) .

Неопределенный оператор – это оператор, который заранее не был определен. Такой оператор не выполняет никаких действий.

```
> f(3,4,5);
```

$$f(3, 4, 5)$$

Нейтральный оператор (определяемый пользователем) обозначается значком амперсанта &.

Синтаксис описания оператора: & имя.

Имя – это любое разрешенное в Maple имя или набор специальных символов.

В имя оператора нельзя включать цифры и следующие символы:

& | ( ) [ ] { } ; : ' ` # <ENTER> <SPACE>_

Максимальная длина имени - 495 символов.

&-операторы можно использовать как унарный префиксный оператор или как инфиксный бинарный. Эти операторы генерируют функциональные вызовы с именем нейтрального оператора.

```
> a &* b;
```

$$a \,\&*\, b$$

```
> a * b &+ c;
```

$$a\,(b\ \&{+}\ c)$$

```
> a &xx b - &xx(a,b);
```

$$0$$

Оператор также можно описать через процедуру.

```
> fg:=proc(x)
> option operator;
>   x^2
> end;
```

$$fg := \langle x^2 \mid x \rangle$$

```
> fg(5);
```

$$25$$

В Maple можно определять функциональные операторы. Функциональные операторы – это форма представления пользовательских математических функций. Определить функциональный оператор можно в "стрелочной нотации". Синтаксис стрелочной нотации:

(список переменных функции)->(функция)

```
> fg:=x - >(x^2);
```

$$fg := x \rightarrow x^2$$

```
> fg(4);
```

$$16$$

```
> df:=(x,y,z) - >(x+y+z);
```

$$df := \langle x+y+z \mid x, y, z \rangle$$

> df(1,2,3);

$$6$$

Функциональный оператор можно определить в нотации "угловых скобок".

Синтаксис такой нотации: < функция | список переменных>

> fg:=<x^2|x>;

$$fg := \langle x^2 \mid x \rangle$$

> fg(4);

$$16$$

> df:=<x+y+z|x,y,z>;

$$df := \langle x+y+z \mid x, y, z \rangle$$

> df(1,2,3);

$$6$$

Можно определить @ - композиционный оператор:

> f:=cos@sin;

$$f := \cos@\sin$$

> (cos@sin) (x);

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

> (cos@@2)(x);

$$\cos^{(2)}(x)$$

Определение операторов с помощью define

Оператор define реализует определение оператора и его свойств. Define позволяет определить оператор в двух нотациях:

- инфиксной (& - имя);
- функциональной (f(x)).

Синтаксис: define(f,<список свойств>),
где f - имя оператора, реализующего вычисления.

Свойства операторов:

unary	– унарный оператор;
binary	– бинарный оператор;
associative	– ассоциативный оператор; f(x,f(y,z))=f(f(x,y),z)= f(x,y,z).
commutative или symmetric	– коммутативный оператор, f(x,y)=f(y,x);
antisymmetric	– асимметричный оператор, f(x,y) = -f(y,x);
inverse=g	– определяет инверсный оператор, (f ->g);
zero=x	– определение "нуля" оператора. Если любой из аргументов оператора равен x, то оператор возвращает x.

> define(f,binary,associative,zero=1);

Warning: new definition for f

Просмотреть текст процедуры можно с помощью print.

> print(f);

```
proc( )
local _AA,i,j,Sign;
  _AA := [args];
  _AA := map(
   proc(x,pn)
     if type(x,function) and op(0,x) = pn then
       op(x)
     else x
     fi
   end
   ,_AA,procname);
  i := 1;
  j := nops(_AA);
  while i < j do
```

```
    if j <= 2 then break fi;
    [procname(_AA[i],_AA[i+1])];
    if has(",procname) then i := i+1
    elif false then
      _AA := subsop(i = NULL,i+1 = NULL,_AA);
      if 1 < i then i := i-1 fi
    else
      _AA := subsop(i = op(''),i+1 = NULL,_AA)

      if j <= nops(_AA) then
        ERROR(
         `A forall( ) with two arguments ret\
        urns an expression sequence`

      fi;
      if 1 < i then i := i-1 fi
    fi;
    j := nops(_AA)
  od;
  if member(1,_AA) then RETURN(1) fi;
  if nops(_AA) = 1 then RETURN(_AA[1]) fi;
  'procname'(op(_AA))
end
```

> f(2,1,4);

$$1$$

> f(f(x,y),z);

$$f(x, y, z)$$

> f(34,45);

$$f(34, 45)$$

> define(`&+`, associative, commutative, identity=0);

Warning: new definition for &+

```
> x &+ (y &+ z) &+ (0 &+ x);
```

$$\&+(x, x, y, z)$$

```
> define(f, commutative, associative, inverse=g);
```

Warning: new definition for f

```
> f( g(x), z, f(x,y), y );
```

$$f(y, y, z)$$

18. ВНЕШНИЕ ВЫЗОВЫ И МАНИПУЛЯЦИИ

Maple предоставляет возможность вызова других программ и команд операционной системы. Эта функция работает в среде Maple, установленной под DOS. Реализует эту функция команда system.

Синтаксис system: system (<строка>);
где <строка> – команда, передаваемая в DOS.

Например, если выполнить команду system (dir); то будет выведено содержимое текущего каталога. С помощью system можно организовать обмен данными с другими программами.

Рассмотрим простой пример внешнего вызова. Например, на Pascal написана программа, реализующая численный метод (для простоты пусть это будет программа перемножения двух чисел). DOS позволяет в командной строке определить файл, из которого будут считаны исходные данные, и файл, в который будут записаны результаты. Например, c:\prog.exe < input.txt > output.txt реализует сказанное выше. В этой команде DOS файл prog.exe – запускаемая программа, файл input.txt – файл исходных данных, файл output.txt – файл результатов.

Чтобы организовать такой обмен в Maple, надо создать файл исходных данных. Это можно сделать стандартными средствами Maple. Затем выполняем внешний вызов: system(`prog.exe < input.txt > output.txt`). После этого результаты работы программы prog.exe можно считать из файла output.txt.

Описанный способ внешнего вызова не рациональный, так как обмен данными идет через текстовый файл.

19. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ ЯЗЫКА MAPLE

19.1. Отладка и синтаксис сообщений об ошибках

Наиболее простой способ отладки программ на языке Maple V – это отладка с помощью printlevel. Printlevel – это глобальная переменная, начальное значение которой равно единице. Если установить ее значение больше чем 1, то будет распечатываться пошаговое выполнение введенной процедуры, причем, чем больше это значение, тем больше шагов будет выводиться. Например, введем две процедуры, в одной из которых есть ошибка – деление на нуль:

```
> f:=proc(x) local y; y:=1; g(x,y); end:

> g:=proc(u,v) local s,t; s:=0; t:=v/s; s+t end:
```

Вызовем процедуру f:

```
> f(3);
```

Error, (in g) division by zero

Установим printlevel=100:

```
> printlevel:=100:

> f(3);
```

{--> enter f, args = 3

$$y := 1$$

{--> enter g, args = 3, 1

$$s := 0$$

<-- ERROR in g (now in f) = division by zero}
<-- ERROR in f (now at top level) = division by zero}
Error, (in g) division by zero
executing statement: t := v/s
locals defined as: s = 0, t = t
g called with arguments: 3, 1
f called with arguments: 3

19.2. Переназначения и макросы

Функция alias – определить сокращение или обозначение.
Формат вызова:
alias(e1, e2, ..., eN)
Параметры:
e1, e2, ..., eN – нуль или более выражений.

В математике очень много различных длинных имен, названия которых были перенесены и в среду Maple V. Чтобы не использовать эти длинные имена, можно определить сокращения для них. Для этого используется команда alias. Пример: определим сокращенное название для стандартной функции среды Maple V fibonacci (возвращает числа Фибоначчи):

> alias(F=fibonacci);

$$I, F$$

> with(combinat):

> F(6);

```
> fibonacci(6);
```

$$8$$

Можно также использовать функцию alias для сокращения названий, определенных пользователем. Например, определим сокращения: F для Func(x), Fx для производной от Func(x) по x.

```
> alias(F=Func(x)):

> alias(Fx=diff(Func(x),x)):

> diff(F,x);
```

$$Fx$$

```
> diff(Fx,x);
```

$$\frac{\partial}{\partial x} Fx$$

Функция macro – определение макрообозначений.

Например, предположим, что в одной и той же процедуре используется длинное обозначение (например combinat[fibonacci] – вызов функции fibonacci) один или несколько раз:

```
> f:=proc(a);
> f:=combinat[fibonacci](a);
> end:
```

Warning, `f` is implicitly declared local

```
> seq(f(i),i=1..10);
```

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55$$

Чтобы не использовать это длинное имя, определим макрообозначение:

```
> macro(Fib=combinat[fibonacci]);
```

```
> g:=proc(a);
>  g:=Fib(a);
> end:
```

Warning, `g` is implicitly declared local

```
> seq(g(i),i=1..10);
```

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55$$

Еще один пример использования macro:

```
> macro(v = linalg[vandermonde]);
```

```
> v([3,2,1]);
```

$$\begin{bmatrix} 1 & 3 & 9 \\ 1 & 2 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

Функцию macro можно использовать для создания такого привычного для Pascal типа данных, как запись. Для понимания сущности создания записи в среде Maple рассмотрим пример:

```
> a:=[x,y,z];
```

$$a := [x, y, z]$$

```
> macro(one=1,two=2,three=3);
```

```
> a[1];
```

$$x$$

```
> a[one];
```

$$x$$

```
> s:=a[one]+a[two]+a[three];
```

$$s := x + y + z$$

Таким образом, к элементам списка можно обращаться традиционно по индексам или введенным макросам.

Рассмотрим более сложный пример – запись выражения

$$-0.5\,(x+1)^2(x-1)$$

```
> a:=[ – 1/2,[[x+1,2],[x – 1,1]]];
```

$$a := \left[\frac{-1}{2}, [[x+1, 2], [x-1, 1]]\right]$$

```
> macro(unit=1,factors=2,base=1,exponent=2);
```

```
> a[unit];
```

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

```
> a[factors][1][base];
```

$$x + 1$$

```
> a[unit]*a[factors][1][base]^a[factors][1][exponent]*a[2][2][1]^a[2][2][2];
```

$$-\frac{1}{2}(x+1)^2(x-1)$$

20. ОБЗОР БИБЛИОТЕК MAPLE V

Библиотеки Maple V содержат более 95 % кода математических знаний и экспертных решений. Ее процедуры могут быть просмотрены, изучены и даже модифицированы и расширены.

Информацию по библиотекам Maple всегда можно получить из справочной системы среды, которая содержит подробную информацию о каждой библиотеке. Для просмотра достаточно ввести вопросительный знак и рядом имя библиотеки. Например: >?linalg.

В состав среды Maple V входят следующие библиотеки:

- numapprox – численная аппроксимация;
- combinat – комбинаторика;
- Detools – средства для работы с дифференциальными уравнениями;
- difforms – средства для работы с различными дифференциальными формами;
- geom3d – трехмерная евклидова геометрия;
- geometry – двухмерная евклидова геометрия;
- grobner – базис Гробнера;
- group – теория групп;
- linalg – линейная алгебра;
- logic – булевская логика;
- networks – теория графов;
- np – теория размещений;
- numtheory – теория чисел;
- orthopoly – ортогональные полиномы;
- plots – графическая библиотека;
- powseries – степенные ряды;
- projgeom – проекционная геометрия;
- simplex – линейная оптимизация;
- stats – статистика;
- student – оценка ошибок.

Обратим особое внимание на свободно распространяемые коды библиотек и готовых проектов математических и физических задач через информационный сервер университета г. Ватерлоо в Канаде. Наличие связи INTERNET снимает все проблемы к доступу. Адреса указаны в текстовых файлах в каталоге SHARE/HELP. Пользователь может там же найти подкаталоги по 20 разделам науки и техники, в которых находится богатейший набор программ для различных задач.

ЛИТЕРАТУРА

1. *Robertson J. S.*, Engineering Mathematics with Maple. (International Series in Pure & Applied Mathematics), McGraw-Hill Companies, 1996
2. *Nicolaides R. A., Walkington N.J.* Maple: A Comprehensive Introduction. Cambridge University Press, 1996
3. *Blachman N.*, Practical Guide to Maple, Version 2. Brooks Cole Publishing Company, 1996
4. *Braselton, Sparks and Davenport*. Calculus Labs Using Maple V. HarperCollins College. HarperCollins College , 1995
5. *Greene R. L.* Classical Mechanics with Maple. Springer-Verlag New York, Incorporated, 1995
6. *Beltzer A*. Engineering Analysis with Maple Mathematica. Academic Press Inc., Australia. 1995
7. *Char B. W., Geddes K. O., Gonnet G. H., Leong B. L., Monagan M. B. and Watt S. M.* First Leaves: Tutorial Introduction to Maple V. Springer-Verlag New York, 1995
8. *Burkhardt W.* First Steps in Maple. Springer-Verlag New York, 1995
9. *Kamerich K.* Guide to Maple. Springer-Verlag New York, 1995
10. *Heck.* Introduction to Maple. Springer-Verlag New York, 1995
11. *Redfern.* Maple V Handbook: Maple V Release 4. Springer-Verlag New York, 1995
12. Waterloo, Maple V Mathematics Learning Guide. Springer-Verlag New York, 1995
13. *Манзон.Б.М.* Maple – программа не только для математиков. Мир ПК. 1995, №12

Список терминов Maple V, использованных в книге

A

abs – абсолютное значение числа;
addedge – добавление ребра в граф;
addvertex – добавление вершины в граф;
adjacency – генерация матрицы смежности для графа;
alias – определение сокращения;
allvalues – вычисление всех возможных значений в выражении с RootOf;
and – логическое "И";
angle – определяет процедуру – оператор в нотации "<...|..>";
animate – анимация графиков;
antisymmetric – индексная функция – несимметричная;
appendto – дозапись результатов в существующий файл;
array – создание массива;
arrow – определяет процедуру – оператор в нотации "->";

B

binary – двоичный тип данных;
break – выход из цикла;
builtin – определяет функцию как встроенную;
by – приращение счетчика цикла;

C

C – перевод выражения в код языка C;
cat – вырезать выражение;
charpoly – нахождение характеристического многочлена матрицы;
close – закрытие файла;
complete – создает полный граф;

cond – число обусловленности матрицы;
conformal – построение комплексной функции;
constant – тип данных – константы;
convert – преобразование типов;
Copyright – авторские права на процедуру;

D

D – оператор дифференцирования;
Dchangevar – подстановка новых переменных в уравнение;
define – определение оператора и его свойств;
delete – удаление вершин и ребер из графа;
DEplot – построение решений дифференциальных уравнений и систем;
DEplot1 – построение решения уравнения первого порядка;
DEplot2 – построение решения системы из двух уравнений;
description – секция описания процедуры;
det – вычисление определителя матрицы;
DEtools – графическая библиотека;
dfieldplot – построение поля решения дифференциального уравнения;
diagonal – индексная функция - диагональная;
diff – дифференцирование;
display3d – построение графика 3D по специальной структуре данных;
do – начало тела цикла;
draw – построение чертежа графа;
dsolve – решение дифференциальных уравнений;
duplicate – создание копии графа;

E

edges – список ребер графа;
eigenvals – вычисление собственных значений матрицы;
eigenvects – вычисление собственных векторов матрицы;

elif – конструкция “else – if”;
else – конструкция “иначе”;
end – завершение описания тела процедуры;
ends – список “хвостов” ребер графа;
entries – значения таблицы;
ERROR – завершение выполнения процедуры с сообщением об ошибке;
eval – точное вычисление выраженеия;
evalb – вычисление логического выражения;
evalf – вычисление выражения в числах с плавающей запятой;
evalm – вычисление матричного выражения;
eweight – нахождение весов ребер графа;

F

factorial – вычисление факториала;
FAIL – прерывание процедуры в случае невычисляемого выражения;
false – “ложь” – зарезервированная константа;
fi – окончание конструкции ветвления if;
float – тип данных – число с плавающей запятой;
flow – нахождение максимального потока в графе;
for – конструкция цикла “для”;
fortran – трансляция выражения в код языка Fortran;
fraction – тип данных – дробь;
from – начальное значение счетчика цикла;
fsolve – решение уравнения в числах с плавающей запятой;

H

hastype – проверка на указанный тип;
head – нахождение “голов” ребер графа;

I

I – мнимая единица (зарезервированная константа);
identity – индексная функция – единичная;

if – конструкция ветвления “если”;
igcd – наибольший общий делитель;
ilcm – наименьший общий множитель;
in – конструкция цикла для перечисляемых типов данных;
incidence – определение матрицы инцидентности графа;
infinity – “бесконечность” – зарезервированная константа;
int – вычисление интеграла;
interface – установка интерфейсных переменных;
intersect – пересечение множеств;
inverse – нахождение обратной матрицы;
iquo – частное;
irem – остаток;
isplanar – проверка планарности графа;
isqrt – квадратный корень (целочисленное приближение);

J

jacobian – вычисление якобиана от вектора функций;

L

latex – вывод выражения в редактор LATEX;
length – определение длины выражения;
lhs – выделение левой части выражения;
limit – вычисление предела;
linalg – библиотека линейной алгебры;
local – секция описания локальных переменных процедуры;

M

macro – определение макрообозначений;
map – задание операции над всеми элементами выражения;
matrix – задание матрицы;
max – нахождение максимального элемента;
member – принадлежность элемента множеству;

min – нахождение минимального элемента;
minor – распечатка минора матрицы;
minus – вычитание множеств;
mod – остаток от деления;
multiply – умножение матриц;

N

new – создание пустого графа;
nops – подсчет количества элементов;

O

op – извлечение элементов из выражения;
open – открытие файла;

P

petersen – создание графа Петерсена;
phaseportrait – построение фазового портрета;
plot – построение графика;
plot3d – построение графиков 3D;
print – распечатка содержимого;
printlevel – глобальная переменная, используется для отладки процедур;
proc – задание процедуры;

R

rank – нахождение ранга матрицы;
random – генерация случайного графа;
replot – перерисовать график;
restart – очистка значений переменных;
read – чтение из файла;
readlib – чтение библиотечной функции;
rhs – выделение правой части выражения;
rsolve – решение рекурентных выражений;

S

save – запись выражений в файл;

seq – генерация последовательности;
SearchText – поиск текста в строке;
show – возвращает таблицу, содержащую всю информацию о графе;
shortpathtree – нахождение дерева кратчайших путей в графе;
signum – знак числа;
solve – решение уравнений и систем уравнений в символьном виде;
Sum – распечатка суммы ряда;
sum – нахождение суммы ряда;
subs – подстановка в выражение;

T

taylor – разложение в ряд Тейлора;
table – создание таблицы;
tail – возвращает имя "хвоста" графа;
type – проверка принадлежности типу;

U

union – объединение множеств;

V

vector – задание вектора;
vertices – просмотр узлов графа;
void – создает граф без ребер;
vweight – находит вес вершины;

W

whattype – определение типов;
with – подключение библиотеки;
writeto – запись в новый файл;
write – запись в файл;
writeln – запись строки в файл.

Алфавитный указатель

A

B

C

D

T

U

V

W

X

Z