# УЧЕБНОЕ ПОСОБИЕ

# Основы использования математического пакета MAPLE в моделировании

П.В. Сараев

УДК 51.37 С 20 П.В. Сараев ISBN 5-98858-008-4

ОСНОВЫ ИСПОЛЬЗОВАНИЯ МАТЕМАТИЧЕСКОГО ПАКЕТА МАРLE В МОДЕЛИРОВАНИИ: Учебное пособие / Международный институт компьютерных технологий. Липецк, 2006. 119с.

В учебном пособии рассмотрены принципы работы в математическом пакете MAPLE, достаточные для реализации моделей типовых математических схем и для решения задач, возникающих при моделировании различных систем и процессов. Описана среда пакета, основные структуры, математические возможности пакета, а также язык программирования. Пособие предназначено для студентов технических вузов, обучающихся по специальности 230101 «Вычислительные машины, комплексы, системы и сети» по дисциплине «Моделирование».

Табл. 9. Ил. 3. Библиогр.: 4 назв.

Рецензенты: кафедра прикладной математики Липецкого государственного технического университета (д.ф.-м.н., проф. Блюмин С.Л.); к.т.н., доц. Качановский Ю.П. (Международный институт компьютерных технологий).

Допущено учебно-методическим объединением вузов по университетскому политехническому образованию в качестве учебного пособия для студентов, обучающихся по специальности 230101 «Вычислительные машины, комплексы, системы и сети».

- ISBN 5-98858-008-4  $\odot$  П.В. Сараев, 2006.
  - © Международный институт компьютерных технологий. Липецк. 2006.

# Содержание

B	ведение	4
1.	Среда пакета Maple	6
	1.1. Интерфейс пользователя	. 6
	1.2. Контрольные вопросы к разделу 1	. 15
2.	Объекты и структуры данных пакета Maple	16
	2.1. Простейшие объекты: константы, числа, строки, переменные.	
	2.2. Выражения и работа с ними	
	2.3. Базовые структуры данных: последовательности выражений	
	списки, множества, таблицы, массивы	
	2.4. Контрольные вопросы к разделу 2	
3.	Программные конструкции	54
	3.1. Условный оператор	_
	3.2. Оператор цикла	
	3.3. Процедуры и функции	
	3.4. Контрольные вопросы к разделу 3	
4.	Векторно-матричные операции	70
	4.1. Подключаемые пакеты	. 70
	4.2. Пакет <i>linalg</i>	
	4.3. Пакеты LinearAlgebra и VectorCalculus	
	4.4. Контрольные вопросы к разделу 4	
<b>5</b> .	Случайные числа. Функции распределений	93
	5.1. Генерация псевдослучайных чисел	. 93
	5.2. Генерация случайных величин	
	5.3. Работа с функциями распределений. Квантили	
	5.4. Контрольные вопросы к разделу 5	. 102
6.	Файловые операции	103
	6.1. Общие сведения о работе с файлами	. 103
	6.2. Форматированной ввод и вывод	
	6.3. Команды ввода и вывода строк и таблиц данных	
	6.4. Контрольные вопросы к разделу 6	. 114
За	аключение	115
Пј	редметный указатель	116
Бі	иблиографический список	119

#### Введение

Марlе является мощным математическим пакетом, разработанным подразделением Maplesoft канадской компании Waterloo Maple Inc. В данном пособии изложение опирается на 8-ю версию пакета Maple, однако, большинство информации будет справедливо и для более ранних, и для новых версий. Существует русификация интерфейса 8-й версии Maple, поэтому при ссылке на элементы интерфейса (пунктов меню) будут указываться и соответствующие наименования в русифицированной версии (к сожалению, не всегда удачные). Отметим, что сообщения об ошибках и справка в Maple остались нерусифицированными. Мощные математические средства и программные конструкции пакета Марle могут быть использованы для решения огромного количества задач, возникающих при моделировании различных систем и процессов, протекающих в технических, социальных, экономических системах.

Данное пособие предназначено для самостоятельного изучения студентами, имеющими базовые знания и навыки в программировании. Данное пособие может быть использовано преподавателями на лабораторных занятиях для знакомства студентов с основными понятиями математического пакета Марle. Для усвоения базовых знаний необходимо изучить разделы 1-3, поэтому изучение именно этих разделов рекомендуется преподавателям для знакомства студентов с пакетом Maple. В связи с тем, что при моделировании активно используются вычисления с использованием матриц и векторов, особое внимание следует уделить разделу 4. В разделе 5 приведены возможности генерации последовательностей псевдослучайных чисел и работы с вероятностными распределениями. Раздел 6 рассчитан на тех, кто хочет повысить эффективность своей работы с данными в пакете Марle, работая с файлами.

Для правильного понимания материала, приведенного в пособии, следует обратить внимание на следующее:

- Клавиша, которую следует нажать на клавиатуре, будет обозначаться в угловых скобках <>, например, <Enter>. Комбинация нажимаемых клавиш обозначается с помощью символа + (плюс). Например, запись <Ctrl>+<j> означает, что надо нажать на клавишу <Ctrl> и, не отпуская ее, нажать на клавишу <j>.
- Последовательность вызова пунктов меню будет записываться с помощью стрелок. Например, запись  $File \rightarrow Open \ (\Phia\'un \rightarrow Om\kappa pыmb)$  означает, что следует выбрать из главного меню пункт  $File \ (\Phia\'un)$ , а затем в открывшемся подменю пункт  $Open \ (Om\kappa pыmb)$ .
- Для выделения команд Maple в тексте они будут обозначаться полужирным шрифтом, например, так: **restart**. Командами будут называться и процедуры, и функции ядра и подключаемых пакетов Maple.
- Элементы, вместо которых следует поставить некоторое значение, будут указываться в угловых скобках <> полужирным шрифтом. На-

пример, **<число>** означает, что вместо этого выражения должно быть подставлено некоторое конкретное число.

- В формате вызова процедуры или функции квадратные скобки [] обозначают эти же самые символы в отличие от многих книг, в которых эти скобки обозначают необязательные параметры. Необязательность передачи тех или иных параметров будет указана явно или видна из факта наличия значения параметра по умолчанию. В ряде программных конструкций необязательные элементы будут ограничены вертикальными чертами |.
- Примеры использования команд желательно сопровождать выполнением их в пакете Maple. Имейте также в виду: из-за того, что Maple является интерпретатором, результат исполнения многих команд будет зависеть от того, какие вычисления вы произвели ранее. Иногда будет явно указываться на необходимость сброса всех результатов (очистка переменных и т.д.). В начале каждого раздела необходимо давать команду restart.
- Упражнения представляют собой задания для самостоятельного выполнения. Выполнение всех упражнений позволит максимально быстро и глубоко изучить математический пакет Maple.

Знания, полученные после изучения и проработки всего материала, приведенного в пособии, могут быть успешно применены студентами при написании лабораторных и курсовых работ как по дисциплине «Моделирование», так и по другим дисциплинам математико-компьютерного цикла. Также эти знания могут пригодиться при работе над дипломными работами и проектами.

Тем студентам, у которых есть желание действительно глубоко разобраться с пакетом Maple, рекомендуется воспользоваться информацией с сайта

#### http://www.maplesoft.com

компании-разработчика пакета Maple. На этом сайте можно найти много учебных материалов, а также материалов по решению реальных практических задач.

# 1. Среда пакета Марlе

# 1.1. Интерфейс пользователя

На рис. 1.1 показано окно, возникающее обычно при запуске Maple. В окне интерфейса выделяется несколько основных областей:

- строка основного меню;
- панель инструментов;
- рабочая область (содержащая один или несколько рабочих листов);
- строка состояния.

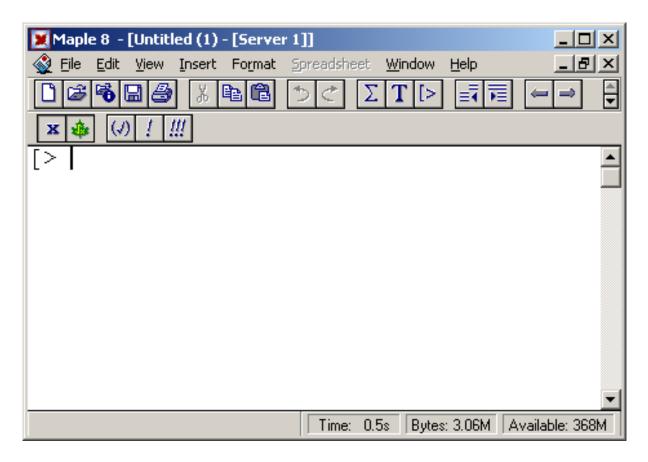


Рис. 1.1. Вид Maple после загрузки

Интерфейс Maple имеет следующие особенности:

- содержимое основного меню (включая доступность тех или иных пунктов) зависит от активного объекта и поэтому является контекстным;
- панель инструментов, дублирующая наиболее часто используемые пункты меню, состоит из 2-х частей основной (верхняя часть) и контекстной (нижняя часть).

Рабочий лист (worksheet) является основным документом, в котором вводятся команды пользователя, и в который выдаются результаты работы пакета Maple. Марle является интерактивной системой, интерпретатором — каждая вводимая команда после нажатия на клавишу <Enter> передается

на выполнение ядру Maple. Для запуска на обработку текущей команды вместо клавиши «Enter» можно левой клавишей мыши нажать кнопку с одним восклицательным знаком на панели инструментов. Далее при упоминании на необходимость нажатия на клавишу «Enter» будет подразумеваться и возможность запуска команды с помощью указанной кнопки на панели инструментов.

В рабочем листе выделяются области ввода и области вывода. В области ввода пользователем вносятся команды и операторы, а также комментарии и текстовая информация. В полях вывода отображаются результаты выполнения введенных команд, включая сообщения об ошибках. При этом графика, выводимая Марle, также обычно отображается в области вывода, но может отображаться и в отдельном окне, если в установках установлен режим вставки графики в отдельный лист. Комментарии и текстовая информация ядром Марle не обрабатываются и предназначены только для разработчиков и пользователей программ.

Область ввода и соответствующая ей область вывода называется *груп- пой вычислений*. На рабочем листе она отмечается квадратной скобкой слева. В группе вычислений может содержаться несколько областей ввода и вывода: все команды и операторы в областях ввода одной группы вычислений обрабатываются системой за одно обращение по нажатию клавиши <Enter>. Давайте введем после символа приглашения Maple > выражение

# 2+2\*3;

и далее нажмем на <Enter>. На экран Maple выведет ответ — число 8 (приоритет арифметических операций, естественно, поддерживается). В примере показано, как это будет выглядеть в окне Maple.

# Пример 1.1.

> 2+2\*3;

8

Обратите внимание на символ; (точка с запятой) в завершении команды. Если вы введете только

$$2+2*3$$

то Maple выдаст сообщение об ошибке «Warning, premature end of input» («Предупреждение, преждевременное окончание ввода»).

После обработки команды и помещения результата в область вывода группы вычислений Maple сформирует начало следующей группы и предложит ввести команду в том же самом режиме, который и был до обработки

предыдущей команды. Стандартные настройки пакета Maple таковы, что команды вводятся шрифтом красного цвета, вывод осуществляется шрифтом синего цвета.

Добавить новую группу после курсора можно по нажатию комбинации клавиш <Ctrl>+<J> либо через пункт меню «Insert» $\rightarrow$ «Group» $\rightarrow$ «After cursor» («Вставка» $\rightarrow$ «Группа» $\rightarrow$ «После курсора»). Перед курсором вставка новой группы вычислений осуществляется по нажатию комбинации клавиш <Ctrl>+<K>, а также через пункт меню «Insert» $\rightarrow$ «Group» $\rightarrow$ «Before cursor» («Вставка» $\rightarrow$ «Группа» $\rightarrow$ «Перед курсором»).

В области ввода вводимая информация может быть двух типов:

- 1. Команды и операторы Maple, которые обрабатываются пакетом: в строке рабочего листа выдается приглашение на ввод команд символ > (больше). Команды могут вводиться либо в форме синтаксиса языка Maple в режиме «Maple Input» («Марlе ввод»), либо в форме стандартной математической записи в режиме «Standard Math Input» («Стандартный ввод»). Во втором случае в строке рабочего листа после символ > выдается символ ? (вопросительный знак).
- 2. Текстовая информация, которая не обрабатывается Maple. Здесь может вводиться просто текст в режиме «Text» («Teкст»), либо формулы в математической нотации в режиме «Standard Math» («Стандартная»), когда в строке рабочего листа выдается символ ? (вопросительный знак).

Таким образом, существует всего четыре режима ввода информации в Maple:

- 1. Ввод текстовой информации.
- 2. Ввод команд Maple в стандартном режиме.
- 3. Ввод текстовой математической символики.
- 4. Ввод команд Maple в виде математической символики.

Для смены режимов можно выбрать пункт основного меню «Insert» («Вставка»), после чего на экран будет выведено подменю, первые четыре пункта которого и соответствуют указанным режимам ввода. Выберем первый пункт подменю «Text» («Текст»). На экране будет отображена только квадратная скобка [. Введем строку «Знакомство с пакетом Maple» и нажмем на «Enter». В результате Maple на экран ничего не выведет, а лишь переместит курсор на следующую строку.

# Пример 1.2.

# Знакомство с пакетом Maple

В данном режиме Maple функционирует как текстовый редактор. Можно изменять гарнитуру, размер шрифта, параметры выравнивания (по левому краю, по центру или по правому краю), параметры набора (жирность шрифта, наклон, подчеркивание).

Теперь выберем в меню пункт «Insert» («Вставка») и далее второй пункт выпавшего подменю «Standard Math» («Стандартная») . На экране появится знак вопроса ? и дополнительное поле для ввода текста на панели инструментов (оно похоже на поле, возникающее при редактировании ячейки в табличном редакторе Microsoft Excel). В появившемся поле необходимо ввести выражение Maple (например, int(2\*x,x)), после чего нажать клавишу «Enter». После этих операций в рабочем листе на месте знака вопроса появится соответствующее математическое выражение.

#### Пример 1.3.

 $\int 2 x \, dx$ 

В данном режиме решается проблема вставки математической символики (интегралов, пределов, сумм и т.д.) в документы. Как видно, пакет Maple является удобным инструментом не только вычислений, но и создания хорошо оформленных документов, содержащих математическую нотацию.

Создадим теперь новую группу вычислений, нажав комбинацию клавиш <Ctrl>+<J>. На экране появится приглашение Maple (символ >) к вводу команды. Попытаемся перейти в третий режим ввода, для чего выберем в меню пункт «Insert» («Вставка») и далее пункт «Maple Input» («Марle ввод») в третьей строке выпавшего подменю. На экране ничего не изменится — это означает, что Maple в настоящее время в этом режиме и находится. Данный режим является основным для Maple. Наберем в строке выражение

$$int(2*x,x);$$

и нажмем на <Enter>. В отличие от предыдущего результата Maple в области вывода выведет ответ на команду вычисления неопределенного интеграла от выражения.

#### Пример 1.4.

 $x^2$ 

Переход в четвертый режим осуществляется путем выбора в меню пункта «Insert» («Вставка») и далее пункта «Standard Input» («Стандартный ввод») в четвертой строке. На экране одновременно отобразятся 2 символа: >?. Как и раньше, при появлении знака вопроса в области панели инструментов появится дополнительное поле для ввода строки, в котором наберем

$$int(2*x,x)$$

(можно без символа «;») и нажмем на <Enter>. В рабочем листе в области ввода появится изображение команды пакета Maple, но в математической нотации. Еще раз нажав на <Enter>, получим результат выполнения команды.

#### Пример 1.5.

 $> \int 2 x dx$ 

 $x^2$ 

Обратите внимание, что первое нажатие на <Enter> привело к записи команды Maple в математической нотации, а второе — к выполнению команды. Этот режим позволяет людям, не знакомым с пакетом Maple, но работающим с математикой, понимать смысл программ на языке пакета Maple.

Все дальнейшее рассмотрение будет опираться на работу в основном режиме — режиме «Maple Input» («Марlе ввод»). Если сейчас Марlе находится в другом режиме, перейдите в основной режим ввода команд.

Несколько групп вычислений, включая текстовые комментарии, могут быть объединены в ceкцию. Секция представляется в виде серого квадратика со знаком + (плюс) или - (минус) и вертикальной скобки, объединяющей группы секции. Секция может быть раскрытой - в этом случае на листе отображены все группы и команды в группах, объединенных секцией, а также квадратик показан со знаком -. Если мышкой щелкнуть на знаке -, то секция станет свернутой - на экране будет находиться лишь знак +, а все содержимое будет скрыто. Для вставки секции необходимо выбрать пункт меню  $Insert \rightarrow Section$  ( $Bcmaeka \rightarrow Pasden$ ) или выделить имеющиеся группы и выбрать в меню пункт  $Format \rightarrow Indent$  ( $\Phiopmam \rightarrow Ocmyn$ ) (горячая комбинация клавиш - «Ctrl > + <.»). Отменить объединение групп в секцию можно с помощью пункта меню  $Format \rightarrow Outdent$  ( $\Phiopmam \rightarrow Bmssecka$ ) (горячая комбинация клавиш - «Ctrl > + <.»).

Для изучения команд пакета Maple следует знать следующие правила набора команд:

1. Марle чувствителен к регистру вводимых символов, т.е. большие и маленькие буквы система воспринимает по-разному. Если команда написана в виде

$$int(2*x,x);$$

ее не следует набирать как

$$INT(2*x,x);$$

В последнем случае в области вывода исходная команда будет переписана, а это свидетельствует о том, что такая команда в настоящий момент времени Maple неизвестна.

#### Пример 1.6.

```
> INT(2*x,x); INT(2x,x)
```

Из-за невнимательности иногда можно получить непонятные результаты и труднонаходимые ошибки.

# Пример 1.7.

```
> int(2*x,X); 2xX
```

- 2. При вводе длинной команды, не помещающейся в одной строке, Maple автоматически переходит на следующую строку, считая при этом части команды единым целом.
- 3. В одной строке можно вводить несколько команд.
- 4. Признаком завершения каждой команды является символ: (двоеточие) или; (точка с запятой). Если команда заканчивается символом; (точка с запятой), то команда будет обработана, а результаты исполнения выданы в области вывода. Если же команда завершается символом: (двоеточие), то команда будет выполнена (!), но на экран результаты выполнения команды выданы не будут. Нельзя путать отсутствие результата в области вывода при работе в режиме ввода текста и в режиме ввода команд Maple.

#### Пример 1.8.

```
> x := 3:
> X := x+1:
> x^2;
```

9

В данном примере используется оператор присваивания :=, который результат вычисления в правой части присваивает переменной в левой части. Из данного примера видно, что опечатка во втором операторе привела к тому, что переменная x не получила увеличения на единицу, а вместо этого была введена новая переменная X. В результате пользователь вместо ожидаемого ответа 16 получил 9.

5. Если необходимо, чтобы команды располагались по одной на строке, а Maple обрабатывал их в рамках единой операции, необходимо после ввода команды вместо <Enter> нажать клавиши <Shift>+<Enter>. В этом случае введенная команда не обрабатывается, а курсор устанавливается на следующую строку.

# Пример 1.9. > 2+4\*3; 7-3^2;

14

-2

- 6. Для запуска на исполнение клавиша <Enter> может нажиматься не только в конце области ввода (т.е. когда курсор находится после всех введенных команд), но и в любом месте этой области ввода.
- 7. На результат выполнения предыдущей операции можно ссылаться с помощью символа % (процент). На результат выполнения «предпредыдущей» и «предпредыдущей» команды можно ссылаться с помощью символов %% и %%% соответственно. Это так называемые «операторы Дитто» («Ditto operators»), означающие, согласно словарю Вебстера, «вещи, подразумеваемые ранее или выше».

### Упражнение 1.1. Найдите значение выражения

$$2 - \frac{3+1}{4-2} \cdot 3$$

и затем к полученному результату (воспользуйтесь символом %) прибавьте 7.

8. Комментарий в Maple начинается с символа # (решетка), он продолжается до конца команды. Текст, приведенный в комментарии, не обрабатывается.

В связи с тем, что Марlе является интерпретатором, следует четко понимать, что после ввода и выполнения некоторой команды (группы) можно в любой момент времени вернуться на соответствующую строку, после чего по нажатию на «Enter» команда будет выполнена вновь. При этом в ряде случаев результат будет отличен от предыдущего, т.к. переменные и прочие конструкции (функции и т.д.) могут иметь уже другое значение.

Чтобы выполнить все команды заново в том порядке, в котором они введены в систему, вместо того, чтобы на каждой команде заново нажимать  $\langle \text{Enter} \rangle$ , можно воспользоваться пунктом меню  $Edit \rightarrow Execute \rightarrow Worksheet$ 

 $( \Pi pae \kappa a \rightarrow 3 any c \kappa \rightarrow Tab \pi u y )$ . Можно также выделить требуемый для выполнения фрагмент и воспользоваться пунктом меню  $Edit \rightarrow Execute \rightarrow Selection$   $( \Pi pae \kappa a \rightarrow 3 any c \kappa \rightarrow B \omega b pah u e e)$ .

Для более удобного ввода наиболее часто используемых команд в Maple имеются средства, называемые *палитрами* (palettes). Пожалуй, лучше их было бы назвать шаблонами, т.к. эти средства формируют «скелет» команды, предоставляя пользователю возможность ввести необходимое количество параметров. В 8-й версии Maple доступны 4 вида палитр:

- 1. Для ввода символов (Symbol Palette).
- 2. Для ввода выражений (интегралов и т.д.) (Expression Palette).
- 3. Для ввода матриц размером не более  $4 \times 4$  (Matrix Palette).
- 4. Для ввода векторов (строк или столбцов), состоящих не более чем из 5 элементов (Vector Palette).

Для их вывода на экран нужно в пункте «View» («Вид») основного меню выбрать пункт «Palettes» («Палитры») и далее требуемую палитру или пункт «All palettes» («Показать все») для вывода всех четырех палитр.

Например, для вычисления следующего предела

$$\lim_{n \to \infty} \frac{2n^2 + 3n - 4}{7n^2 - 8n + 1}$$

можно вызвать палитру выражений (рис. 1.2), после чего выбрать верхнюю правую кнопку, нажать на *«Enter»* и заполнить предложенный шаблон.

_	EXPRESSION						
	J.	$\int_{c}^{p}$	d E o bec	g Thou	<u>а о</u>	lima b→c	
	a+b	α−b	a×b	a./b	a=b	a≔b	
	αb	α <sub>b</sub>	$\sqrt{\alpha}$	₩	a.i	a	
	ea	ln	log	sin	cos	tan	

Рис. 1.2. Палитра выражений

В любой момент пользователю доступна удобно организованная справочная система по среде и командам пакета Maple. Вызвать ее можно либо по нажатию клавиши <F1>, либо по выбору пункта меню Help и далее соответствующего пункта, например, Mathematics. Справочная система (рис. 1.3) организована в виде гипертекстового документа; ссылки выделены бирюзовым цветом и подчеркнуты.

Обычно внизу справки по каждой команде приводятся примеры по работе с данной командой. Эти примеры можно скопировать в буфер с помощью

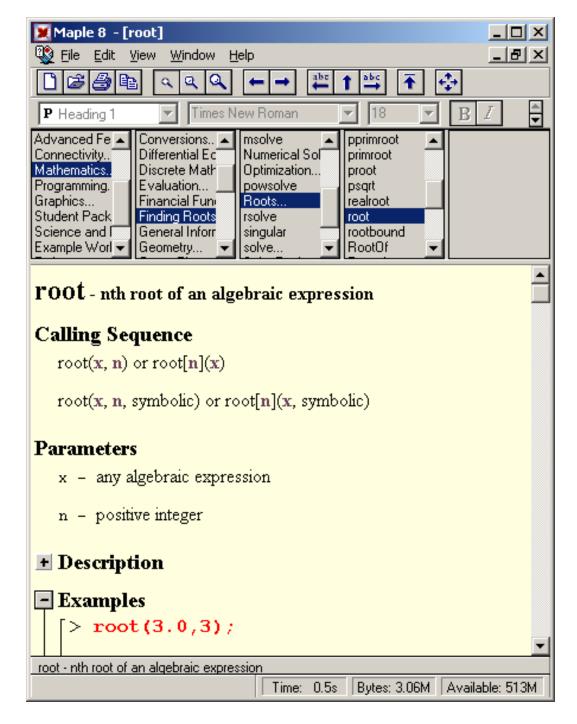


Рис. 1.3. Вид справочной системы пакета Maple

пункта меню  $Edit \rightarrow Copy\ Examples$  (меню в справочной системе не русифицировано) и затем вставить в рабочий лист. Наличие большого количества примеров позволяет эффективно использовать Марle даже тем, которые недостаточно хорошо знают английский язык. Хотя, конечно, знание языка позволит более полно изучить математический пакет. Можно получить быструю справку по команде, на которой установлен курсор, с помощью комбинации клавиш <Ctrl>+<F1>, а также путем ввода команды вызова помощи ?.

#### ⊳ Команда ?.

Назначение: вызов контекстной справочной информации.

Формат вызова:

# ?<тема>[<подтема>]

Параметры:

- -<тема>- тема, по которой необходимо вывести справку;
- -<подтема>- раздел указанной темы (необязательный параметр).

После вызова справки с помощью команды ? следует нажать <Enter>; символы «;» или «:» в качестве завершения команды в этом случае ставить нет необходимости. К примеру, команда

#### ?type

выведет справку о функции проверки типа указанного выражения. Команда же

# ?type[set]

вызовет справку по использованию функции **type** с параметром **set** (проверка того, является ли переданный объект множеством).

В качестве параметра **<тема>** может быть использовано также одно из ключевых слов:

- **intro** для вызова информации о знакомстве с пакетом Maple;
- worksheet для вызова информации об интерфейсе рабочих листов;
- **index** для вывода всех категорий справочной информации.

Упражнение 1.2. С помощью палитры выражений рассчитайте

$$\sqrt[3]{8}$$
.

Смысл параметров узнайте с помощью справочной системы.

# 1.2. Контрольные вопросы к разделу 1

- 1. Среда Maple. Особенности интерфейса.
- 2. Принципы ввода и вывода информации в Maple.
- 3. Ввод команд и запуск их на выполнение. Как можно обратиться к результатам предыдущих вычислений? Как выполнить команду без вывода результатов на экран?
- 4. Секции: назначение и работа с ними.
- 5. Упрощенный механизм ввода символов, математических структур и функций. Как вы считаете, насколько полезен этот механизм?
- 6. Организация справочной системы.

# 2. Объекты и структуры данных пакета Maple

# 2.1. Простейшие объекты: константы, числа, строки, переменные

Простейшими объектами языка Maple, как, впрочем, и любого другого языка программирования, являются:

- предопределенные именованные константы (обратите внимание на регистр символов в названиях констант!);
- числа;
- строки;
- логические высказывания;
- переменные.

К наиболее важным константам в Maple относятся следующие:

- I мнимая единица  $(\sqrt{-1})$ ;
- $\mathbf{Pi}$  число  $\pi \approx 3.14159\ldots$ ;
- **infinity** бесконечность  $(\infty)$ ;
- false ложь в логических выражениях;
- **true** истина в логических выражениях.

Еще раз обратите внимание на важность правильной записи регистров символов. Например, записи  $\mathbf{Pi}$  и  $\mathbf{pi}$  выведут на экран одинаковое изображение  $\pi$ , хотя суть у них будет разной. В первом случае это будет константа, а во втором — неинициализированная переменная. Проверить это можно следующим образом (о команде  $\mathbf{evalf}$  будет сказано ниже).

# Пример 2.1.

> Pi; evalf(%); # Предопределенная константа

7

3.141592654

> pi; evalf(%); # Просто переменная

 $\pi$ 

 $\pi$ 

Напомним, что символ % означает подстановку результата выполнения предыдущей команды. Этот символ позволяет упростить запись некоторых команд и отказаться от введения ненужных переменных.

Числа в пакете Maple могут быть следующими:

- целыми;
- обыкновенными дробями;
- радикалами;

- числами с плавающей точкой (запятой);
- комплексными (мнимая единица представляется указанным выше символом  ${f I}$ ).

Первые три вида чисел являются точными типами, т.е. позволяют выполнять точные вычисления (если в выражении встречаются только точные типы чисел, результат не будет округляться по умолчанию), тогда как последние два — приближенные. Преобразование чисел в формат с плавающей можно с помощью команды evalf (от англ. evaluate as float — вычислить как число с плавающей точкой).

#### ⊳ Команда **evalf**.

Назначение команды: перевод точного числового типа в приближенный. Формат вызова 1:

Формат вызова 2:

Параметры:

- **<выражение>** выражение, в котором все числа требуется привести к виду чисел с плавающей точкой;
- -<кол\_цифр>- целое число, указывающее количество значащих цифр для вычислений (необязательный параметр).

◁

Заметим, что параметр <кол\_цифр> не следует понимать как количество цифр после десятичной точки; он означает количество цифр, начиная с первой ненулевой цифры слева (подсчитайте количество цифр в числе  $\pi$ , приведенное в примере выше). По умолчанию значение этого параметра равно значению переменной пакета Maple **Digits**, которое инициализируется значением 10, но которое пользователь может изменить. Приведем пример использования команды **evalf** и влияния на ее работу переменной **Digits**.

# Пример 2.2.

```
Digits := 5: evalf(3/4*x^2+1/3*x-sqrt(2)); 0.75000 \, x^2 + 0.33333 \, x - 1.4142
Digits := 10: evalf(3/4*x^2+1/3*x-sqrt(2)); 0.7500000000 \, x^2 + 0.333333333 \, x - 1.414213562
```

Обратите внимание, что в данном примере после присваивания переменной **Digits** нового значения стоит символ : (двоеточие), который требуемое действие выполняет, но результат выполнения команды на экран не выводит.

Упражнение 2.1. Найдите сумму двух обыкновенных дробей:

$$\frac{1}{2} + \frac{1}{3}$$
.

Объясните, почему результат не является числом с плавающей точкой. Преобразуйте результат к виду числа с плавающей точкой.

Сделаем небольшое отступление, обратив внимание на возможность задания точности вывода вещественных чисел на экран с помощью команды настройки интерфейса **interface**. Она является в некотором смысле дополнением к переменной **Digits**. Команда установки точности отображения результатов (еще раз подчеркнем, не точности вычислений!) имеет следующий вид:

# interface(displayprecision=<точность>)

В этой команде <**точность**> указывает на количество цифр после десятичной точки, отображаемых в вещественных числах, и может принимать значения -1, 0 или быть положительным целым числом. Значение -1 является значением по умолчанию и указывает на то, что выводить числа следует с полной вычисляемой точностью, которую задает переменная **Digits**.

#### Пример 2.3.

> evalf(2/3);

0.666666667

> interface(displayprecision=3): evalf(2/3); # Отображаем 3 знака после запятой

0.667

> interface(displayprecision=0): evalf(2/3); # Отображаем 0 знаков после запятой (округляем до целых)

1

> interface(displayprecision=-1): evalf(2/3); # Восстанавливаем точность, определяемую переменной Digits 0.666666667

Строковые константы ограничиваются двойными кавычками ("").

# Пример 2.4.

> "Hello, world!";

"Hello, world!"

При необходимости задания в строке непосредственно двойной кавычки ее следует указать в строке, предварив символом \ (обратный слэш).

#### Пример 2.5.

Конкатенация двух строк (слияние) производится с помощью оператора | (две вертикальные черты).

#### Пример 2.6.

Строка в системе Maple определяется как одномерный массив символов (индексация с единицы), поэтому для выделения подстроки можно воспользоваться конструкцией извлечения элементов массива [].

#### Пример 2.7.

```
> "Hello, world!"[1..5]; "Hello"
```

Высказыванием называется предложение, относительно которого можно сказать, истинно оно или ложно. Таким образом, логические высказывания при использовании классической двузначной логики могут принимать только одно из двух значений — истину, которая обозначается константой **true**, и ложь, которая обозначается константой **false**. Высказывания используются, в частности, в операторах условий.

Для того, чтобы сравнить два объекта, например, два числа, недостаточно записать в командной строке выражение 2=2 или 2<>2 — Марlе не выведет на экран константу **true** или **false**, которая сказала бы об истинности или ложности указанного высказывания. Связано это с тем, что Maple будет считать, что указан один из объектов — равенство или неравенство (об этих объектах будет рассказано далее).

# Пример 2.8.

> 2<>2;

 $2 \neq 2$ 

В этом примере Maple решил, что в команде указан (введен) объект «неравенство». Для того, чтобы узнать, истинно или ложно высказывание, следует воспользоваться функцией **evalb** (от англ. **eval**uate as **b**oolean — вычислить как булевское выражение).

⊳ Команда **evalb**.

Назначение команды: проверка истинности или ложности логического высказывания.

Формат вызова:

### evalb(<выражение>)

Параметр:

— **<выражение>** — выражение, об истинности или ложности которого нужно сказать.

В параметре <выражение> можно использовать операторы отношений = (равно), <> (не равно), < (меньше), <= (меньше или равно), > (больше), >= (больше или равно).

# Пример 2.9.

> evalb(2<>2);

false

Упражнение 2.2. Выясните, какое из двух чисел больше:

0.76

unu

 $\frac{7}{9}$ .

Переменные представляют собой объекты, которые могут изменяться в программе. Наименования переменных подчиняются стандартным правилам языков программирования: они представляют собой последовательности символов больших и маленьких латинских букв, цифр и символа подчеркивания. Начинаться наименование переменной с цифры не может. Для того, чтобы переопределить значение переменной, ей необходимо присвоить некоторое значение с помощью оператора присваивания:

—.

⊳ Команда :=.

Назначение команды: присваивание переменной значения.

Формат вызова:

Параметры:

- <**левая часть**> наименование переменной;
- -<правая  $\_$ часть>- объект, который присваивается переменной.

Пример использования оператора присваивания был приведен при изучении команды **evalf**, когда присваивалось новое значение переменной **Digits**. На самом деле, параметр **<левая\_часть>** может быть также выражением, функцией или последовательностью имен, однако, в данном пособии такой способ использования оператора присваивания не описывается.

Komanдa **assigned** проверяет, было ли переменной присвоено некоторое значение или нет.

#### ⊳ Команда **assigned**.

Назначение команды: проверка переменной на наличие присвоенного ей значения.

Формат вызова:

# assigned(<переменная>)

Параметр:

— **<переменная**> — имя переменной.

o nomo

◁

Эта команда возвращает значение **true**, если переменной было ранее присвоено некоторое значение, и **false** в противном случае.

#### Пример 2.10.

> assigned(newvar); # Переменная "newvar" пока неинициализирована

false

- > newvar := 4;
- newvar := 4
- > assigned(newvar); # Теперь в переменную "newvar" записано значение 4

true

В приведенном примере видно, что вначале переменной *newvar* не было присвоено никакого значения, о чем и сообщила функция **assigned**, выдав значение **false**. После же присваивания переменной *newvar* значения 4 команда **assigned** выдала значение **true**, говорящее о том, что переменной значение присвоено.

Для того, чтобы очистить значение переменной, необходимо присвоить ей свое собственное имя, записанное в одинарных кавычках.

# Пример 2.11.

```
> a := 2; assigned(a); a := 2 \\ true
```

$$>$$
 a := 'a'; assigned(a); 
$$a := a \\ false$$

Существует второй вариант очистки содержимого переменной — использование команды **unassign**.

# ⊳ Команда **unassign**.

Назначение команды: очистка содержимого переменной.

Формат вызова:

unassign(
$$<$$
перем $1>$ , $<$ перем $2>$ ,...)

#### Параметры:

-<перем1>,<перем $2>,\ldots-$  имена переменных, заключенные в одинарные кавычки.

◁

#### Пример 2.12.

```
> a := 2; assigned(a); a := 2 true > unassign('a'); assigned(a); false
```

Команда **anames** (от англ. **a**ssigned **names** — инициализированные переменные) позволяет вывести список всех инициализированных переменных, т.е. те переменные, которым присвоено некоторое значение.

#### ⊳ Команда anames.

Назначение команды: список инициализированных переменных. Формат вызова:

# anames(<\piapam>)

#### Параметр:

— < парам > — уточняющий параметр, который говорит о необходимости вывода определенного списка переменных (необязательный параметр).

 $\triangleleft$ 

Если параметр не передается, то выводится список всех инициализированных переменных. Если в качестве **<парам>** указывается тип переменной, то будет выведен список всех переменных указанного типа (включая переменные среды пакета Maple). Если же в качестве **<парам>** указывается значение **'environment'**, то выведется список переменных окружения (**Digits** и т.д.).

#### Пример 2.13.

- > anames(integer);  $sqrt/primes, \ Digits, \ printlevel, \ Order, \ newvar$
- > anames(environment);

Testzero, UseHardwareFloats, Rounding, %, %%%, Digits, index/newtable, mod, %%, Order, printlevel, Normalizer, NumericEventHandlers

В первой строке выводится список всех целочисленных переменных, к которым относятся, в частности, определенная ранее переменная newvar и переменная пакета **Digits**. Значение переменной a не выведено, т.к. в предыдущем примере ее содержимое было очищено. Во второй строке выводится перечень всех переменных окружения. Заметьте, что переменной newvar там больше нет.

Часто при проведении вычислений, упрощении выражений, решении уравнений и неравенств требуется использовать предположения относительно переменных или выражений. К ним относятся, например, предположения об области определения переменных, задаваемые с помощью команды assume.

#### ⊳ Команда assume.

Назначение команды: указание предположений относительно переменных. Формат вызова:

```
assume(<перем1>, <свойство1>, <перем2>, <свойство2>, ...)
```

Параметры:

- -<перем1>, <перем2>, ... переменные или выражения;
- -<свойство1>, <свойство2>, ... предположения.

Параметры **<свойство1>**, **<свойство2>** могут принимать, в частности, значения, приведенные в табл. 2.1.

 $\triangleleft$ 

# Пример 2.14.

> assume(x,nonnegative);

Таблица 2.1. Некоторые предположения команды **assume** 

Значение	Описание
negative	Отрицательное вещественное число, нуль не включается
nonnegative	Неотрицательное вещественное число, нуль включается
positive	Неотрицательное вещественное число, исключая нуль
natural	Неотрицательное целое число (нуль включается)
posint	Натуральное число (нуль не включается)
odd	Нечетное число
even	Четное число

В данном примере пакету Maple говорится, что относительно переменной x имеется предположение, что она может принимать только неотрицательные значения.

Также в качестве параметров **свойство1**, **свойство2**, ... можно указывать тип или диапазон чисел.

#### Пример 2.15.

> assume(y>0);

Следует также понимать, что предположения — это не ограничения, т.е. при предположении о положительности переменной ей все же удастся присвоить отрицательное значение.

#### Пример 2.16.

$$>$$
 y := -1;  $y:=-1$ 

Существуют предположения, предназначенные для функций (четная, дифференцируемая и т.д.) и матриц (невырожденная, симметричная и т.д.). Если на переменную наложены ограничения, в результатах она обычно будет показываться вместе с символом ~ (тильда).

# Пример 2.17.

$$>$$
 x+2;  $x^{\sim}+2$ 

Вывод символа при использовании переменных, на которые наложены предположения, можно отменить или сказать пакету Maple, чтобы он выводил переменные вместе с текстовой информацией о предположениях. Для этого необходимо выбрать пункт меню  $File \rightarrow Preferences$  ( $\Phiaŭn \rightarrow \Pi pednoumenus$ ) и далее вкладку I/O Display (нерусифицирована). После этого необходимо

выбрать нужную установку в группе Assumed Variables (также нерусифицирована).

Для снятия всех предположений относительно переменной необходимо присвоить ей ее же символьное имя (так же, как и для очистки содержимого переменной):

$$x := 'x';$$

Следует иметь в виду, что это не снимает предположений относительно всех выражений, в которых такая переменная уже была использована.

Также в команде **assume** можно указать несколько предположений.

# Пример 2.18.

> assume(x>0, x<1);

Новый вызов команды **assume** отменяет предыдущие предположения относительно этой же самой переменной. Чтобы было добавлено новое предположение, а старое не утратило силу, необходимо воспользоваться командой **additionally**.

#### ⊳ Команда additionally.

Назначение команды: введение дополнительных предположений относительно переменных.

Формат вызова:

additionally(
$$<$$
перем $1>$ ,  $<$ свойство $1>$ ,  $<$ перем $2>$ ,  $<$ свойство $2>$ , ...)

Параметры:

- -<перем1>, <перем2>, ... переменные или выражения;
- -<свойство1>, <свойство2>, ... дополнительные предположения.

Пример 2.19.

> assume(x>0); additionally(x<1);</pre>

Об имеющихся предположениях на переменную сообщает команда Maple **about**.

◁

#### ⊳ Команда **about**.

Назначение команды: вывод информации о предположениях относительно переменной.

Формат вызова:

```
about(<перем>)Параметр:- <перем> — переменная.
```

 $\triangleleft$ 

#### Пример 2.20.

```
> about(x);
Originally x, renamed x~:
  is assumed to be: RealRange(Open(0),Open(1))
```

В сообщении, выданном Maple, сказано, что предположение относительно переменной x заключается в том, что она принимает значение из интервала (0;1).

# 2.2. Выражения и работа с ними

Из констант, чисел, переменных, арифметических операций и математических функций могут составляться их комбинации— выражения.

# Пример 2.21.

```
> 3*a^2-2*a*sin(a)+1; 3a^2-2a\sin(a)+1
```

Арифметические операции над числами задаются с помощью обычных символов: + для сложения, - для вычитания, \* для умножения, / для деления. Возведение в степень производит операция (крышка) или \*\* (две звездочки). Это позволяет использовать Maple как «большой» калькулятор. Пакет Марle поддерживает большой набор встроенных математических функций, основные из которых приведены в табл. 2.2.

Не забывайте, что для вычисления экспоненты  $e^x$  надо писать не

 $\hat{\mathbf{e}}\mathbf{x}$ 

a

 $\exp(x)$ 

Таблица 2.2. Основные математические функции

Функция в Maple	Описание				
Функции округления и отсечения					
ceil	Округление до целого в большую сторону				
floor	Округление до целого в меньшую сторону				
round	Округление по правилам математического округления				
frac	Дробная часть числа				
trunc	Отсечение дробной части				
Экспонента и логарифмы					
exp	Экспонента				
ln	Натуральный логарифм				
$\log$	Логарифм				
$\log 10$	Логарифм по основанию 10				
Тригонометрические и гиперболические функции					
cos	Косинус				
$\sin$	Синус				
tan	Тангенс				
cosh	Гиперболический косинус				
sinh	Гиперболический синус				
tanh	Гиперболический тангенс				
Обратные тригонометрические и гиперболические функции					
arccos	Арккосинус				
arcsin	Арксинус				
arctan	Арктангенс				
arccosh	Гиперболический арккосинус				
arcsinh	Гиперболический арксинус				
arctanh	Гиперболический арктангенс				
	Прочие функции				
!	Факториал				
abs	Модуль				
max	Максимальное из набора действительных чисел				
min	Минимальное из набора действительных чисел				
signum	Знак числа (для нуля значение равно 0)				
sqrt	Квадратный корень				
surd	Корень заданной степени (радикал)				

Разницу в результатах выполнения этих команд можно заметить даже визуально: в области вывода Maple в первом случае e выведется курсивом (Maple считает, что это переменная), а во втором случае — жирным шрифтом  $\mathbf{e}$  (так Maple сообщает, что требовалось подставить экспоненту).

При работе с выражениями очень часто приходится вычислять их значения при заданных величинах входящих в них переменных. Вычисление значения выражения можно осуществить несколькими способами. Самый простой состоит в том, чтобы переменной присвоить некоторое значение, а затем просто записать выражение, в которую эта переменная входит. В этом случае в выражении вместо переменных будет подставлено их содержимое — соответствующие значения.

#### Пример 2.22.

> 
$$a := 5$$
;  $a := 5$ ;  $b := 5$ ;  $b$ 

Правда, как видно из примера, при таком способе вычисления теряется смысл выражения как выражения с переменной, т.е. как некоторой самостоятельной конструкции. Поэтому вместо рассмотренного выше способа вычислений выражений лучше пользоваться командой **eval**.

#### ⊳ Команда **eval**.

Назначение команды: вычисление выражения при заданных значениях переменных.

Формат вызова 1:

$$eval(<выражение>, <перем> = <значение>)$$

Формат вызова 2:

#### Параметры:

- **<выражение>** выражение, зависящее от одной или нескольких переменных;
- -<перем>-имя переменной;
- **<значение**> значение переменной (может быть также выражением);
- <равенства> список или множество равенств.

Рассмотрим сначала первый формат использования команды **eval**, который применяется для вычисления выражений при заданном значении одной переменной.

 $\triangleleft$ 

#### Пример 2.23.

> a := 'a'; 
$$a := a$$
> y := a^2; 
$$y := a^2$$
> eval(y, a = 5); 
$$25$$
> y; 
$$a^2$$

Обратите внимание, что в этом примере для предотвращения влияния результатов предыдущих вычислений сначала мы очищаем содержимое переменной a, присваивая ей свое собственное имя. В последней команде выводится содержимое переменной y, которое представляет собой заданное ранее выражение.

Рассмотренный выше формат использования команды **eval** позволяет легко вычислять значение выражения в том случае, если в его состав входит только одна переменная. Для того, чтобы можно было вычислить выражения при нескольких переменных, необходимо использовать второй формат вызова функции **eval**.

Для передачи параметра **<pae>pabeнства>** необходимо перечислить набор равенств вида **<переменная>** = **<правая\_часть>** через запятую, заключив их в фигурные или квадратные скобки. Понятия списков и множеств будут подробно рассмотрены далее.

# Пример 2.24.

Упражнение 2.3. Присвойте некоторой переменной выражение

$$e^x + \frac{\cos x}{y} - 2|x - y|.$$

Рассчитайте значение выражения при  $\{x=0,y=3\}$  и  $\{x=1.2,y=-0.3\}$ . Определите, при возможности, точные ответы (в виде обыкновенных дробей), а также в виде десятичных чисел с плавающей точкой.

На команду **eval** в ряде случаев по функциональности похожа команда **subs** (от англ. **subs**titute - 3amenumb).

⊳ Команда **subs**.

Назначение команды: синтаксическая замена подвыражений в выражениях. Формат вызова:

Параметры:

- **<выражение>** выражение, в котором необходимо произвести замену;
- **< старое** подвыражение> выражение, которое заменяется;
- **<новое\_подвыражение>** выражение, на которое заменяются вхождения выражения **<старое подвыражение>**.

 $\triangleleft$ 

Пример 2.25.

> subs(a=5, a^2-2\*a+1);

16

В приведенном примере команда **subs** работает аналогично команде **eval**.

#### Пример 2.26.

> subs(a^2+b^3=sqrt(c), 
$$\sin(a^2+b^3)*(a^2+b^3)$$
); 
$$\sin(\sqrt{c})\sqrt{c}$$

В этом примере все вхождения выражений  $a^2 + b^3$  заменяются на выражение  $\sqrt{c}$ . Однако, если мы немного изменим исходное выражение, добавив в одном из множителей единицу, результат несколько удивит.

# Пример 2.27.

> subs(a^2+b^3=sqrt(c), sin(a^2+b^3)\*(a^2+b^3+1)); 
$$\sin(\sqrt{c})\,(a^2+b^3+1)$$

Если в предыдущем случае все вхождения выражения были заменены, то сейчас во втором множителе замена не была сделана. Связано это с тем, что команда **subs** обеспечивает так называемую *синтаксическую* замену.

Более мощную замену выражений осуществляет команда *алгебраической* замены **algsubs**.

# ⊳ Команда **algsubs**.

Назначение команды: алгебраическая замена подвыражений в выражениях. Формат вызова:

algsubs(<старое\_подвыражение>=<новое\_подвыражение>, <выражение>)

Параметры:

- **<выражение**> выражение, в котором необходимо произвести замену;
- -<старое подвыражение>- выражение, которое заменяется;
- <новое\_подвыражение> выражение, на которое заменяются вхождения выражения <старое подвыражение>.

 $\triangleleft$ 

 $\triangleleft$ 

Эта команда обеспечивает более мощную, алгебраическую, замену выражений, в отличие от команды **subs**.

#### Пример 2.28.

> algsubs(a^2+b^3=sqrt(c), 
$$\sin(a^2+b^3)*(a^2+b^3+1)$$
); 
$$\sin(\sqrt{c})(\sqrt{c}+1)$$

В ряде задач встает необходимость упрощения выражения, т.е. приведения их к более простому виду. Для этого можно воспользоваться командой simplify.

#### ⊳ Команда **simplify**.

Назначение команды: упрощение выражений.

Формат вызова:

# simplify(<выражение>)

Параметр:

-<выражение>-выражение, которое следует упростить.

Пример 2.29.

- >  $e:=\cos(a)^5+\sin(a)^4+2*\cos(a)^2-2*\sin(a)^2-\cos(2*a)$ ;  $e:=\cos(a)^5+\sin(a)^4+2\cos(a)^2-2\sin(a)^2-\cos(2a)$
- > simplify(%);

$$\cos(a)^4(\cos(a)+1)$$

Упражнение 2.4. Присвойте некоторой переменной выражение

$$e^{\ln x + \ln 3} - 2x,$$

после чего попытайтесь упростить его.

На работу команды **simplify** влияет знание о предположениях на встречающиеся в выражении переменные, заданные в команде **assume**.

#### Пример 2.30.

simplify(a-abs(a)); a - |a|assume(a>0); simplify(a-abs(a));

Пример показывает, как предположение о положительности переменной a позволило упростить выражение, раскрыв выражение, стоящее под знаком модуля.

*Уравнение* представляет собой объект Maple вида

$$<$$
выражение $1>=<$ выражение $2>$ 

Здесь <выражение1> и <выражение2> — некоторые выражения. Данный объект соответствует обычному пониманию уравнения в математике. Следует правильно понимать запись в языке Maple, приведенную в следующем примере.

#### Пример 2.31.

> eq := 3\*z^2 = 12; 
$$eq := 3\,z^2 = 12$$

Это надо понимать так: содержимым переменной eq является уравнение  $3z^2=12$ . Правильность такого понимания можно проверить путем вызова функции **whattype** (ее описание будет приведено далее в этом разделе).

# Пример 2.32.

> whattype(eq);

 $Cucmema\ ypaвнений\ представляет\ собой\ множество\ из\ N\ уpавнений, т.е.$ конструкцию вида

> ${<$ выражениеA1>=<выражениеB1>,  $\langle$ выражение $A2\rangle = \langle$ выражение $B2\rangle,\ldots \}$

Знак равенства, который вернул Maple, как раз и говорит о том, что переменной 
$$eq$$
 присвоен объект «уравнение».

Запись системы в виде множества отражает тот факт, что решение системы уравнений не изменится от изменения порядка перечисления уравнений. Допустима следующая конструкция для присваивания системы уравнений некоторой переменной.

#### Пример 2.33.

> eqs := { 
$$2*x1-x2=4$$
,  $4*x1+2*x2=16$  };   
  $eqs := \{2x1-x2=4, 4x1+2x2=16\}$ 

При моделировании довольно часто возникает необходимость решения уравнений и их систем (например, при моделировании с использованием цепей Маркова, сетей Петри). Решение уравнений производится с помощью команды **solve**.

#### ⊳ Команда **solve**.

Назначение команды: решение алгебраических уравнений и их систем. Формат вызова 1:

Формат вызова 2:

$$solve({,,...}, {<перем1>,<перем2>,...})$$

#### Параметры:

- **<ypавнение**>,**<ypавн1**>,**<ypавн2**>,...— алгебраические уравнения;
- <перем>,<перем1>,<перем2>,...> переменные, относительно которых необходимо решить уравнения (необязательный параметр).

Отметим особенности применения данной команды:

- команда ищет решение аналитическими методами, а не численными,
   т.е. пытается определить точное решение;
- в случае отсутствия решения команда возвращает NULL (см. далее);
- если имеется один корень, результат представляет собой выражение;
- при наличии нескольких корней выводится последовательность выражений;
- если переменные, относительно которых ищутся решения, не заданы, то Maple ищет решения относительно всех неинициализированных переменных.

Проверку найденных решений можно выполнить с использованием команд **subs** или **eval**. Приведем пример решения уравнения с одним корнем и проверку правильности найденного решения.

#### Пример 2.34.

> solve(z+3=7, z);  $4 \\ > \text{subs(z=\%, z+3=7);} \\ 7 = 7$ 

В команде **subs** мы подставляем вместо переменной z корень уравнения, в результате подстановки Maple возвращает верное тождество:

$$7 = 7$$

А теперь посмотрим на ответ, полученный при решении уравнения, записанного в переменной eq, имеющего несколько корней.

#### Пример 2.35.

> res := solve(eq, z); res := 2, -2 > subs(z=op(1,{res}), eq); 12 = 12 > subs(z=op(2,{res}), eq); 12 = 12

При проверке правильности решения следует по очереди подставить корни в уравнение. Чтобы можно было извлечь корни по очереди, ответ, который возвращает команда **solve**, необходимо привести к типу «множество» (заключить в фигурные скобки) и далее последовательно применить команду выделения операндов из выражения **ор** (подробнее об этом будет сказано в разделе ниже).

**Упражнение 2.5.** Найдите точное и приближенное (в виде десятичных чисел с плавающей точкой) решение уравнения

$$x^2 - 5x + 5 = 0.$$

Проверьте правильность найденных решений.

Для решения системы уравнений надо использовать команду во втором формате.

# Пример 2.36.

> res := solve(eqs, {x1,x2}); 
$$res := \{x1 = 3, \, x2 = 2\}$$
 > subs(res, eqs); 
$$\{16 = 16, \, 4 = 4\}$$

Обратите внимание на ответ. Во-первых, ответ представляет собой множество уравнений. Во-вторых, порядок вывода переменных не обязательно совпадает с порядком их указания в команде. Так как ответ представляет собой множество, при проверке достаточно передать в качестве первого аргумента команды **subs** переменную *res*, которой был присвоен результат работы команды **solve**.

Упражнение 2.6. Найдите точные решения системы уравнений

$${x^2 = y, y = x + 2}.$$

Проверьте правильность найденных решений.

Полезными могут оказаться также следующие команды решения уравнений:

- **fsolve** поиск численных решений уравнения или системы уравнений;
- **isolve** поиск всех целочисленных решений;
- dsolve решение обыкновенных дифференциальных уравнений и их систем;
- **pdsolve** решение дифференциальных уравнений в частных производных и их систем;
- **linsolve** решение систем линейных алгебраических уравнений в матричной форме (будет рассмотрено в разделе, посвященном работе с векторами и матрицами).

В частности, команда **isolve** полезна при моделировании сетей Петри для проверки достижимости определенных разметок и для определения сохраняющих весов сети. Команды **dsolve** и **pdsolve** могут использоваться при моделировании процессов, описываемых с помощью дифференциальных уравнений и их систем.

Не всегда удобно для вычисления значений выражений пользоваться форматом команды **eval**. Значительно привычней и несколько проще для записи функций использовать такую форму:

$$f(x) = x + 3.$$

Тогда значение функции в точке x=3 можно представить как

$$f(3) = 6.$$

Для повышения эффективности работы с пакетом Maple в нем введен специальный тип объектов, называемых  $\phi y + \kappa y$ 

Для определения функции в Maple используется операторная форма записи.

⊳ Команда ->.

Назначение команды: определение функции в операторной форме. Формат вызова:

#### Параметры:

— < переменные > — одна или последовательность переменных, заключенных в круглые скобки, являющихся входными параметрами функции;

<1

— **<результат>** — правило преобразования входных переменных.

Покажем пример определения функции, зависящей от одной переменной.

#### Пример 2.37.

> 
$$x := 'x'$$
:  
>  $f := x -> x^2 + 2*x - 3$ ;  
 $f := x \rightarrow x^2 + 2x - 3$   
>  $f(x)$ ;  
 $x^2 + 2x - 3$   
>  $f(v)$ ;  
 $v^2 + 2v - 3$   
>  $f(2)$ ;

Здесь мы определили функцию и присвоили ее переменной f для того, чтобы можно было вызвать функцию. Для вызова определенной ранее функции необходимо записать имя переменной, ссылающуюся на эту функцию, и подставить в качестве параметров аргументы функции. Сравните приведенную форму определения функции и операторной формой записи функций, принятой в математике:

$$f: X \to Y$$
.

Пример определения и вызова функции, зависящей от нескольких переменных:

#### Пример 2.38.

> g := (x,y,z) -> x^2+x\*y-3\*z-2\*x\*y\*z; 
$$g := (x, y, z) \rightarrow x^2 + xy - 3z - 2xyz$$
 > g(x1,x2,x3); 
$$x1^2 + x1x2 - 3x3 - 2x1x2x3$$
 > g(1,2,3); 
$$-18$$

# Упражнение 2.7. Определите функцию

$$f(x,y,z) = \frac{x - 2y + 3z - 4}{x^2 + y^2 + z^2 + 1}.$$

Рассчитайте значение функции при значениях  $\{x=0,y=0,z=0\}$  и  $\{x=0.1,y=\frac{2}{3},z=-2\}.$ 

Еще раз уточните для себя различие понятий «выражение» и «функция» в Maple. Выражение — это совокупность переменных, констант, соединенных арифметическими операциями и с использованием математических функций. Функция же — правило преобразования входных переменных в выходную, правило вычисления.

Функцию можно преобразовать в выражение: для этого нужно в качестве параметров передать в функцию неинициализированные переменные. Наоборот, выражение можно преобразовать в функцию, для чего применяется команда **unapply**.

#### ⊳ Команда **unapply**.

Назначение команды: преобразование выражения в функцию (функциональный оператор).

Формат вызова 1:

Формат вызова 2:

Параметры:

- **<выражение>** выражение, которое необходимо преобразовать в функцию;
- -<перем1>,<перем2>,...- последовательность переменных, от которых зависит функция;

◁

— **<список\_перем>** — список переменных (последовательность переменных, заключенная в квадратные скобки).

Пример 2.39.

$$>$$
 f := unapply(x1^2+x2^3, x1,x2);

$$f := (x1, x2) \to x1^2 + x2^3$$

> f(2,1);

5

Вот еще один пример использования функции **unapply** с той разницей, что список переменных передается как список, а не как последовательность.

#### Пример 2.40.

> g:=unapply(x1^2+x2^3-x3^4, [x1,x2,x3]); 
$$g:=(x1, x2, x3) \to x1^2 + x2^3 - x3^4$$
 > g(2,4,3); 
$$-13$$

**Упражнение 2.8.** Объясните результат работы Maple при такой последовательности команд:

> 
$$f := x^2 + y^3: f(2, 1);$$
 
$$x(2, 1)^2 + y(2, 1)^3$$

Упражнение 2.9. Присвойте некоторой переменной выражение

$$\ln(xy) - x^{y+1} - 2.$$

Преобразуйте переменную в функцию. Рассчитайте значение функции (не выражения, т.е. без использования команды eval) при значениях переменных  $\{x=1,y=1\}$  и  $\{x=0.2,y=2\}$ .

В ряде случаев полезными могут оказаться функции для нахождения оптимумов математических функций (в терминах Maple — выражений) — **minimize** для вычисления минимума (более точно, инфинума) и **maximize** (более точно, супремума) — для максимума.

## ⊳ Команда **minimize**.

Назначение команды: определение минимума функции.

Формат вызова:

Параметры:

- **<выражение>** алгебраическое выражение;
- -<опция1>, <опция2>, ... опции (необязательные параметры).

## ⊳ Команда maximize.

Назначение команды: определение максимума функции.

Формат вызова:

maximize(<выражение>, <опция1>, <опция2>, ...)

Параметры:

- **<выражение>** алгебраическое выражение;
- -<опция1>,<опция $2>,\ldots-$ опции (необязательные параметры).

Наиболее часто в качестве опций задается диапазон изменения переменных в виде

Здесь < **перем**> — имя переменной в оптимизируемой функции, а < **диапа- зон**> — диапазон изменения значения указанной переменной в виде

где <**a**> и <**b**> — минимальная и максимальная граница изменения переменной соответственно. В этом случае оптимизация происходит в пределах указанного диапазона (по умолчанию диапазон — множество действительных чисел).

Оптимизация производится по всем неинициализированным переменным. Следует заметить, что данная функция по умолчанию возвращает только само значение оптимума, а значения переменных, доставляющих оптимум, выведены не будут. Их можно вывести, если в опциях указать ключевое слово location или равенство location=true.

Рассмотрим нахождение минимума функции без указания диапазонов изменений переменных.

## Пример 2.41.

- > х := 'x': у := 'y': # Очищаем переменные х и у
- > minimize(2\*(x+5)^2+3\*y^2-1);

-1

Чтобы вывести значения переменных, при которых функция достигает своего минимального значения, надо указать в опциях ключевое слово **location**.

# Пример 2.42.

> minimize(2\*(x+5)^2+3\*y^2-1, location); 
$$-1, \, \{[\{x=-5, \, y=0\}, \, -1]\}$$

Если задать диапазоны изменения переменных, оптимальное значение может стать другим, как в следующем примере.

## Пример 2.43.

> minimize(
$$2*(x+5)^2+3*y^2-1$$
, x=0..10, y=0..10);

В частности, использование этих функций может быть полезно при использовании метода статистических испытаний Монте-Карло при вычислении площадей фигур для определения границ, в которые заключена фигура.

Дифференцирование выражений производится с использованием команды **diff**.

#### ⊳ Команда **diff**.

Назначение команды: дифференцирование.

Формат вызова:

#### Параметры:

- <выражение> алгебраическое выражение;
- <**перем1**>, <**перем2**>, ... переменные, по которым следует произвести дифференцирование.

Приведем пример вычисления производной выражения, зависящего от одной переменной.

 $\triangleleft$ 

## Пример 2.44.

> diff(x^2-exp(-x)+3\*cos(x),x); 
$$2\,x + e^{(-x)} - 3\sin(x)$$

Если необходимо найти производную по двум переменным, их следует перечислить через запятую.

# Пример 2.45.

> diff(x\*y-y^3,x,y); # Сначала дифференцируем по переменной x, a затем по переменной y

1

Если указана переменная в виде  $\mathbf{x}$ \$ $\mathbf{n}$ , то будет определена производная n-го порядка. Покажем, как вычислить производную второго порядка.

# Пример 2.46.

 $> diff(x^2*y-y^2,x$2);$ 

Упражнение 2.10. Найдите следующую производную:

$$\frac{\partial^3}{\partial x^2 \partial y} \left( x^3 \sin y - x^3 y^3 - 2 \right).$$

Упражнение 2.11. Задайте функцию

$$y(x) = e^{x-2}x^3 + 2x - \frac{\sin x}{x^2 + 1}.$$

Найдите производную этой функции (в терминах Maple — выражения!) и преобразуйте получившееся выражение производной в функцию, зависящую от переменной x. Рассчитайте значение функции y(x) и ее производной при значениях x=0, x=1 и  $x=\pi$ . Приведите при необходимости результаты  $\kappa$  виду десятичных чисел c плавающей точкой.

Интегрирование осуществляется командой **int**. С помощью этой команды можно вычислять как определенные, так и неопределенные интегралы.

⊳ Команда **int**.

Назначение команды: интегрирование.

Формат вызова 1:

Формат вызова 2:

$$int(<выражение>, <перем>=a..b)$$

Параметры:

- **<выражение>** алгебраическое выражение;
- <перем> переменная, по которой необходимо производить интегрирование;
- $-{f a},{f b}$  интервал [a;b], по которому производится интегрирование.

 $\triangleleft$ 

Вычисление неопределенного интеграла

$$\int \sin x \, dx$$

производится с помощью первого формата вызова команды int.

# Пример 2.47.

> int(sin(x), x);

$$-\cos(x)$$

Приведем пример вычисления определенного интеграла

$$\int_0^\pi \sin x \, dx.$$

## Пример 2.48.

> int(sin(x), x=0..Pi);

2

Проверить, является ли выражение (функция) линейным по некоторому параметру (переменной), можно с помощью команды **type**.

## ⊳ Команда **type**.

Назначение команды: проверка линейности входящей в выражение переменной.

Формат вызова:

Параметры:

- <выражение> выражение;
- **<переменная**> параметр, относительно которого проверяется линейность.

◁

Эта функция возвращает истину (**true**), если функция линейна, и ложь (**false**), если функция нелинейна. Данная функция полезна при проверке линейности параметров, входящих в выражение, при идентификации моделей. Пример с использованием функции **type** будет приведен в следующем разделе при рассмотрении условного оператора.

В выражении для любого слагаемого можно определить коэффициент, на который умножается некоторая переменная или ее n-я степень. В частности, это может использовано при выделении базисных функций в линейных по параметрам функциях. Для этого используется оператор **coeff**.

# ⊳ Команда **coeff**.

Назначение команды: получение коэффициента при выражении. Формат вызова:

Параметры:

- **<выражение>** выражение;
- **<переменная**> переменная, при которой следует выделить коэффициент;

— **<степень**> — степень переменной (необязательный параметр). По умолчанию параметр равен единице.

 $\triangleleft$ 

◁

Пример с использованием функции **coeff** также будет приведен в следующем разделе при рассмотрении условного оператора.

Те, кто активно работает со справочной информацией в Maple, обратили внимание на наличие математических функциональных операторов, начинающихся с заглавной буквы, например, **Diff**. Приведем некоторые пояснения относительно различий записи команд с заглавной и строчной буквы.

Для некоторых команд существуют две формы ввода: активная и пассивная (inert). Активная форма приводит к непосредственной обработке команды — для этого команду следует писать со строчной буквы. В пассивной форме (команда вводится с прописной буквы) команда непосредственно не обрабатывается, а преобразуется в математическую нотацию. Выражение в пассивной форме можно вычислить далее, если использовать затем команду value.

#### ⊳ Команда **value**.

Назначение команды: преобразование пассивной формы команды в активную (получение значения).

Формат вызова:

# value(<выражение>)

Параметр:

— **<выражение>** — математическое выражение.

Например, команда дифференцирования в активной форме приводит к немедленному вычислению производной.

# Пример 2.49.

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

2x

При использовании команды в пассивной форме все будет выглядеть по-другому.

# Пример 2.50.

> Diff(x^2,x); 
$$\frac{d}{dx}(x^2)$$
 > value(%); 
$$2x$$
 
$$43$$

# 2.3. Базовые структуры данных: последовательности выражений, списки, множества, таблицы, массивы

Рассмотрим несколько базовых структур (способов организации) данных пакета Maple. К ним относятся, прежде всего, последовательность выражений ( $expression\ sequence$ ), называемая также просто последовательностью, список (list) и множество (set).

Последовательность выражений представляет собой набор элементов, разделенных друг от друга запятыми. В качестве элемента последовательности выражений может выступать любой объект Maple — число, константа, выражение либо любая другая структура данных.

## Пример 2.51.

> 1, 2, abc, x^2+1, "'hi world", Pi, x -> x^2, 1/2, 1; 
$$1, 2, \ abc, \ x^2+1, \ \text{``hi world''}, \ \pi, \ x \to x^2, \frac{1}{2}, \ 1$$

Как и любые другие объекты, последовательности выражений можно присваивать переменным.

## Пример 2.52.

> s := 1, a, Pi, x^2; 
$$s := 1, \, a, \, \pi, \, x^2 \label{eq:spin}$$

Для просмотра типа используемых данных и типа структур предназначена команда **whattype**.

# ⊳ Команда **whattype**.

Назначение команды: получение типа или структуры данных. Формат вызова:

# whattype(<объект>)

Параметр:

— <объект> — объект Maple или переменная, содержащая объект.

Посмотрим на результат выполнения команды **whattype** для последовательности из предыдущего примера.

<1

# Пример 2.53.

> whattype(s);

exprseq

Ответ Maple «exprseq» (от expression sequence) означает, что в переменной s содержится последовательность выражений.

Заметим, что в последовательностях порядок записи элементов важен, т.е. последовательности a,b,c и b,a,c являются разными. Можно соединить несколько последовательностей, перечислив их (или переменные, их содержащие) через запятую.

## Пример 2.54.

```
> \sec 1 := a, b, c; seq 1 := a, b, c > \sec 2 := 1, 2, 3; seq 2 := 1, 2, 3 > \sec 3 := \sec 1, \sec 2; seq 3 := a, b, c, 1, 2, 3
```

В данном примере последовательность seq1 соединяется с последовательностью seq2.

В Марlе выделяется особая последовательность, которая не содержит элементов, называемая пустой: **NULL** (обратите внимание на необходимость записи этого множества большими символами!). Эта последовательность играет важную роль. Если кажется, что Maple не выдает ничего на экран, на самом деле, он возвращает последовательность **NULL**. Например, это происходит при попытке решить уравнение, не имеющего корней.

## Пример 2.55.

```
> solve( cos(ln(x))=tan(x)^2, x );
> evalb( solve( cos(ln(x))=tan(x)^2, x ) = NULL );
true
```

Komanda **evalb** во второй строке этого примера сравнивает результат пустого решения уравнения со специальной последовательностью **NULL**.

Марlе возвращает также последовательности при выводе результата, содержащего несколько элементов. Например, это может происходить при решении уравнений, имеющих несколько корней.

# Пример 2.56.

> solve(
$$x^2-2*x-8=0$$
, x);  $4, -2$ 

Существует и другое важное применение последовательностей выражений: при передаче нескольких входных параметров в процедуру. То, что

последовательности могут содержать элементы различных типов данных, хорошо согласуется с аналогичной необходимостью, возникающей при передаче параметров в процедуры.

## Пример 2.57.

> 
$$f:=(x,y)$$
 ->  $x^2-y$ ; 
$$f:=(x,y)\to x^2-y$$
 >  $s:=2,3$ ;  $f(s)$ ; 
$$s:=2,3$$

Таким образом, последовательности являются основой для ввода команд и вывода результатов их исполнения. Для создания последовательностей значений одного типа по известному алгоритму может использоваться команда **seq**.

#### ⊳ Команда **seq**.

Назначение команды: создание последовательности.

Формат вызова:

$$seq(<выражение>, = ...)$$

Параметры:

- **<выражение>** алгоритм создания последовательности;
- -<і>- имя переменной;
- -<m>,<n> числовые значения пределы изменения переменной **i** (<m> минимальное, <n> максимальное).

# Пример 2.58.

> seq(1/n, n=1..10); 
$$1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}, \frac{1}{9}, \frac{1}{10}$$

Кроме этого, существует оператор \$, который генерирует последовательность заданного количества переданных выражений.

# Пример 2.59.

> x\$5;

Такая запись, как мы уже видели, применяется в команде diff при вычислении производных высоких порядков.

*Cnucoк* — это последовательность выражений, заключенная в квадратные скобки. Список, например, может быть получен из последовательности выражений.

## Пример 2.60.

> s := 1, a, Pi, x^2; 
$$s := 1, \, a, \, \pi, \, x^2$$
 > l := [s]; 
$$l := [1, \, a, \, \pi, \, x^2]$$

Если вызвать команду определения типа whattype и передать в качестве аргумента список, то Maple в качестве результата вернет строку list, которое и означает, что переданный объект является списком.

## Пример 2.61.

whattype(1);

list

Для того, чтобы узнать количество элементов в структуре данных, используется команда **nops** (от **n**umber of **op**erands — количество операндов).

## ⊳ Команда **nops**.

Назначение команды: количество элементов в структуре данных. Формат вызова:

Параметр:

$$-<$$
объект $>-$  структура данных.

 $\triangleleft$ 

При применении этой команды к последовательности последнюю необходимо предварительно заключать в квадратные скобки, т.е. преобразовывать к списку. Связано это с тем, что команда **nops** может принимать только один аргумент, а при передаче последовательности команде **nops** в неизменном виде Maple будет считать, что передается несколько параметров.

# Пример 2.62.

> nops([s]);

Извлечь конкретный элемент или несколько элементов можно с помощью команды  $\mathbf{op}$  (от  $\mathbf{op}$  erand), а также посредством использования квадратных скобок  $[\ ]$ .

## ⊳ Команда ор.

Назначение команды: извлечение одного или нескольких элементов из структуры данных.

Формат вызова 1:

$$\operatorname{op}(<$$
i $>$ , $<$ объект $>$ )

Формат вызова 2:

Формат вызова 3:

#### Параметры:

- **<объект**> структура данных;
- -<i>,<j>- целые числа, определяющие диапазон номеров операндов;

 $\triangleleft$ 

— **<список\_позиций>** — список целых чисел, записанных в возрастающем порядке, означающих номера позиций, из которых следует взять элементы.

## Пример 2.63.

> op(2,[s]); s[2];   
 
$$a$$
   
  $a$    
 > op(2..4,[s]); s[2..4];   
  $a, \pi, x^2$    
  $a, \pi, x^2$ 

Как видно, результаты работы обоих способов доступа к элементам указанной структуры данных совпадают (хотя при указании диапазонов это не всегда верно). В команде **ор** могут использоваться также отрицательные числа, которые показывают, что элементы необходимо считать, начиная с конца.

# Пример 2.64.

Если указывается диапазон элементов, то в качестве первого операнда команды **ор** могут совместно использоваться положительные и отрицательные индексы. При их указании следует пользоваться таким правилом: первый индекс должен указывать на элемент, предшествующий тому, который указан вторым. В следующем примере показано неправильное использование этого правила.

#### Пример 2.65.

```
> op(4..-4, [a, b, c, d, e] );
Error, improper op or subscript selector
```

А в следующем примере показано правильное использование этого правила.

#### Пример 2.66.

> op(3..-3, [a, b, c, d, e] ); 
$$c$$

Аналогичным образом можно использовать отрицательные индексы и при выборе элементов с помощью квадратных скобок.

## Пример 2.67.

$$>$$
 [a, b, c, d][-3..-1]; 
$$[b,\,c,\,d]$$

Так же, как и в последовательностях выражений, при использовании списков порядок указания элементов имеет важное значение.

Понятие *множества* в Maple аналогично соответствующему математическому понятию. В Maple множество внешне выглядит как последовательность выражений, заключенная в фигурные скобки. Тем не менее, множество состоит только из уникальных, отличимых друг от друга элементов. В следующем примере множество будет состоять из четырех элементов, хотя в последовательности их перечислено пять. Повторяющиеся элементы во множество не включаются.

# Пример 2.68.

> s := {a, b, 1, a, 2}; 
$$s := \{1,\,2,\,a,\,b\}$$

Другое отличие от последовательностей состоит в том, что порядок указания элементов не важен. Марlе включает процедуры для работы со множествами: union для объединения, intersect для пересечения, minus для разности.

## Пример 2.69.

> set1 := {A, B, C, D}; set2 := {C, D, E, F, G}; # Исходные множества

$$set1 := \{D, C, A, B\}$$
  
 $set2 := \{D, F, C, E, G\}$ 

- > set1 union set2; # Объединение множеств  $\{{
  m D},\,F,\,C,\,A,\,B,\,E,\,G\}$
- > set1 intersect set2; # Пересечение множеств  $\{\mathrm{D},\,C\}$
- > set1 minus set2; # Разность множеств-1  $\{A, B\}$
- > set2 minus set1; # Разность множеств-2  $\{F,\,E,\,G\}$

Как уже упоминалось выше, для последовательностей выражений вводился специальный вид — пустая последовательность **NULL**. Для списков и множеств необходимость введения подобных специальных объектов отсутствует. Пустой список и пустое множество выражаются соответственно как [] и {}, что эквивалентно записям [NULL] и {NULL}.

Последовательности выражений, списки и множества являются основными структурами данных. Существуют, однако, и другие структуры, например, таблицы и массивы.

Таблицы представляют собой наборы данных, состоящих из двух компонентов — содержимого таблицы (элементов, значений) и индексов (ключей) — объектов, позволяющих осуществлять доступ к содержимому (для чтения и записи). Индексы таблиц могут быть произвольными объектами, не только целыми числами. Таблицы создаются с помощью команды **table**.

# ⊳ Команда **table**.

Назначение команды: объявление таблицы.

Формат вызова:

Параметры:

- **<индекс\_функция>** индексирующая функция (необязательный параметр);
- **<значения>** список или множество начальных значений (необязательный параметр).

 $\triangleleft$ 

Если параметр **<значения>** представляет список или множество равенств, то левая часть равенства будет являться индексом, а правая часть — значением таблицы. В противном случае значениям последовательно ставятся в соответствие натуральные числа 1, 2, . . . . Доступ к значениям таблицы осуществляются с помощью квадратных скобок [ ].

#### Пример 2.70.

- > t := table([4=sin, "next"=cos, 7=tan]); # Создание таблицы  $t:= {
  m table}([4=\sin,\,7=\tan,\,\,{
  m ``next''}=\cos])$
- > t[4]; # Доступ к элементу таблицы по индексу 4

sin

> t["next"]; # Доступ к элементу таблицы по индексу-строке "next"

cos

> t[1]; # Попытка доступа к элементу по несуществующему индексу

 $t_1$ 

Из примера видно, что при обращении к неинициализированному элементу таблицы выводится имя таблицы с указанием индекса справа внизу (см. элемент  $t_1$ ). Понятие индексирующей функции в пособии рассмотрено не будет (справочная информация по этому понятию вызывается с помощью команды **?indexfcn**). Далее приведен пример индексации значений таблицы по умолчанию.

## Пример 2.71.

> table([3,5,7,9]);  $table([1=3,\,2=5,\,3=7,\,4=9])$ 

Простой записи переменной для вывода содержимого недостаточно. Вывести содержимое таблицы можно с помощью команды **print**.

# ⊳ Команда **print**.

Назначение команды: вывод выражений и содержимого структур данных на экран.

Формат вызова:

$$print($$
 <выражение1>, <выражение2>, ... )

Параметры:

-<выражение1>, <выражение2>, ...- выражения.

#### Пример 2.72.

- > t; # Так содержимое таблицы не выведется
  - t
- > print(t); # Команда print позволяет вывести содержимое таблицы

$$table([4 = \sin, 7 = \tan, "next" = \cos])$$

Массивы — таблицы с целочисленными значениями в качестве индексов. Частный случай двумерного массива — матрица (у матрицы индексы должны быть не просто целыми числами, но и начинаться с единицы), поэтому нижесказанное про массивы будет относится в равной мере и к матрицам. Массивы создаются с помощью оператора **аrray**.

## ⊳ Команда **array**.

Назначение команды: объявление массива.

Формат вызова:

## Параметры:

- **<индекс\_функция>** индексирующая функция аналогично параметру в команде **table** (необязательный параметр);
- **<границы>** последовательность диапазонов (необязательный параметр);

 $\triangleleft$ 

— <**значения**> — список начальных значений (необязательный параметр).

# Пример 2.73.

> a := array( 3..5, 1..2, [ [4,5],[-17,0],["test",9] ] );  $a := \operatorname{array}(3..5, 1..2, [$  (3,1)=4 (3,2)=5 (4,1)=-17 (4,2)=0 (5,1)= "test" (5,2)=9 ])  $= [3,2]; \ \# \ \Im \text{лемент c индексами } (3,2) \ - \ \operatorname{число} -17$  > a[5,1];  $\# \ \Im \text{лемент c индексами } (5,1) \ - \ \operatorname{строка}$ 

"test"

# 2.4. Контрольные вопросы к разделу 2

- 1. Простейшие объекты Maple и формы их записи.
- 2. Константы в Maple. Имеет ли значение регистр символов при обращении к константам?
- 3. Точные и приближенные типы чисел. Преобразование чисел точных типов в приближенные.
- 4. В чем отличие точности вычислений от точности отображения результатов? Каким образом задаются эти точности?
- 5. Булевы переменные. Проверка истинности высказываний.
- 6. Переменные и их значения. Присваивание, проверка наличия и очистка значений переменных. Как можно вывести список всех инициализированных переменных?
- 7. Предположения относительно переменных: назначение и работы с ними. Отличие предположений от ограничений на переменные.
- 8. Математические функции. Преимущества и недостатки механизма ввода математических функций с помощью палитр.
- 9. Составление математических выражений в Maple. Способы вычисления выражений, их преимущества и недостатки.
- 10. Упрощение выражений. Замена подвыражений. Чем отличается синтаксическая замена от алгебраической?
- 11. Уравнения и их системы как объекты Maple. Можно ли переменной присвоить уравнение или систему уравнений? Решение алгебраических уравнений и их систем.
- 12. Проверка правильности решений уравнений и их систем.
- 13. Функция (функциональный оператор) как объект Maple. Отличие функций от выражений. Каким образом можно преобразовать функцию в выражение и наоборот?
- 14. Оптимизация математических функций. Как определить точку, доставляющую математической функции экстремальное значение?
- 15. Работа с производными. Вычисление производных высоких порядков.
- 16. Вычисление неопределенных и определенных интегралов. Можно ли указывать бесконечные пределы при вычислении определенных интегралов?
- 17. Структуры данных в Maple. Отличия последовательностей выражений, списков и множеств. Как можно получить элемент структуры данных?
- 18. Специальные виды структур: пустые последовательность, список и множество. Приведите примеры их использования в Maple.
- 19. Области применения последовательностей выражений.
- 20. Применение таблиц и массивов. Всякая ли таблица является массивом и наоборот?

#### 3. Программные конструкции

# 3.1. Условный оператор

Maple, как и любой язык программирования, содержит условный оператор.

## ⊳ Команда **if**.

Назначение команды: условный оператор.

Формат вызова:

```
if <условие> then <операторы>
| elif <ycловиe2> then <oператоры2> |
| else <операторы3> |
end if
```

#### Параметры:

- -<условие>, <условие2> булевские выражения;
- <операторы>, <операторы2>, <операторы3> последовательности операторов пакета Maple.

Особенности условного оператора:

— указанное между вертикальными чертами | является необязательным;

◁

- конструкция **elif** может быть повторена в операторе произвольное количество раз;
- булевские выражения формируются с использованием операторов сравнения (<,<=,>,>=,=,<>), логических операторов (and — логическое «и»,  $\mathbf{or}$  — логическое «или»,  $\mathbf{not}$  — логическое отрицание, логических констант (true, false).

Рассмотрим пример использования оператора if для определения наибольшего из двух чисел.

## Пример 3.1.

```
restart: # Очищаем память
 a := 3; b := 6;
                           a := 3
                            b := 6
  if a > b then
> print("a больше b");
  elif a < b then
> print("a меньше b");
> print("a равно b");
   end if;
                         "а меньше b"
```

Данный формат записи не всегда удобен в использовании при выполнении вычислений. Например, стоит задача вычисления выражения

$$m^2 + \sin m + \frac{1}{m},$$

где m — наибольшее из чисел a и b. Возможное решение этой задачи приведено в следующем примере.

## Пример 3.2.

```
> a := 4; b := 2; a := 4 b := 2 > if a > b then m := a; > else m := b; > end if; m := 4 > m^2 + \sin(a) + 1/b; evalf(%); \frac{33}{2} + \sin(4) 15.74319750
```

Получилась довольно громоздкая конструкция, которая, в случае большего числа различных условий разрослась бы значительно сильнее. Для решения такой задачи значительно проще и понятней было бы использование сокращенной формы условного оператора.

#### ⊳ Команда **'if'**.

Назначение команды: условный оператор (сокращенная форма). Формат вызова:

Параметры:

- -<условие>- логическое выражение;
- **<значение\_истина>** значение, возвращаемое функцией, если условие истинно;
- **<значение\_ложь>** значение, возвращаемое функцией, если условие ложно.

Сокращенную форму условного оператора удобно использовать при вычислении выражений.

◁

#### Пример 3.3.

> 'if'(a>b,a,b)^2 + 
$$\sin(a)$$
 + 1/b; evalf(%); 
$$\frac{33}{2} + \sin(4)$$
 15.74319750

Рассмотрим пример проверки линейности параметров функции.

#### Пример 3.4.

```
> a := 'a': b:= 'b': # Очищаем переменные a и b
 f := (x,a,b) -> (a*x^b);
                      f := (x, a, b) \rightarrow a x^b
   # Проверяем линейность функции по параметру a if
  type(f(x,a,b),linear(a)) then
> print("линейна по a");
> else
> print("нелинейна по a");
   end if;
                        "линейна по а"
   # Проверяем линейность функции по параметру b if
> type(f(x,a,b),linear(b)) then
> print("линейна по b");
> else
> print("нелинейна по b");
   end if;
                       "нелинейна по b"
```

# **Упражнение 3.1.** Определите функции

$$f(a,x) = a_1 + a_2 e^{x_1} - 5\cos(a_3 x_2) + a_4 x_1 x_2$$

u

$$g(a,x) = \frac{a_1}{x_1 + 1} + a_2 \sin^{x_2 - 1} - a_3 + a_4 x_1^2.$$

Выясните, линейны ли функции f и g по вектору параметров а. Если функция нелинейна, выведите список параметров, по которым условие линейности не выполняется.

В следующем примере показано выделение двух базисных функций.

# Пример 3.5.

```
> f := (x,a,b) -> (a+b*exp(x^3)); f := (x,a,b) \to a + b \, e^{(x^3)}> # Выделяем базисную функцию при коэффициенте a % if type(f(x,a,b),linear(a)) then print(coeff(f(x,a,b),a)); end if;
```

```
> # Выделяем базисную функцию при коэффициенте b% if type(f(x,a,b),linear(b)) then print(coeff(f(x,a,b),b)); end if; e^{(x^3)}
```

**Упражнение 3.2.** Выделите в линейной по вектору параметров функции из предыдущего упражнения все базисные функции.

# 3.2. Оператор цикла

В пакете Maple циклы организуются с помощью оператора **for**.

▶ Команда for.

Назначение команды: цикл.

Формат вызова 1:

```
|for <ums>| |from <начальное_значение>| |by <шаг>| |to <конечное_значение>| |while <выражение>| do <последовательность_операторов> end do
```

Формат вызова 2:

```
|for <uмя>| |in <выражение>| |while <выражение>| do <последовательность_операторов> end do
```

# Параметры:

- -<имя>-имя переменной-параметра цикла;
- **<начальное\_значение>** начальное значение, присваиваемое параметру цикла;
- <конечное \_ значение > конечное значение параметра цикла, после превышения которого цикл прекращается;
- <шаг> приращение параметра цикла после завершения этапа цикла;
- **<выражение>** логическое выражение;
- **<последовательность\_операторов>** последовательность операторов пакета Maple, выполняемых в теле цикла.

Особенности оператора:

— указанное между вертикальными чертами | является необязательным;

◁

- в первом формате цикла объединены идеи использования двух типов циклов с параметром цикла и с предусловием, т.е. *for* и *while* могут использоваться одновременно;
- конструкция *in* используется при работе с перечисляемыми типами (массивами, списками и т.д.).

Рассчитаем, например, задачу расчета суммы целых чисел от 1 до 10, решаемую с помощью первого формата вызова оператора цикла.

## Пример 3.6.

```
s := 0; # Обнуляем переменную для накопления суммы
> for i from 1
> to 10 do
> s := s + i;
> end do;
> print("Сумма равна "||s);
                             s := 0
                             s := 1
                             s := 3
                             s := 6
                             s := 10
                             s := 15
                             s := 21
                             s := 28
                             s := 36
                             s := 45
                             s := 55
                        "Сумма равна 55"
```

Обратите внимание, что при работе цикла выводятся все промежуточные накопленные суммы, что является, конечно же, лишним.

Упражнение 3.3. Измените решение в предыдущем примере таким образом, чтобы промежуточные результаты вычислений не выводились.

Теперь найдем сумму всех четных чисел от 1 до 10.

## Пример 3.7.

```
> s := 0;

> for i from 2 by 2 to 10 do

> s := s + i;

> end do;

> print("Сумма равна "||s);

s := 0
```

$$s := 2$$
 $s := 6$ 
 $s := 12$ 
 $s := 20$ 
 $s := 30$ 
"Сумма равна 30"

**Упражнение 3.4.** Измените решение в примере таким образом, чтобы ключевое слово **by** не использовалось, а в теле цикла использовался условный оператор.

**Упражнение 3.5.** Измените решение в примере так, чтобы не использовались ни ключевое слово **by**, ни условный оператор в теле цикла.

Приведем пример вычисления с использованием ключевого слова in.

#### Пример 3.8.

```
> s := 0;

> for i in [2,4,6,8,10] do

> s := s + i;

> end do;

> print("Сумма равна "||s);

s := 0

s := 2

s := 6

s := 12

s := 20

s := 30

"Сумма равна 30"
```

В циклах могут также использоваться следующие управляющие операторы:

- **break** выход из выполняющегося цикла;
- **next** переход на следующий виток цикла.

Приведем пример расчета сумм чисел от 1 до 10 до тех пор, пока сумма не станет больше 25.

# Пример 3.9.

```
> s := 0;
> for i from 1 to 10 do
> s := s + i;
> if s > 25 then
> break;
> end if;
> end do;
> print("Сумма равна "||s);
```

s := 0 s := 1 s := 3 s := 6 s := 10 s := 15 s := 21 s := 28"Сумма равна 28"

**Упражнение 3.6.** Измените последний пример таким образом, чтобы не использовать оператор **break**, а использовать ключевое слово **while** в цикле **for**.

# 3.3. Процедуры и функции

При решении сложных задач процедуры позволяют повысить эффективность разработки и качество работы программ. Описание *процедуры* и функции (процедуры, возвращающей значение) имеет такой синтаксис:

## ⊳ Команда **proc**.

Назначение команды: определение процедуры или функции. Формат вызова:

```
proc (<формальные_аргументы>) local <локальные_переменные>; global <глобальные_переменные>; options <последовательность_опций>; description <строка>; <последовательность операторов> end proc
```

## Параметры:

- <формальные \_аргументы> список параметров процедуры, разделенных запятой, который может быть и пустым. После каждого параметра после двух символов двоеточия :: может указываться тип параметра;
- **<локальные\_переменные>** переменные процедуры, которые доступны только внутри процедуры (необязательный параметр);
- **<глобальные\_переменные>** переменные процедуры, которые доступны вне данной процедуры (необязательный параметр);

- <последовательность опций> последовательность дополнительных опций процедуры, задающих режим работы, т.е. указывающих на возможность ее дальнейшего использования или влияющих на эффективность работы процедуры (необязательный параметр);
- <**строка**> описание действий, которые выполняет процедура (необязательный параметр);
- **<последовательность\_операторов>** последовательность операторов пакета Maple, составляющих суть процедуры.

#### Особенности:

— для того, чтобы после создания процедуры к ней можно было обращаться, ее необходимо присвоить переменной;

 $\triangleleft$ 

— результат последнего выполненного оператора в процедуре является значением, возвращаемым функцией.

Приведем простейший пример процедуры, возвращающей квадрат передаваемого ей параметра.

## Пример 3.10.

```
> sq := proc(a)
> description 'Квадрат числа a';
> a^2;
> end proc;

sq := proc(a) description ' a'; a<sup>2</sup> end proc
```

В этом примере определяется процедура sq. Ее назначение описывается после ключевого слова **description**. Вызов этой функции очень прост.

# Пример 3.11.

> sq(3);

9

**Упражнение 3.7.** Напишите функцию, которая принимает один параметр x и возвращает значение  $x^2$ , если x>0, и 0 в противном случае.

Чтобы было меньше ошибок, для аргументов процедур желательно указывать типы параметров, к которым относятся, например, следующие:

- array массив;
- **integer** целое число;
- matrix матрица;
- **numeric** любое число;

```
    — operator — оператор;
    — positive — положительное число;
    — vector — вектор.
```

Полный список типов можно просмотреть в помощи Maple, набрав в командной строке ?type. В предыдущем примере укажем, что передаваемый параметр a может быть только числовым.

## Пример 3.12.

```
> sq := proc(a::numeric)
> description 'Квадрат числа a';
> a^2;
> end proc;
sq := proc(a::numeric) description 'a'; a<sup>2</sup> end proc
```

Вызовем функцию sq, передав ей в качестве параметра строку.

#### Пример 3.13.

```
> sq("test"); #Передача ошибочного параметра
Error, sq expects its 1st argument, a, to be of type numeric, but received test
```

При объявлении процедуры аргументы у нее могут не перечисляться (или перечисляться частично), но при обращении к функции — передаваться необходимое количество фактических параметров. Обращение к параметрам производится с помощью функции  $\operatorname{args}[i]$ , а количество параметров передается в переменной  $\operatorname{nargs}$ . Это позволяет писать процедуры с произвольным количеством параметров. Приведем пример процедуры вывода максимума из набора переданных чисел, которая в случае отсутствия переданных параметров будет возвращать  $-\infty$ .

# Пример 3.14.

```
> mymax := proc()
> description 'Maκcumym чисел';
> local i,m;
> if nargs = 0 then
> -infinity
> else
> m := args[1];
> for i from 2 to nargs do
> if m < args[i] then
> m := args[i];
> end if;
> end o;
> m;
> end if;
> end proc;
```

```
mymax := \mathbf{proc}()
\mathbf{local}\,i,\,m;
\mathbf{description'}\,';
\mathbf{if}\,\mathbf{nargs} = 0\,\mathbf{then}\,-\infty
\mathbf{else}
m := \mathbf{args}_1;
\mathbf{for}\,i\,\mathbf{from}\,2\,\mathbf{to}\,\mathbf{nargs}\,\mathbf{do}\,\mathbf{if}\,m < \mathbf{args}_i\,\mathbf{then}\,m := \mathbf{args}_i\,\mathbf{end}\,\,\mathbf{if}\,\mathbf{end}\,\,\mathbf{do};
m
\mathbf{end}\,\,\mathbf{if}
\mathbf{end}\,\,\mathbf{proc}
```

При таком подходе в процедуре обязательно следует проверять тип входящих переменных. Напомним, что проверка типа осуществляется с помощью команды Maple **type**. Если тип параметров неверный, то следует вызвать процедуру, возвращающую сообщение об ошибке **error**.

```
⊳ Команда error.
```

Назначение команды: определение процедуры или функции.

Формат вызова:

```
error \langle \text{текст} \rangle, \langle \text{параметр1} \rangle, \langle \text{параметр2} \rangle, ...
```

## Параметры:

- -<**текст**> текст сообщения об ошибке;
- <**параметр1>**, <**параметр2>**, ... параметры для замены в строке <**текст**>.

При вызове команды **error** возникает исключение и происходит принудительный выход из процедуры. Обратите внимание на то, что аргументы этой процедуры записываются не в скобках, а указываются непосредственно за ключевым словом **error**.

Напишем новую процедуру определения максимального числа *тутах2*.

◁

## Пример 3.15.

```
> mymax2 := proc()
> description 'Максимум чисел';
> local i,m;
> if nargs = 0 then
> -infinity
> else
```

```
for i from 1 to nargs do
   if type(args[i], numeric) = false then
   error "Параметр номер %1 не является числом", i;
   end if;
> end do;
> m := args[1];
> for i from 2 to nargs do
> if m < args[i] then</pre>
> m := args[i];
> end if;
> end do;
> m;
> end if;
> end proc;
    mymax2 := \mathbf{proc}()
    local i, m;
    description ';
       if nargs = 0 then -\infty
       else
          for i to narge do
             if type(args_i, numeric) = false then error
    "Параметр \epsilon%1 не является числом", i end if
          end do:
          m := \operatorname{args}_1;
          for i from 2 to narge do if m < \arg s_i then m := \arg s_i end if end do;
          m
       end if
    end proc
```

Посмотрите на разные сообщения об ошибках, выводимые процедурами mymax и mymax2.

# Пример 3.16.

```
> mymax(2, 4, "строка", 1, 3);
Error, (in mymax) invalid terms in sum
> mymax2(2, 4, "строка", 1, 3);
Error, (in mymax2) Параметр номер 3 не является числом
```

Как видно, сообщение, выводимое процедурой mymax2, более информативно. Рассмотрим еще один способ проверки параметров, передаваемых в процедуру, который основан на втором формате вызова оператора цикла for — с использованием ключевого слова in.

#### Пример 3.17.

**Упражнение 3.8.** Разработайте функцию, принимающую произвольное количество чисел и возвращающую сумму их квадратов. Если хотя бы один из аргументов не будет являться числом, следует вывести сообщение об ошибке с указанием номера параметра и самого значения, которое вызвало ошибку.

В заголовке процедуры рекомендуется явно объявлять глобальные и локальные переменные после ключевых слов **global** и **local** соответственно. Если в процедуре глобальные и локальные переменные не объявлены, то используются следующие правила определения типа переменной:

- переменная локальна, если она появляется в левой части оператора присваивания, или как индексная переменная цикла оператора **for**, или как индексная переменная в командах **seq**, **add** или **mul**;
- переменная считается глобальной в остальных случаях.

В разделе **options** могут содержаться следующие зарезервированные слова (перечислены основные):

- **arrow** используется вместе с ключевым словом **operator**, позволяет использовать стрелочную нотацию задания функционального оператора;
- **builtin** встроенные в ядро функции (пользователю недоступна возможность использования такого модификатора);
- call external указание на внешнюю функцию;
- **operator** указывает на то, что процедура является функциональным оператором;

- **remember** результаты вычислений заносятся в таблицу значений для возможности повторного использования результатов (эффективно при частом вызове процедуры с одинаковыми параметрами, в частности, при реализации рекурсивных процедур);
- system используется вместе с ключевым словом remember и указывает, что таблица значений удаляется во время процедуры сборки мусора системой Maple (вручную такая сборка мусора выполняется командой gc — garbage collection);
- **trace** в максимальной степени выводит информацию о работе процедуры в целях проведения отладки.

#### Пример 3.18.

```
> fibo := proc(n::integer)
> description 'Вычисление чисел Фибоначчи (долгая)';
> if n < 2 then
> n;
> else
> fibo(n-1) + fibo(n-2);
> end if;
> end proc:
```

Вызовем функцию fibo и определим время работы этой процедуры с помощью команды time.

#### ⊳ Команда **time**.

Назначение команды: время работы с момента начала сессии Maple. Формат вызова:

```
time(<выражение>)
```

Параметр:

— **<выражение>** — выражение Maple или вызов процедуры.

## Пример 3.19.

```
> fibo(32);
```

2178309

◁

> time(fibo(32)); #Расчет времени выполнения процедуры 9.000

С опцией **remember** процедура вычисления числа Фибоначчи выполняется быстрее. Такая опция полезна при разработке рекурсивных процедур, т.е. процедур, вызывающих в своем теле саму себя. Опция **remember** приводит к тому, что все промежуточные результаты работы процедуры Марle сохраняет в таблице, запоминая результат при каждом вызове. В отсутствии этой опции процедура вычисляет значения при обращении к ней каждый раз заново.

#### Пример 3.20.

```
> fibo_quick := proc(n::integer)
> options remember;
> description 'Вычисление чисел Фибоначчи (быстрая)';
> if n < 2 then
> n;
> else
> fibo_quick(n-1) + fibo_quick(n-2);
> end if;
> end proc:
```

Теперь рассчитаем значение функции и время ее работы. Посмотрите на время, затрачиваемое процедурами *fibo* и *fibo* quick.

## Пример 3.21.

> fibo\_quick(32);

2178309

> time(fibo\_quick(32)); #Расчет времени выполнения процедуры 0.

Возврат значения функции и досрочное завершение ее процедуры производится с помощью оператора **return**.

#### ⊳ Команда **return**.

Назначение команды: возврат значения и завершение работы функции. Формат вызова:

# return <выражение1>,<выражение2>,...

# Параметр:

-<выражение1>,<выражение $2>,\ldots-$  последовательность возвращаемых выражений (может отсутствовать или, если точнее, быть  $\mathbf{NULL}$ ).

Отловить и обработать исключительную ситуацию (например, ошибку) в процедуре можно с помощью оператора **try**.

 $\triangleleft$ 

## ⊳ Команда **try**.

Назначение команды: обработка исключительных ситуаций. Формат вызова:

```
try <команды>
| catch <исключения>: <команды_при_исключении> |
| finally <завершающие_команды> |
end
```

## Параметры:

- **<команды>** команды языка Maple, в которых может возникнуть и должна обработаться исключительная ситуация (например, деление на 0);
- **<исключения>** список исключений, при которых вызывается их обработка;
- **<команды\_при\_исключении>** команды, которые будут отработаны при возникновении исключений;
- **<завершающие\_команды>** операторы, которые выполняются в любом случае при отсутствии возникновения исключительной ситуации и при ее возникновении.

Приведем пример использования оператора **try**.

## Пример 3.22.

**Упражнение 3.9.** Разработайте процедуру, реализующую метод Ньютона для решения нелинейных уравнений. Найдите с ее помощью корень уравнения

$$f(x) = x^4 - x^3 + \frac{x}{x^2 + 1} - 1$$

на отрезке [-3;0].

# 3.4. Контрольные вопросы к разделу 3

- 1. Условный оператор: полная и сокращенная форма записи. В каких случаях удобнее использовать сокращенную форму условного оператора?
- 2. Возможности оператора цикла в Maple. Сравните оператор цикла в Maple и в других известных вам языках программирования.

- 3. Управление ходом циклического процесса: прерывание выполнения цикла и переход на следующий виток. Приведите примеры, демонстрирующие целесообразность использования операторов управления ходом циклического процесса.
- 4. Определение пользовательских процедур и функций в Maple. Каким образом указывается значение, возвращаемое функцией? Как можно выдать сообщение об ошибке?
- 5. Описание аргументов пользовательских процедур, указание типов аргументов. Процедуры с заранее неизвестным (переменным) количеством аргументов, целесообразность их использования.
- 6. Опции процедур. Эффективная организация рекурсивных вычислений.
- 7. Что такое исключительная ситуация? Обработка исключительных ситуаций.

# 4. Векторно-матричные операции

# 4.1. Подключаемые пакеты

В пакете Марlе команды работы с простейшими объектами, другие базовые команды обрабатываются ядром пакета, тогда как для работы с более сложными объектами требуется подключение *пакетов* (*packages*), включающих в себя процедуры и функции для решения определенного круга задач. В Марlе имеется несколько пакетов, некоторые из которых, наиболее полезные при моделировании, перечислены в табл. 4.1.

Таблица 4.1. Некоторые пакеты Maple

Наименование	Описание
combinat	Комбинаторные функции (перестановки, сочетания)
DEtools	Процедуры для работы с дифференциальными
	уравнениями
finance	Команды для выполнения финансовых вычислений
linalg	Векторно-матричные операции (1)
LinearAlgebra	Векторно-матричные операции (2)
Matlab	Доступ к функциям пакета Matlab
networks	Процедуры для работы с сетями и графами
PDEtools	Процедуры для работы с дифференциальными
	уравнениями в частных производных
plots	Работа с графикой и анимацией
plottools	Основные графические объекты и структуры
simplex	Линейное программирование (симплекс-метод)
stats	Статистические процедуры
student	Команды для выполнения пошаговых
	дифференциальных и интегральных вычислений
VectorCalculus	Векторные вычисления

Подключение функциональности пакетов осуществляется с помощью команды  ${f with}.$ 

## ⊳ Команда with.

Назначение команды: подключение процедур пакета.

Формат вызова:

$$with(<$$
пакет $>$ ,  $<$ процедура $1>$ ,  $<$ процедура $2>$ ,  $\ldots)$ 

# Параметры:

— **<пакет>** — наименование подключаемого пакета;

— <процедура1>, <процедура2>, ...) — перечень подключаемых процедур (необязательные параметры).

◁

Особенности команды with:

- подключение пакетов возможно только на верхнем уровне иерархии программы, т.е. в теле пользовательской процедуры подключать пакеты нельзя;
- при указании в команде параметров <процедура1>, <процедура2>, . . . будут доступны для использования только перечисленные процедуры;
- возможна разработка пользовательских пакетов.

Рассмотрим пример расчета числа Фибоначчи с помощью определенной в пакете **combinat** функции **fibonacci**.

## Пример 4.1.

- > restart: #Очищаем память
- > with(combinat, fibonacci); # Подключение функции fibonacci из пакета combinat

[fibonacci]

- > fibonacci(32); # Вызов функции fibonacci 2178309
- > numbcomb(5, 2); # Попытка вызова неподключенной функции  $\mathrm{numbcomb}(5,\,2)$

В данном примере при подключении пакета мы сделали доступной только одну функцию **fibonacci**, которая и была успешно вызвана. При попытке вызвать функцию **numbcomb** для вычисления числа сочетаний  $C_5^2$  Maple переписал нам наименование функции, что говорит о том, что указанная функция Maple неизвестна.

Функцию пакета можно вызвать без подключения соответствующего пакета, если использовать следующий синтаксис:

В этой записи:

- -<пакет>- наименование пакета, в котором определена процедура;
- <**процедура**> наименование вызываемой процедуры;
- **<параметры>** параметры процедуры.

# Пример 4.2.

> combinat[numbcomb](5, 2); # Непосредственный вызов функции из пакета

Можно также подключить сначала весь пакет, а затем с помощью обычного синтаксиса вызывать определенные в нем процедуры.

## Пример 4.3.

> with(combinat); # Подключение всех процедур пакета combinat Warning, the protected name Chi has been redefined and unprotected

[Chi, bell, binomial, cartprod, character, choose, composition, conjpart, decodepart, encodepart, fibonacci, firstpart, graycode, inttovec, lastpart, multinomial, nextpart, numbcomb, numbcomp, numbpart, numbperm, partition, permute, powerset, prevpart, randcomb, randpart, randperm, setpartition, stirling1, stirling2, subsets, vectoint]

> numbcomb(5, 2); # Вызов подключенной функции 10

Обратите в приведенном выше примере внимание на сообщения, выдаваемые Maple при подключении пакета. В первой строке написано предупреждение, что имя *Chi* было переопределено и теперь имеет иной смысл, чем до подключения пакета (из-за совпадения имен). Далее в квадратных скобках указывается список процедур, ставших доступными для использования.

Узнать, какие пакеты подключены в настоящий момент, можно с помощью команды **packages**.

# ⊳ Команда **packages**.

Назначение команды: вывод списка подключенных пакетов. Формат вызова:

## packages()

◁

# Пример 4.4.

> packages();

# [combinat]

Векторно-матричные вычисления широко применяются во многих математических и компьютерных дисциплинах. Моделирование не является исключением из этого списка: моделирование систем типа «черный ящик», конечных автоматов, цепей Маркова, сетей Петри и других систем активно используют аппарат векторов и матриц.

Для работы с матрицами можно подключить один из двух пакетов: linalg или LinearAlgebra (операции с векторами из пакета LinearAlgebra

вынесены в пакет VectorCalculus, поэтому в дальнейшем при ссылке на пакет LinearAlgebra будет пониматься и пакет VectorCalculus). Пакет linalg был включен в Maple очень давно, он в большей степени ориентирован на символьные вычисления. Пакет же LinearAlgebra был введен в состав Maple, начиная с 6-й версии, и основан он на известной библиотеке численных расчетов NAG. Хотя разработчики рекомендуют применять более современный пакет LinearAlgebra, многие приложения были написаны с подключенным пакетом linalg.

В данном пособии внимание будет уделено обоим пакетам (заметим, что большинство процедур этих двух пакетов очень похожи по своей функциональности). При выборе пакета можно пользоваться следующими рекомендациями разработчиков Maple:

- пакет **linalg** предпочтительней использовать при нечисловых расчетах;
- в пакете **LinearAlgebra** имеется несколько специальных конструкторов для создания матриц;
- интерфейс вызовов процедур пакета **LinearAlgebra** более дружелюбен, чем интерфейс пакета **linalg**;
- пакет **LinearAlgebra** эффективней при численных расчетах, особенно в случае матриц больших размеров.

## 4.2. $\Pi$ aket linalg

Подключим пакет linalg с помощью команды with.

## Пример 4.5.

> with(linalg);

Warning, the name fibonacci has been redefined

Warning, the protected names norm and trace have been redefined and unprotected

Перед использованием матрицы или вектора их необходимо объявить и при необходимости определить их содержимое. Для этого предназначены команды **matrix** для матриц и **vector** для векторов. Рассмотрим их последовательно, приведя лишь наиболее важные форматы вызова.

#### ⊳ Команда **matrix**.

Назначение команды: объявление матрицы.

Формат вызова 1:

 $\operatorname{matrix}(< \operatorname{кол\_строк}>, < \operatorname{кол\_столбцов}>)$ 

Формат вызова 2:

matrix(<кол строк>, <кол столбцов>, <элементы>)

Формат вызова 3:

matrix(<кол строк>, <кол столбцов>, <функция>)

Параметры:

- -<кол\_строк>- количество строк матрицы (положительное целое число);
- **<кол\_столбцов>** количество столбцов матрицы (положительное целое число);
- **<элементы>** список элементов матрицы (значения перечислены в квадратных скобках), заполняемых построчно (сначала первая строка, затем вторая и т.д.);

 $\triangleleft$ 

— **<функция>** — функция (шаблон) для заполнения матрицы элементами.

Для того, чтобы к матрице можно было обращаться, ее (как и процедуру) необходимо присвоить некоторой переменной. Рассмотрим пример использования первого формата команды **matrix**.

## Пример 4.6.

```
> a := matrix(2, 5); a := \mathrm{array}(1..2, \, 1..5, \, []) > type(a, 'matrix'); # Действительно ли в переменной "a" содержится матрица?
```

Доступ к элементам вектора или матрицы осуществляется с помощью указания индексов в квадратных скобках [], как и при доступе к элементам списков.

true

## Пример 4.7.

> a[2,4];

 $a_{2,4}$ 

В данном примере Maple вернул нам элемент  $a_{2,4}$ , указывающий на то, что этому элементу никакого значения присвоено еще не было (при объявлении использовался первый формат вызова процедуры **matrix**).

Теперь посмотрим, как при объявлении функции заполнить ее некоторыми элементами. Для этого необходимо воспользоваться вторым форматом вызова.

#### Пример 4.8.

> a := matrix(2, 4, [1, -2, 0, 2.1, 4.3, -3.4, 0, 6.2]); 
$$a := \begin{bmatrix} 1 & -2 & 0 & 2.1 \\ 4.3 & -3.4 & 0 & 6.2 \end{bmatrix}$$

Заметим, что элементы могут записываться не только в виде одного списка, но и виде списка списков, т.е. в виде списка строк.

Теперь изменим элемент, стоящий в 1-й строке и 3-ем столбце, на значение 999, после чего попытаемся вывести матрицу на экран, указав ее имя. При этом Maple содержимого матрицы не выведет! В этом проявляется специфика работы с матрицами в пакете Maple — пока не просят, Maple выводит результаты в символьном виде. Для вывода содержимого следует переменную, содержащую матрицы, передать в качестве параметра команде **print** (как и для вывода таблицы).

#### Пример 4.9.

Упражнение 4.1. Объявите функцию

$$f(x,y) = x^3 - \sin(x * y).$$

Объявите матрицу, состоящую из 5 строк и 2 столбцов. Заполните ее элементами на основе формулы  $a_{ij} = f(i,j)$  с помощью двух циклов — внешнего по строкам и внутреннего по столбцам.

Предыдущее упражнение может быть решено много проще, если использовать третий формат команды **matrix**.

## Пример 4.10.

> a := matrix(3, 4, (i,j)->evalf(i^3-sin(i\*j))); 
$$a := \begin{bmatrix} 0.1585290152 & 0.0907025732 & 0.8588799919 & 1.756802495 \\ 7.090702573 & 8.756802495 & 8.279415498 & 7.010641753 \\ 26.85887999 & 27.27941550 & 26.58788151 & 27.53657292 \end{bmatrix}$$

Напомним, что вызвать справочную информацию по данной функции можно с помощью такой команды:

## ?linalg[matrix]

Такой способ вызова справки позволит получить информацию конкретно по команде **matrix** пакета **linalg**.

Аналогично команде **matrix** команда **vector** позволяет объявить и при необходимости заполнить содержимое вектора.

#### ⊳ Команда **vector**.

Назначение команды: объявление вектора.

Формат вызова 1:

$$vector(<$$
кол элементов $>)$ 

Формат вызова 2:

$$vector(<$$
кол элементов $>$ ,  $<$ элементы $>$ )

Формат вызова 3:

#### Параметры:

- **<кол\_элементов>** количество элементов вектора (положительное целое число);
- **<элементы>** список элементов вектора (значения перечислены в квадратных скобках), заполняемых построчно (сначала первая строка, затем вторая и т.д.);

 $\triangleleft$ 

— **<функция>** — функция (шаблон) для заполнения вектора элементами.

## Пример 4.11.

```
> v := vector(5, [1, -2.1, 10.5, 0, 4]); v := [1, -2.1, \, 10.5, \, 0, \, 4]
```

> type(v, 'vector'); # Действительно ли в переменной "v" содержится вектор?

true

Матрицы и векторы, так же как и числа, можно складывать (вычитать) и умножать на числа. Рассмотрим пример умножения матрицы на число.

## Пример 4.12.

$$>$$
 b :=  $3*a$ ;

$$b := 3 \, a$$

К сожалению, в этом примере Maple не вывел нам содержимое матрицы b. В этом проявляется особенность команд пакета linalg — все матричные выражения, содержащие только арифметические операции и степени матриц, Maple пытается оставить в символьном виде. Чтобы преобразовать такие матричные выражения к числовому виду, следует воспользоваться функцией evalm (от англ. evaluate as matrix — ewaluanumb как матрицу).

#### ⊳ Команда **evalm**.

Назначение команды: преобразование матричных (векторных) выражений к матрицам (векторам), содержащим числовые значения. Формат вызова:

```
evalm(<выражение>)
```

#### Параметр:

— **<выражение>** — матричное выражение.

◁

#### Пример 4.13.

```
> b := evalm(3*a); b := \begin{bmatrix} 0.4755870456 & 0.2721077196 & 2.576639976 & 5.270407485 \\ 21.27210772 & 26.27040748 & 24.83824649 & 21.03192526 \\ 80.57663997 & 81.83824650 & 79.76364453 & 82.60971876 \end{bmatrix}
```

Теперь приведем пример сложения матриц (обратите внимание на использование команды **evalm**).

#### Пример 4.14.

```
c := evalm(a + b); c := \begin{bmatrix} 0.6341160608 & 0.3628102928 & 3.435519968 & 7.027209980 \\ 28.36281029 & 35.02720998 & 33.11766199 & 28.04256701 \\ 107.4355200 & 109.1176620 & 106.3515260 & 110.1462917 \end{bmatrix}
```

Некоторые команды для работы с векторами и матрицами приведены в табл. 4.2.

Рассмотрим часть команд для работы с матрицами подробней. Транспонирование матрицы осуществляется с помощью команды **transpose**.

## ⊳ Команда **transpose**.

Назначение команды: транспонирование матрицы (вектора). Формат вызова:

# transpose(<матрица>)

Таблица 4.2. Команды работы с матрицами пакета **linalg** 

Команда	Описание	
Команды создания специальных видов матриц и векторов		
band	Ленточная матрица	
diag	Диагональная матрица	
randmatrix	Случайная матрица	
randvector	Случайный вектор	
Команды работы с размерами и содержимым		
col	Выделение столбца из матрицы	
coldim	Количество столбцов в матрице	
delrows	Удаление строк матрицы	
delcols	Удаление столбцов матрицы	
extend	Добавление строк и столбцов к матрице	
$\operatorname{rowdim}$	Количество строк в матрице	
row	Выделение строки из матрицы	
submatrix	Выделение подматрицы из матрицы	
vectdim	Количество элементов в векторе	
	Математические операции	
&*	Умножение матриц и векторов	
det	Определитель (детерминант) матрицы	
grad	Вычисление градиента	
inverse	Обратная матрица	
jacobian	Вычисление матрицы Якоби	
linsolve	Решение системы линейных алгебраических уравнений	
transpose	Транспонированная матрица	

#### Параметр:

```
-<матрица>- матрицы или вектор.
```

◁

## Пример 4.15.

$$at := \begin{bmatrix} 0.1585290152 & 7.090702573 & 26.85887999 \\ 0.0907025732 & 8.756802495 & 27.27941550 \\ 0.8588799919 & 8.279415498 & 26.58788151 \\ 1.756802495 & 7.010641753 & 27.53657292 \end{bmatrix}$$

Умножение матриц и векторов осуществляется командой &\*. Заметьте, что перед символом \* (звездочка) обязательно необходимо указывать символ & (амперсанд). Использование амперсанда говорит пакету Maple о том, что

умножение матриц — некоммутативная операция и это необходимо учитывать при оптимизации матричных выражений.

Напомним, что умножать можно только матрицы согласованных размеров (количество столбцов матрицы, стоящей слева, должно совпадать с количеством строк матрицы, стоящей справа). При этом произведение матриц AB в общем случае не равно произведению матриц BA.

#### Пример 4.16.

```
> d1 := evalm(a &* at); d1 := \begin{bmatrix} 3.857388252 & 21.34568385 & 77.94434445 \\ 21.34568385 & 244.6574717 & 842.5099492 \\ 77.94434445 & 842.5099492 & 2930.744235 \end{bmatrix}
> d2 := evalm(at &* a); d2 := \begin{bmatrix} 771.7026287 & 794.8008081 & 772.9637489 & 789.5903871 \\ 794.8008081 & 820.8563269 & 797.8809759 & 812.7317658 \\ 772.9637489 & 797.8809759 & 776.2018390 & 791.6920365 \\ 789.5903871 & 812.7317658 & 791.6920365 & 810.4983010 \end{bmatrix}
```

Обратите внимание, что при использовании команд **transpose** и **inverse** нет необходимости применять команду **evalm**, а при умножении — есть. В принципе, лишнее использование команды **evalm** не приведет к возникновению сообщения об ошибке.

Для квадратных матриц существуют специальные операции, например, вычисление определителя (детерминанта) и обращение.

#### ⊳ Команда det.

Назначение команды: определитель квадратной матрицы.

Формат вызова:

## Параметры:

- **<матрица>** квадратная матрица,
- **sparse** ключевое слово, указывающее, что матрица разреженная (необязательный параметр).

 $\triangleleft$ 

Ключевое слово **sparse** позволяет увеличить скорость вычисления определителей разреженных матриц.

#### ⊳ Команда **inverse**.

Назначение команды: обращение квадратной матрицы.

Формат вызова:

## inverse(<матрица>)

Параметр:

— <матрица> — квадратная невырожденная матрица.

 $\triangleleft$ 

При попытке найти обратную матрицу к вырожденной, т.е. к той, у которой определитель равен нулю, будет выдано соответствующее сообщение об ошибке (по-английски вырожденная матрица обозначается термином singular). Приведем пример вычисления определителя и обратной матрицы к матрице d1.

## Пример 4.17.

```
> det(d1);
```

> d1\_1 := inverse(d1); # Определитель не равен нулю, значит, существует обратная

$$d1\_1 := \begin{bmatrix} 0.7539391592 & 0.3254283053 & -0.1136031849 \\ 0.3254283053 & 0.5472063772 & -0.1659619790 \\ -0.1136031849 & -0.1659619790 & 0.05107212787 \end{bmatrix}$$

- > evalm(d1 &\* d1\_1); # Проверка 1
- > evalm(d1\_1 &\* d1); # Проверка 2

$$\begin{bmatrix} 1.000000002 & -0.110^{-7} & 0. \\ -0.2210^{-6} & 0.9999999 & 0.110^{-7} \\ -0.710^{-6} & -0.210^{-6} & 1.0000000 \end{bmatrix}$$

$$\begin{bmatrix} 1.000000002 & -0.1 \, 10^{-7} & 0. \\ -0.22 \, 10^{-6} & 0.9999999 & 0.1 \, 10^{-7} \\ -0.7 \, 10^{-6} & -0.2 \, 10^{-6} & 1.0000000 \end{bmatrix}$$

$$\begin{bmatrix} 1.000000002 & -0.22 \, 10^{-6} & -0.7 \, 10^{-6} \\ -0.1 \, 10^{-7} & 0.9999999 & -0.2 \, 10^{-6} \\ 0. & 0.1 \, 10^{-7} & 1.0000000 \end{bmatrix}$$

Мы выполнили также проверку правильности вычислений (для этого необходимо обратную матрицу два раза умножить на исходную — один раз слева, а другой раз справа). Как видно, при проверке фактически получились единичные матрицы (погрешность вызвана округлениями).

**Упражнение 4.2.** Найдите обратную к матрице d2. Объясните резульmam.

Выделение из матрицы некоторой ее подматрицы возможно с помощью команд **submatrix**, **row**, **col**. Их применение приводит не к изменению исходной матрицы, а к созданию новой, состоящей из требуемых элементов.

## ⊳ Команда **submatrix**.

Назначение команды: выделение подматрицы.

Формат вызова 1:

submatrix(<матрица>,<диапазон\_строк>,<диапазон\_столбцов>)

Формат вызова 2:

submatrix(<матрица>,<список строк>,<список столбцов>)

Параметры:

- **<матрица>** матрица;
- -<диапазон\_строк>/<диапазон\_столбцов>-диапазоны целых чисел индексов строк/столбцов в формате a..b, где a, b— начало и конец диапазонов соответственно;

◁

— **<список\_строк>,<список\_столбцов>**— список индексов строк/столбцов для выбора.

Особенности команды submatrix:

- при указании списков строк и столбцов элемент возвращаемой подматрицы (i,j) получается как элемент исходной матрицы с индексами < список\_строк>[i], < список\_столбцов>[j], т.е. порядок указания индексов в списках имеет значение;
- можно комбинировать два указанных формата, например, указывать диапазон строк и список столбцов.

#### Пример 4.18.

> a := matrix(2, 4, [11, 12, 13, 14, 21, 22, 23, 24]); # Исходная матрица

$$a := \left[ \begin{array}{cccc} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \end{array} \right]$$

> submatrix(a, 1..2, 2..3); # Формат вызова 1

$$\begin{bmatrix} 12 & 13 \\ 22 & 23 \end{bmatrix}$$

> submatrix(a, [1,2], [2,3]); # Формат вызова 2

$$\begin{bmatrix} 12 & 13 \\ 22 & 23 \end{bmatrix}$$

> submatrix(a, [2,1], [2,3]); # Изменен порядок индексов в списке строк

$$\left[\begin{array}{cc} 22 & 23 \\ 12 & 13 \end{array}\right]$$

> submatrix(a, 1..2, [2,3]); # Комбинирование форматов вызова 1 и 2

$$\left[\begin{array}{cc} 12 & 13 \\ 22 & 23 \end{array}\right]$$

Если необходимо выделить часть строк или столбцов, то применяются команды **row** и **col**.

#### ⊳ Команда **row**.

Назначение команды: выделение строки (строк) как вектора.

Формат вызова 1:

Формат вызова 2:

#### Параметры:

- -<матрица>- матрица;
- <**строка**> номер строки;
- -<диапазон строк>-диапазон индексов строк.

#### ⊳ Команда **col**.

Назначение команды: выделение столбца (столбцов) как вектора.

 $\triangleleft$ 

◁

Формат вызова 1:

Формат вызова 2:

## Параметры:

- <матрица> матрица;
- -<**столбец>** номер столбца;
- <**диапазон столбцов**> диапазон индексов столбцов.

## Пример 4.19.

> v := row(a, 1..2); # Выделение двух строк и представление их как вектора

> type(v, 'vector'); # Действительно ли в переменной "v" вектор?

true

Обратите внимание, что при извлечении нескольких строк они записываются в виде вектора — одномерного массива. Использование команды **col** аналогично.

Расширить границы существующей матрицы можно с помощью команды **extend**.

#### ⊳ Команда **extend**.

Назначение команды: добавление строк и/или столбцов к матрице. Формат вызова:

#### Параметры:

- **<матрица>** матрица, которую следует расширить;
- **<доб\_строк>**, **<доб\_столбцов>** количество добавляемых строк и столбцов (целое число, может быть нулем);
- **<доб\_элемент>** выражение, которым заполнятся все вновь добавленные элементы (необязательный параметр).

 $\triangleleft$ 

#### Пример 4.20.

> extend(a, 1, 2); # Добавление 1 строки и 2 столбцов

$$\begin{bmatrix}
11 & 12 & 13 & 14 & ?_{1,5} & ?_{1,6} \\
21 & 22 & 23 & 24 & ?_{2,5} & ?_{2,6} \\
?_{3,1} & ?_{3,2} & ?_{3,3} & ?_{3,4} & ?_{3,5} & ?_{3,6}
\end{bmatrix}$$

> extend(a, 1, 3, 999); # Добавление 1 строки и 3 столбцов (все новые элементы заполняются значением 999)

$$\begin{bmatrix} 11 & 12 & 13 & 14 & 999 & 999 & 999 \\ 21 & 22 & 23 & 24 & 999 & 999 & 999 \\ 999 & 999 & 999 & 999 & 999 & 999 & 999 \end{bmatrix}$$

В тестовых целях, а также задачах имитационного моделирования часто возникает необходимость генерации случайных чисел и, в частности, матриц и векторов, заполненных случайными элементами (например, при моделировании цепей Маркова). Чтобы пользователю каждый раз заново не вносить вручную случайные числа, можно воспользоваться командами создания матриц и векторов, содержащих случайные элементы: **randmatrix** (от англ. **rand** *om* **wector**).

#### ⊳ Команда randmatrix.

Назначение команды: создание матрицы, состоящих из случайных элементов.

Формат вызова:

## randmatrix(<кол строк>, <кол столбцов>, <опции>)

Параметры:

- -<кол **строк**>- количество строк в создаваемой матрице;
- **<кол столбцов>** количество столбцов в создаваемой матрице;
- **<опции>** параметры генерации последовательность ключевых слов или равенств (необязательный параметр).

Особенности команды randmatrix:

- по умолчанию матрица заполняется целыми числами в диапазоне от —99 до 99;
- в опциях может быть указано одно из ключевых слов: **sparse** разреженная, **dense** плотная (сильно заполненная) (значение по умолчанию), **symmetric** симметричная, **antisymmetric** антисимметричная, **unimodular** верхнетреугольная с единичными элементами на главной диагонали;
- в опциях может быть указана фраза **entries**=<**f**>, где <**f**> функция, возвращающая случайные числа по заданному закону распределения (подробнее об этом в следующем разделе).

#### ⊳ Команда randvector.

Назначение команды: создание вектора, состоящего из случайных элементов.

Формат вызова:

$$randvector(< кол элементов>, entries = < f>)$$

Параметры:

-<кол\_столбцов>- количество элементов в создаваемом векторе;

◁

-<f>- функция, возвращающая случайные числа по требуемому закону распределения (необязательный параметр).

## Пример 4.21.

> randmatrix(2, 4);

$$\begin{bmatrix}
-85 & -55 & -37 & -35 \\
97 & 50 & 79 & 56
\end{bmatrix}$$

> randmatrix(3, 3, symmetric); # Симметричная матрица

$$\begin{bmatrix} 49 & 63 & -59 \\ 63 & 57 & 45 \\ -59 & 45 & -8 \end{bmatrix}$$

> randmatrix(2, 5, entries = rand(0..2)); # Матрица, состоящая из элементов 0, 1 и 2

$$\left[\begin{array}{ccccc} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 & 0 \end{array}\right]$$

Использование команды randvector аналогично.

**Упражнение 4.3.** Объявите матрицу A, состоящую из 3 строк и 4 столбиов, и заполните ее произвольными случайными числами ( с помощью команды **randmatrix**). Объявите вектор b, состоящий из 4 элементов, и заполните его случайными числами (с помощью команды **randvector**). Рассчитайте следующее:

- $-B = A^T A;$
- удалите в матрице A столбец и найдите обратную матрицу к оставшейся;
- добавьте к матрице А одну строку, состоящую из единиц.

Выше рассматривалось решение алгебраических уравнений и их систем с помощью команды **solve**. Как известно, системы линейных алгебраических уравнений (СЛАУ) могут быть представлены в матричной форме. Например, СЛАУ

$$\begin{cases} x_1 + 3x_2 = 5 \\ -2x_1 + 4x_2 = 0 \end{cases}$$

в матричном виде представляется как

$$Ax = b$$
,

где A — основная матрица системы:

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

b — вектор свободных членов:

$$b = \begin{bmatrix} 5 \\ 0 \end{bmatrix},$$

x — вектор неизвестных:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

В связи с тем, что в ряде задач моделирования приходится решать СЛАУ, опишем способ ее решения с помощью команды **linsolve**.

## ⊳ Команда linsolve.

Назначение команды: решение системы линейных алгебраических уравнений, представленной в матричной форме.

Формат вызова:

$$linsolve(<$$
ocн\_матрица $>$ ,  $<$ cвоб\_вектор $>$ )

Параметры:

- **<осн матрица>** основная матрица СЛАУ;
- -<**своб вектор**>- вектор свободных членов.

◁

Пример 4.22.

> a := matrix(2, 2, [1, 3, -2, 4]);  
> b := vector(2, [5, 0]);  

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

$$b := [5, 0]$$
> linsolve(a, b);

Упражнение 4.4. C помощью команд linsolve u solve peware CЛАУ <math>uсравните ответы.

$$\begin{cases} 2x_1 - 3x_2 &= -2\\ x_1 + x_2 - 4x_3 &= -5\\ -x_1 + 2x_3 &= 3 \end{cases}$$

В задачах параметрической идентификации моделей приходится решать задачи оптимизации нелинейных по параметрам функций. Если функция является непрерывной относительно своих параметров, то используются методы оптимизации, основанные на информации о локальном поведении функции за счет информации, содержащейся в градиенте. Для этих целей в Maple существуют команды вычисления градиента функции grad и вычисления матрицы Якоби jacobian.

Напомним, что градиент скалярной функции (в терминах Maple — выражения)

$$f(x_1, x_2, \ldots, x_n),$$

зависящей от n переменных  $x_1, x_2, \ldots, x_n$ , представляет собой вектор из nэлементов, составленный из частных производных функции f по переменным:

$$\nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

⊳ Команда **grad**.

Назначение команды: градиент выражения.

Формат вызова:

Параметры:

- **<выражение>** скалярное выражение;
- **<вектор>** вектор или список переменных, по которым следует вычислять градиент.

 $\triangleleft$ 

#### Пример 4.23.

> grad(5\*x1-2\*x2\*x3+exp(x1-x3)^2,[x1,x2,x3]); 
$$\left[ 5 + 2 \left( e^{(x1-x3)} \right)^2, -2 \, x3, \, -2 \, x2 - 2 \left( e^{(x1-x3)} \right)^2 \right]$$

Упражнение 4.5. Рассчитайте градиент функции

$$f(x,y) = -2xy + \sin(\cos y)x^2 + 1$$

npu значениях  $nepementum \{x=1,y=2\}$  u  $\{x=-1.2,y=0\}$ .

Для векторной функции векторного аргумента вместо градиента рассчитывается матрица Якоби (называемая часто в последнее время якобианом, хотя раньше под термином «якобиан» понимался определитель матрицы Якоби). Она представляет вектор транспонированных градиентов каждого элемента исходной функции. Говоря формально, матрица Якоби векторной функции

$$\begin{bmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ & \cdots \\ f_m(x_1, x_2, \dots, x_n) \end{bmatrix}$$

зависящей от n переменных  $x_1, x_2, \ldots, x_n$ , представляет собой матрицу размера  $m \times n$  элементов:

$$J_{x}f = \begin{bmatrix} \nabla_{x}^{T} f_{1}(x_{1}, x_{2}, \dots, x_{n}) \\ \nabla_{x}^{T} f_{2}(x_{1}, x_{2}, \dots, x_{n}) \\ \vdots \\ \nabla_{x}^{T} f_{m}(x_{1}, x_{2}, \dots, x_{n}) \end{bmatrix} = \begin{bmatrix} \frac{\partial f_{1}}{\partial x_{1}} & \frac{\partial f_{1}}{\partial x_{2}} & \dots & \frac{\partial f_{1}}{\partial x_{n}} \\ \frac{\partial f_{2}}{\partial x_{1}} & \frac{\partial f_{2}}{\partial x_{2}} & \dots & \frac{\partial f_{2}}{\partial x_{n}} \\ \vdots \\ \frac{\partial f_{m}}{\partial x_{1}} & \frac{\partial f_{m}}{\partial x_{2}} & \dots & \frac{\partial f_{m}}{\partial x_{n}} \end{bmatrix}.$$

⊳ Команда **jacobian**.

Назначение команды: матрица Якоби для вектора выражений. Формат вызова:

Параметры:

- **<вектор выражений>** вектор или список выражений;
- **<вектор>** вектор или список переменных, по которым следует вычислять матрицу Якоби.

◁

## Пример 4.24.

> jacobian([x1\*x2-2\*x3, exp(x1)+2\*x2-2\*x1\*x3], [x1,x2,x3]);

$$\begin{bmatrix}
 x2 & x1 & -2 \\
 e^{x1} - 2x3 & 2 & -2x1
\end{bmatrix}$$

**Упражнение 4.6.** Разработайте функцию, возвращающую Адамарово произведение матриц одинаковых размеров

$$C = A \circ B$$

(поэлементное умножение элементов двух матриц одинаковых размеров), т.е. вычисление элементов по следующей формуле:

$$c_{ij} = a_{ij} b_{ij}.$$

## 4.3. Пакеты LinearAlgebra и VectorCalculus

В данном подразделе довольно кратко остановимся на использовании пакетов LinearAlgebra и VectorCalculus, акцентировав внимание на наиболее важных понятиях и особенностях. Освоив пакет linalg, несложно будет перейти к использованию новых пакетов для работы с матрицами. Прежде всего, подключим пакеты LinearAlgebra и VectorCalculus с помощью команды with.

Имейте в виду, что процедуры и функции пакета LinearAlgebra (а также VectorCalculus) начинаются с заглавной буквы. Создание матрицы производится в команде Matrix, которая имеет большое количество параметров. В данном пособии мы приведем сокращенный формат вызова команды Matrix, наиболее часто применяемый в программах Maple.

#### ⊳ Команда **Matrix**.

Назначение команды: создание матрицы (Matrix). Формат вызова: Matrix(<кол\_строк>, <кол\_столбцов>, <инициализация>, <только\_чтение>, <форма>)

#### Параметры:

- **<кол\_строк>** количество строк матрицы (необязательный параметр);
- **<кол\_столбцов>** количество столбцов матрицы (необязательный параметр);
- <инициализация> функция, массив (типа array или Array), таблица (table), список списков (list), матрица (Matrix), вектор (Vector) или множество равенств вида (i, j) = <значение>, где i и j индексы элемента, содержащие начальные значения элементов матрицы (необязательный параметр); по умолчанию этот параметр имеет значение 0;
- **<только\_чтение>** указывает на доступность элементов матрицы только для чтения (необязательный параметр);
- <форма> свойства матрицы в виде shape=<список\_свойств>, указывающие на определенные характеристики матрицы (симметричность и т.д.) (необязательный параметр).

## Особенности использования команды Matrix:

- Если указан только параметр **<кол\_строк>**, а **<кол\_столбцов>** не задан, то будет создана квадратная матрица.
- Может быть задан только параметр **<инициализация>**, на основании которого создана матрица.
- Если в качестве параметра **<инициализация**> задано число, вся матрица будет заполнена данным элементом.
- Для вывода содержимого матрицы типа **Matrix** необязательно для этого применять команду **print**, можно просто указать имя.
- В параметре **<форма>** в качестве свойств могут указываться, в частности, такие ключевые слова:
  - $\circ$  antisymmetric антисимметричная;
  - **band** ленточная (ненулевые элементы идут вдоль главной диагонали);
  - **diagonal** диагональная (ненулевые элементы идут только по главной диагонали);
  - $\circ$  identity единичная;
  - $\circ$  **symmetric** симметричная;
  - triangular треугольная;
  - **zero** нулевая.

## Пример 4.25.

> Matrix(2); # Квадратная матрица порядка 2

$$\left[\begin{array}{cc} 0 & 0 \\ 0 & 0 \end{array}\right]$$

> Matrix(2, 3, [[1,2,3],[4,5,6]]); # Матрица размеров 2x3, заполненная заданными элементами

$$\left[\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array}\right]$$

> Matrix([[-1,-2,-3],[-4,-5,-6]]); # Матрица, заполненная элементами, без непосредственного указания размерностей

$$\left[\begin{array}{rrr} -1 & -2 & -3 \\ -4 & -5 & -6 \end{array}\right]$$

> Matrix(2, 2,  $\{(1,1)=-1, (2,2)=1\}$ ); # Инициализация некоторых значений с помощью множества равенств

$$\left[\begin{array}{cc} -1 & 0 \\ 0 & 1 \end{array}\right]$$

> a := Matrix(2, [[1,2],[3,4]], readonly = true); # Создание матрицы, доступной только для чтения

$$a := \left[ \begin{array}{cc} 1 & 2 \\ 3 & 4 \end{array} \right]$$

> a[1,1] := 0; # Попытка изменить элемент матрицы, доступной только для чтения

Error, cannot assign to a read-only Matrix  $\,$ 

> b := Matrix(2, shape=symmetric); # Создание квадратной симметричной матрицы

$$b := \left[ \begin{array}{cc} 0 & 0 \\ 0 & 0 \end{array} \right]$$

> b[1,2] := 1; # Вследствие симметричности изменится и элемент b[2,1]

> b; # Вывод элементов матрицы без команды print

$$b_{1,2} := 1$$

$$\left[\begin{array}{cc} 0 & 1 \\ 1 & 0 \end{array}\right]$$

> Matrix(3, shape=identity); # Единичная матрица порядка 3

$$\left[\begin{array}{ccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right]$$

Как видно, команда **Matrix** представляет собой значительно более мощный конструктор матриц, чем **matrix**. Рассмотрение процедур и функций

90

пакетов LinearAlgebra и VectorCalculus оставим на самостоятельное изучение.

Упражнение 4.7. На основе команд пакета LinearAlgebra найдите псевдообратную матрицу  $A^+$  к

$$A = \begin{bmatrix} -1 & 2 & 0 \\ 12 & -7 & -8 \\ 3 & 0 & -10 \\ 4 & 2 & 6 \end{bmatrix}.$$

по формуле

$$A^+ = (A^T A)^{-1} A^T.$$

Функции пакета LinearAlgebra нельзя применять к матрицам, определенным с помощью команды matrix пакета linalg и наоборот, т.к. объекты типов matrix и Matrix являются различными. Тем не менее, эти объекты могут быть преобразованы друг в друга с помощью команды convert.

#### ⊳ Команда **convert**.

Назначение команды: преобразование объектов из одного формата в другой. Формат вызова:

convert(<выражение>, <форма>, <аргументы>)

Параметры:

- <выражение> переменная или выражение исходного типа;
- <форма> тип или форма, к которому преобразуется исходное выражение;
- **<аргументы>** дополнительные аргументы (необязательный параметр).

Пример 4.26.

> a := Matrix(2, 3, [[7,3,-4],[2,0,-4]]); # Создание объекта типа Matrix

 $\triangleleft$ 

$$a := \left[ \begin{array}{ccc} 7 & 3 & -4 \\ 2 & 0 & -4 \end{array} \right]$$

> type(a, Matrix); # Действительно ли переменная "a" является типом "Matrix"

true

> type(a, matrix); # Действительно ли переменная "a" является типом "matrix"

false

> b := convert(a, matrix); # Теперь "b" содержит объект типа "matrix"

$$b := \left[ \begin{array}{ccc} 7 & 3 & -4 \\ 2 & 0 & -4 \end{array} \right]$$

> type(b, Matrix); # Действительно ли переменная "b" является типом "Matrix"

false

> type(b, matrix); # Действительно ли переменная "b" является типом "matrix"

true

## 4.4. Контрольные вопросы к разделу 4

- 1. Понятие подключаемого пакета в Maple. Приведите аналоги пакетов в других системах программирования. Подключение всех и части команд пакетов. Можно ли вызвать команду пакета, ранее не подключив пакет?
- 2. Векторы и матрицы в Maple. Дайте краткую характеристику пакетов, содержащих команды для работы с векторами и матрицами.
- 3. Создание векторов и матриц, доступ к их элементам. Особенности вывода на экран содержимого векторов и матриц.
- 4. Общие и специальные матричные операции. В чем специфика результатов вычислений выражений с векторами и матрицами? Операция матричного умножения.
- 5. Работа с подматрицами, строками и столбцами матриц.
- 6. Вычисление определителей матриц. Обратные матрицы. Как проверить правильность нахождения обратной матрицы?
- 7. Генерация векторов и матриц, состоящих из случайных элементов. Приведите ситуации, когда необходима или желательна генерация таких векторов и матриц.
- 8. Решение систем линейных алгебраических уравнений, представленных в матричной форме. Каким еще способом можно решать эти системы? Сравните эти способы.
- 9. Вычисление градиента и матрицы Якоби.
- 10. Сравните пакет **LinearAlgebra** с пакетом **linalg**. Объясните различия в типах **matrix** и **Matrix** и расскажите о преобразовании матриц указанных типов друг в друга.

# 5. Случайные числа. Функции распределений

# 5.1. Генерация псевдослучайных чисел

Имитационное моделирование представляют собой по сути стохастические (вероятностные) эксперименты над моделями, описанными в виде алгоритмов. Соответственно, для их проведения требуется генерация случайных чисел, распределенных по требуемым законам. Кроме того, случайные числа полезны при проведении различных тестов в разработанных программах. Основой получения случайных чисел по произвольному закону распределения являются последовательности равномерно распределенных псевдослучайных чисел. Для их генерации во всех современных языках программирования имеются необходимые процедуры. В Марlе для получения псевдослучайных равномерно распределенных чисел предназначена функция **rand**.

#### ⊳ Команда **rand**.

Назначение команды: генерация псевдослучайных чисел.

Формат вызова:

## rand(<диапазон>)

#### Параметр:

— **<диапазон>** — целое число или диапазон целых чисел (необязательный параметр).

Komanдa rand имеет следующие особенности:

— при отсутствии параметра **<диапазон>** функция возвращает целое неотрицательное 12-значное число;

 $\triangleleft$ 

- если в качестве параметра указано целое число n, функция возвращает процедуру (!), генерирующую псевдослучайные целые числа из отрезка [0; n-1];
- если в аргументе указан диапазон целых чисел a..b, функция возвращает процедуру (!), генерирующую псевдослучайные целые числа из отрезка [a;b].

Если функция **rand** возвращает процедуру, то для получения псевдослучайных чисел следует вызвать полученную процедуру без параметров. Приведем пример использования команды **rand** при разных вариантах передачи параметров.

## Пример 5.1.

- > restart: #Очищаем память
- > rand(); # Случайное 12-значное целое число

#### 427419669081

```
> r1 := rand(5); # Процедура, возвращающая случайное целое
число от 0 до 4
  r1 := \mathbf{proc}()
  local t;
  global seed;
     \_seed := irem(a * \_seed, p);
     t := seed;
     \mathbf{to}\; concats\; \mathbf{do}\; \_seed := \mathrm{irem}(a * \_seed,\; p)\; ;\; t := s * t + \_seed\; \mathbf{end}\;\; \mathbf{do}\; ;
     irem(t, divisor) + offset
  end proc
   r1(); # Получение случайного числа
                                     1
  r1(); # Еще одно случайное число
   r2 := rand(20..30); # Процедура, возвращающая случайное
целое число от 20 до 30
  r2 := \mathbf{proc}()
  local t;
  global seed;
     seed := irem(a * seed, p);
     t := seed;
     to concats do \_seed := irem(a * \_seed, p); t := s * t + \_seed end do;
     irem(t, divisor) + offset
  end proc
> r2(); # Получение случайного числа
                                    21
```

Теперь давайте рассмотрим пример, показывающий, что после команды очистки памяти **restart** значения, выдаваемые командой **rand**, будут повторяться, если не использовать другие специальные команды.

## Пример 5.2.

Такое поведение команды **rand** связано с характером генерации псевдослучайных чисел. Как известно, они генерируются с помощью детерминированной рекуррентной процедуры

$$x_{n+1} = f(x_n),$$

производящей числа, которые можно считать случайными. Для этой процедуры необходимо задание начального значения  $x_0$ . При одинаковых значениях  $x_0$  все дальнейшие числа будут одинаковыми. В Maple начальное значение  $x_0$  задается в глобальной переменной \_seed, которой присваивается положительное число (по умолчанию это делает сам Maple). Представление о влиянии значения параметра \_seed на генерацию случайных чисел показано в примере, приведенном ниже.

#### Пример 5.3.

Случайное значение переменной \_seed для получения различных случайных последовательностей может быть реализовано самостоятельно (например, на основе текущей даты и/или времени), а может быть получено с помощью вызова команды randomize.

#### ⊳ Команда **randomize**.

Назначение команды: инициализация датчика псевдослучайных чисел. Формат вызова:

## randomize(<значение>)

## Параметр:

— **<значение>** — начальное значение датчика случайных чисел — целое положительное число (необязательный параметр).

Если значение параметра команды **randomize** не указано, то инициализация датчика псевдослучайных чисел будет получено на основе текущего системного времени. Приведем пример.

## Пример 5.4.

Упражнение 5.1. Получите 10 равномерно распределенных случайных целых чисел из диапазона [100; 200]. Начальное значение датчика псевдослучайных задайте с помощью команды randomize без параметра.

На основе целых чисел могут быть получены и дробные. Для этого необходимо выбрать соответствующий диапазон целых чисел, из которого будут генерироваться числа, и затем полученные числа делить на знаменатель для получения чисел из нужного диапазона с заданной точностью. Предположим, что требуется сформировать случайное вещественное число из диапазона [2; 5] с точностью 0.01. Решение этой задачи показано в следующем примере.

## Пример 5.5.

```
> r := evalf(rand(200..500)/100):
> r(); # Получаем числа из диапазона [2;5] с точностью 0.01
2.86000000
```

**Упражнение 5.2.** Получите 5 случайных дробных чисел с 3 знаками после запятой, равномерно распределенных на отрезке [0;1].

Как было сказано в разделе, посвященном матрицам, для получения случайных матриц используется команда **randmatrix** (для векторов – команда **randvector**). Для заполнения матриц элементами, отличными от случайных целых чисел из диапазона [-99;99] (режим по умолчанию), следует в опциях указывать выражение **entries**=<**f**>. Функция, передаваемая в качестве параметра <**f**>, определяется с помощью команды **rand**. Заполнение матрицы размера  $3 \times 5$  вещественными элементами, принимающими значения из диапазона [1;10] с одним десятичным знаком, приведено ниже.

#### Пример 5.6.

```
> with(linalg): # Подключение пакета для работы с матрицами > r := rand(10..100): > evalf(evalm(randmatrix(3, 5, entries=r)/10));  \begin{bmatrix} 3.100000000 & 3.700000000 & 1.900000000 & 3.800000000 & 5.500000000 \\ 4.400000000 & 9.800000000 & 7.500000000 & 6.300000000 & 6.100000000 \\ 2.100000000 & 3.500000000 & 2.600000000 & 8.300000000 & 2.600000000 \end{bmatrix}
```

Обратите внимание, что деление вынесено отдельно. К сожалению, более логичный способ записи вызывает в Maple вывод сообщения об ошибке, что показано в следующем примере.

#### Пример 5.7.

```
> r := rand(10..100)/10:
> randmatrix(3, 5, entries=r);

Error, (in randmatrix) bad option entries = 1/10*proc () local t;
global _seed; _seed := irem(a*_seed,p); t := _seed; to concats do
_seed := irem(a*_seed,p); t := s*t+_seed end do;
irem(t,divisor)+offset end proc
```

**Упражнение 5.3.** Сгенерируйте матрицу размера  $2 \times 6$ , состоящую из вещественных элементов, равномерно распределенных на отрезке [0;1], с точностью 0.0001.

**Упражнение 5.4.** Разработайте процедуру, которая будет принимать два аргумента – порядок квадратной матрицы n и точность генерации случайных чисел ерs, которая будет возвращать переходную матрицу для моделей марковских цепей. Элементы  $p_{ij}$ ,  $i, j = 1, \ldots, n$ , такой переходной матрицы должны удовлетворять свойствам:

```
-0 ≤ p_{ij} ≤ 1, т.к. эти элементы - вероятности;
```

```
-\sum\limits_{j=1}^{n}p_{ij}=1, i=1,\ldots,n,\ m.e. сумма элементов в каждой строке должна равняться 1.
```

## 5.2. Генерация случайных величин

Во многих задачах требуется получать случайные числа в соответствии с некоторым законом распределения. Это встречается, например, при имитационном моделировании систем массового обслуживания — в них очень часто требуются случайные числа, распределенные по закону Пуассона, экспоненциальному закону или по закону Эрланга заданного порядка.

Случайные числа могут быть получены с помощью команды пакета **stats**. Заметим, что этот пакет в отличие от многих других (например, пакета **linalg**) состоит из нескольких *подпакетов* (*subpackages*). Подпакет для

моделирования случайных чисел называется **random**. Для его подключения необходимо дать команду

#### with(stats,random)

Подпакет **random** содержит единственную функцию **random**, которая и генерирует заданное количество случайных чисел, распределенных в соответствии с указанным законом распределения.

#### ⊳ Команда **random**.

Назначение команды: генерация случайных чисел в соответствии с заданным законом распределения.

Формат вызова:

# $\begin{array}{l} {\rm random[pac пределениe](<\! количество>, <\! paвномерн\_проц>,} \\ {<\! {\rm метод>})} \end{array}$

#### Параметры:

- **<распределение>** распределение, в соответствии с которым необходимо генерировать числа (см. таблицы 5.1 и 5.2);
- **<количество>** положительное целое число, указывающее, какое количество случайных чисел необходимо генерировать (по умолчанию 1) (необязательный параметр);
- **<равномерн\_проц>** процедура, генерирующая равномерно распределенные числа (по умолчанию 'default') (необязательный параметр);
- **<метод>** одно из ключевых слов **'auto'**, **'inverse'**, **'builtin'** (по умолчанию 'auto') (необязательный параметр).

◁

#### Особенности команды random:

- функция преобразует входной поток равномерно распределенных случайных чисел в выходной поток чисел, распределенных в соответствии с указанным законом распределения. Соответственно, выдаваемый результат зависит от значения глобальной переменной \_seed, поэтому выдаваемая последовательность может быть изменена с помощью команды randomize;
- процедура, указываемая в качестве параметра **<pавномерн\_проц>** должна возвращать равномерно распределенные случайные числа из диапазона [0; 1];
- параметр **<метод>** указывает на способ получения случайных чисел: значение **'inverse'** указывает, что должен использоваться метод обратной функции, а значение **'builtin'** что специально разработанная для данного распределения функция в Maple.

Таблица 5.1. Дискретные законы распределения

Ключевое слово и параметры	Распределение
binomiald[n,p]	биномиальное
discrete uniform[a,b]	дискретное
$empirical[list\_prob]$	эмпирическое
hypergeometric[N1, N2, n]	гипергеометрическое
negativebinomial[n,p]	отрицательное биномиальное
poisson[mu]	Пуассона

Таблица 5.2. Непрерывные законы распределения

Ключевое слово и параметры	Распределение
beta[nu1, nu2]	бета
cauchy[a, b]	Коши
chisquare[nu]	$\chi^2$
exponential[alpha, a]	экспоненциальное
fratio[nu1, nu2]	Фишера
gamma[a, b]	гамма (при четных —
	распределение Эрланга)
laplaced[a, b]	Лапласа
logistic[a, b]	логистическое
lognormal[mu, sigma]	логнормальное
normald[mu, sigma]	нормальное (Гаусса)
studentst[nu]	Стьюдента
uniform[a, b]	равномерное
weibull[a, b]	Вейбулла

При указании дискретных законов распределения команда **random** возвращает целые значения, при указании непрерывных — вещественные. Приведем примеры использования команды для получения последовательностей псевдослучайных чисел, распределенных по дискретному равномерному и по нормальному законам.

## Пример 5.8.

```
> random[discreteuniform[1,10]](20); 5, 4, 4, 5, 6, 8, 1, 8, 7, 8, 3, 4, 8, 1, 1, 10, 9, 5, 7, 10
```

## Пример 5.9.

- > random[normald](10);
- -0.4386378394, -1.140005385, 0.1529160443, 0.7487697029, -0.4908898750,
- $-0.6385850228,\, 0.7648245898,\, -0.04721150696,\, -1.463572278,\, 0.4470293004$

## 5.3. Работа с функциями распределений. Квантили

При построении регрессионных моделей и их применении приходится сталкиваться с задачами оценки точности и достоверности построения моделей и получаемых прогнозов. Для решения таких задач требуется знание значений квантилей соответствующих распределений. Напомним, что квантилью уровня p случайной величины X с функцией распределения F(x) называется число  $d_p$ , такое что вероятность события  $P\{X < d_p\} = p$ . Другими словами,  $d_p$  — решение уравнения

$$F(d_p) = p.$$

Если имеется обратная функция  $F^{-1}$  к функции распределения, квантиль может быть найдена по формуле

$$d_p = F^{-1}(p).$$

К сожалению, обратная функция редко может быть выражена в аналитическом виде, поэтому определение квантилей требуется производить численно. В Maple для этого применятся функция **statevalf** подпакета **statevalf** пакета **stats**.

Команда **statevalf** возвращает результаты численных вычислений со статистическими функциями (функциями, описывающими вероятностные распределения). С помощью этой команды Марlе можно работать с функциями распределений, обратными функциями распределений, плотностями распределений.

#### ⊳ Команда **statevalf**.

Назначение команды: численные вычисления с функциями и плотностями вероятностных распределений.

Формат вызова:

 $statevalf[<вид\_функции>, <pаспределение>](<аргументы>)$ 

Параметры:

- **<вид\_функции>** вид функции, которую необходимо использовать в расчетах;
- **<распределение>** вид распределения из табл. 5.1 или 5.2;
- <аргументы> параметры распределения в зависимости от его вида.

Параметр <вид функции> имеет следующее значение:

— для непрерывных распределений:

 $\triangleleft$ 

- $\circ$  **cdf** (от **c***umulative* **d***ensity* **f***unction*) функция распределения F(x);
- $\circ$  icdf (от inverse cumulative density function) обратная функция распределения  $F^{-1}(x)$ ;
- $\circ$  **pdf** (от **p**robability **d**ensity **f**unction) функция плотности f(x);
- для дискретных распределений:
  - $\circ$  dcdf (от discrete cumulative density function) дискретная функция распределения F(x);
  - $\circ$  idcdf (от inverse discrete cumulative density function) обратная дискретная функция распределения  $F^{-1}(x)$ ;
  - $\circ$  **pf** (от **p**robability **f**unction) функция вероятностей p(x).

Параметры распределений указываются после указания вида распределения в квадратных скобках. В следующем примере показано вычисление значений функций и плотностей распределений.

#### Пример 5.10.

> statevalf[pdf,normald](1.12); # Значение плотности нормированного нормального распределения в точке x=1.12

#### 0.2130691468

> statevalf[cdf,normald](1.12); # Значение функции нормированного нормального распределения в точке x=1.12

#### 0.8686431190

> statevalf[cdf,normald[2,4]](2.3); # Значение функции нормального распределения со средним 2 и стандартным отклонением 4 в точке x=2.3

#### 0.5298926441

> statevalf[cdf,exponential[1.5]](0.5); # Значение функции экспоненциального распределения с интенсивностью 1.5 в точке x=0.5

#### 0.5276334473

Покажем, как использовать команду **statevalf** для определения квантилей распределения. При решении практических задач обращайте особое внимание на то, какие вероятности необходимо передавать в качестве аргумента — умейте определять для различных распределений (например, Стьюдента, Фишера) значения аргументов в зависимости от требуемого уровня значимости или доверительной вероятности.

## Пример 5.11.

> statevalf[icdf,normald[2,4]](0.53); # Квантиль нормального распределения со средним 2 и стандартным отклонением 4 для вероятности p=0.53

#### 2.301079448

> statevalf[icdf,studentst[6]](0.95); # Квантиль распределения Стьюдента с 6 степенями свободы для вероятности p=0.95 (доверительная вероятность 2\*0.95-1=0.9)

1.943180281

> statevalf[icdf,fratio[3,4]](0.95); # Квантиль распределения  $\Phi$ ишера со степенями свободы 3 и 4 для вероятности p=0.95 (доверительная вероятность также 0.95)

6.591382116

## 5.4. Контрольные вопросы к разделу 5

- 1. Генерация псевдослучайных чисел. Инициализация датчика псевдослучайных чисел.
- 2. Применение процедур генерации псевдослучайных чисел в командах создания векторов и матриц, состоящих из случайных элементов.
- 3. Генерация случайных чисел по различным законам распределений.
- 4. Что понимается под статистическими функциями в Maple? Вычисление квантилей распределений.

## 6. Файловые операции

## 6.1. Общие сведения о работе с файлами

Пакет Maple предоставляет пользователю возможность работы с файлами — чтение и запись в них. Синтаксис основных команд для работы с файлами очень близок синтаксису функций языка программирования **C**. Команды работы с файлами поддерживаются ядром Maple — никакие пакеты для этого подключать не надо.

Перед чтением или записью данных в файл его необходимо открыть. Хотя все команды чтения и записи в Maple открывают файл автоматически, лучше делать это явно с использованием функции **fopen** (от **f***ile* **open**).

#### ▶ Команда fopen.

Назначение команды: открытие файла для буферизованного чтения или записи.

Формат вызова:

#### Параметры:

- <**имя**\_файла> имя файла (возможно указание полного пути к файлу);
- <**режим\_доступа**> режим доступа к файлу одно из ключевых слов: **READ** (для чтения файла), **WRITE** (для чтения и записи в файл с его начала), **APPEND** (для чтения и записи в конец файла);
- <вид> характер доступа к файлу одно из ключевых слов:
   ТЕХТ (текстовый файл значение по умолчанию) или BINARY (двоичный файл) (необязательный параметр).

### Особенности использования команды fopen:

— при указании пути символ обратного слэша \ необходимо дублировать: \\ (либо можно использовать одинарный прямой слэш /);

 $\triangleleft$ 

- команда возвращает *дескриптор файла* целое число;
- если в параметре <вид> указано ключевое слово TEXT, то при чтении и записи в файл символ начала новой строки  $(\n)$  будет учитываться в соответствии с правилами соответствующей операционной системы;
- при попытке открыть ранее открытый файл будет выдано сообщение об ошибке;
- если файл открывается для чтения, а его не существует, будет выдано сообщение об ошибке;
- если файл открывается для записи, а его не существует, он будет создан (при ошибке создания будет выдано сообщение об ошибке).

После выполнения всех операций над файлом его следует закрыть, используя ранее созданный дескриптор файла, с помощью команды **fclose**. При разработке приложений не забывайте, что необходимо учитывать возникновение исключений, т.к. в этом случае некоторые команды закрытия файлов Марle могут не выполниться.

#### ⊳ Команда **fclose**.

Назначение команды: закрытие файла.

Формат вызова:

## fclose(<список файлов>)

#### Параметр:

— **<список\_файлов>** — список из одного или нескольких дескрипторов или имен файлов.

Особенности использования команды fclose:

— закрывается ранее открытый файл (если файл не был открыт, то при указании имени файла никакого сообщения об ошибке выдано не будет, а при указании дескриптора — будет);

 $\triangleleft$ 

- закрытие файла гарантирует, что все изменения, сделанные в файле, будут записаны на диск;
- данной команде аналогичны команды close и pclose.

#### Пример 6.1.

> f1 := fopen("d:\\inet\\temp\\data1.txt", READ, TEXT); # Открытие текстового файла для чтения (используется двойной обратный слэш)

$$f1 := 0$$

> f2 := fopen("d:/inet/temp/data2.bin", WRITE, BINARY); # Открытие бинарного файла для записи (используется одинарный прямой слэш)

$$f2 := 1$$

> f3 := fopen("d:/inet/temp/data2.bin", WRITE, BINARY); # Попытка открыть уже открытый файл

Error, (in fopen) file already open

- > fclose(f1, f2); # Закрытие файлов, описываемых дескрипторами f1 и f2
- > f4 := fopen("d:/inet/temp/data.ttt", READ); # Попытка открыть несуществующий файл

Error, (in fopen) file or directory does not exist

Заметим, что для небуферизованного чтения/записи данных в файл используются команды **open** и **close**. Команда **open** может открывать файлы только в режиме **BINARY**. Как отмечалось выше, команды **close** и **fclose** взаимозаменяемы, т.е. могут использоваться для закрытия как буферизованных, так и небуферизованных файлов.

Для чтения и записи данных в файлы используются команды, приведенные в табл. 6.1.

Таблица 6.1. Команды для работы с файлами

Команда	Описание
filepos	Получение или установка текущей позиции в файле
fprint	Запись в файл по формату
fscanf	Чтение из файла по формату
readdata	Чтение числовых данных из текстового файла
readline	Чтение одной строки из текстового файла
writedata	Запись числовых данных в текстовый файл
writeline	Запись одной строки в текстовый файл

# 6.2. Форматированной ввод и вывод

К наиболее мощным и универсальным командам чтения и записи данных в файл относятся команды форматированного ввода и вывода данных:  $\mathbf{fscanf}$  (от  $\mathbf{file\ scan}\ by\ \mathbf{format}$ ) и  $\mathbf{fprintf}$  (от  $\mathbf{file\ print}\ by\ \mathbf{format}$ ) соответственно (обе эти функции базируются на одноименных функциях языка  $\mathbf{C}$ ).

#### ⊳ Команда **fscanf**.

Назначение команды: чтение из файла по формату. Формат вызова:

$$fscanf(<\phi$$
айл $>,<\phi$ ормат $>)$ 

Параметры:

- -<файл>- имя файла или дескриптор;
- <формат> формат (маска) чтения данных.

Команда **fscanf** возвращает список прочитанных элементов. Если успешно не был прочитан ни один элемент, то при окончании читаемых данных возвращается пустой список **NULL**. Параметр **<формат>** представляет собой строку, состоящую из элементов вида

◁

Параметры элемента строки формата:

- **<необяз>** символ \* (звездочка), указывающий, что объект должен быть прочитан, но не должен присутствовать в возвращаемом результате (необязательный параметр);
- **<ширина>** максимальное количество символов в читаемом элементе (необязательный параметр);
- **<модификатор>** модификатор типа возвращаемого значения элемента (необязательный параметр):
  - $\circ$  l, L не влияет на результат, оставлены для совместимости с языком C;
  - $\circ$  **z**, **Z** чтение комплексного числа (может предшествовать одному из кодов, соответствующих числовому формату: **d**, **o**, **x**, **e**, **f**, **g**);
- -<код>-тип элемента для чтения (наиболее важные коды приведены табл. 6.2).

Таблица 6.2. Некоторые коды в описании формата

Таблица 6.2. Некоторые коды в описании формата		
Код	Описание	
d	Десятеричное целое со знаком или без знака	
0	Восьмеричное целое без знака	
X	Шестнадцатеричное целое без знака	
$_{ m e,f,g}$	Вещественное число с плавающей точкой. Допускается	
	использование символов е или Е, указывающих	
	на степень десяти. Допускаются специальные значения	
	inf (бесконечность) и NaN (неопределенность)	
c	Символ (включая пробел, табуляцию, конец строки). При	
	указании параметра <ширина> читается указанное число	
	символов	
S	Строка до пробела, табуляции или конца строки	
[]	Символы, указанные между квадратными скобками. Если	
	вначале списка указан символ (крышка), это	
	означает чтение всех символов, кроме указанных.	
	Символ - (дефис) применяется для указания диапазонов	
a	Алгебраическое выражение до пробела, табуляции	
	и символа конца строки	
$\{\ldots\}$ wft	Прямоугольные таблицы (типы Array, Matrix или Vector).	
	Между скобками { и } указываются опции таблицы (размеры	
	и т.д.), ${\bf w}$ — ширина, ${\bf f}$ — код формата кроме $[\ldots]$ и $\{\ldots\}$ ,	
	$\mathbf{t}$ — тип возвращаемого объекта ( $\mathbf{a} - \mathbf{Array},  \mathbf{m} - \mathbf{Matrix},$	
	c - Vector[column], r - Vector[row]).	

Заметим, что формат использования команды **fscanf** совпадает с форматом команды **scanf** для разбора произвольного текста. Разбор возвращаемого результата может производиться с помощью команды **op**. Приведем пример использования команды **fscanf**.

#### Пример 6.2.

Файл d:\inet\temp\data1.txt содержит следующие строки:

Три десятичных числа: 2, -4.23, 0.45e1

Число в шестнадцатеричном формате: 2ef

Два слова: первое второе

Таблица размером 2х3

-1 3 144

2 12 -576

> f := fopen("d:\\inet\\temp\\data1.txt", READ, TEXT); #Открываем файл для чтения

$$f := 0$$

> fscanf(f, "%s%s%s%d,%f,%f"); # Чтение 1-й строки: 6 элементов

["Три", "десятичных", "числа:", 
$$2, -4.23, 4.5$$
]

> fscanf(f, "%s%s%s%s%x"); # Чтение 2-й строки: 5 элементов (2ef в шестнадцатиричной системе - это 751 в десятиричной)

["Число", "в", "шестнадцатиричном", "формате:", 751]

> fscanf(f, "%s%s%s,%s"); # Чтение 3-й строки: 6 элементов ["Два", "слова:", "первое", "второе"]

> s := fscanf(f, "%\*s%\*s%d%\*c%d"); # Чтение 4-й строки: первые два слова и символ "х" не выводим

$$s := [2, 3]$$

> m := op(1, s); n := op(2, s); # Разбираем результат чтения 4-й строки

$$m := 2$$

$$n := 3$$

> fscanf(f, "%{"||m||","||n||"}da"); # Читаем таблицу 2х3 из 5-й и 6-й строк

$$\begin{bmatrix} -1 & 3 & 144 \\ 2 & 12 & -576 \end{bmatrix}$$

> fclose(f); # Закрываем файл

Теперь рассмотрим команду для записи данных в файл.

## ⊳ Команда **fprintf**.

Назначение команды: запись в файл по формату.

Формат вызова:

## fprintf(<файл>,<формат>,<список выражений>)

Параметры:

- -<файл>- имя файла или дескриптор;
- <формат> формат (маска) записи данных;
- <**список\_выражений**> список выражений для форматирования и вывода.

Параметр **<формат>** представляет собой строку, состоящую из элементов вида

$$\% <$$
флаги $> <$ ширина $> <$ точность $> <$ модификаторы $> <$ код $>$ 

Параметры элемента строки формата:

- -<флаги>- одно или несколько значений (см. табл. 6.3) (необязательный параметр);
- **<ширина>** максимальное количество символов в записываемом элементе (необязательный параметр);
- **<точность>** количество символов для вывода дробной части вещественных чисел (необязательный параметр);
- **<модификаторы>** модификатор типа аналогично используемому в команде **fscanf** (необязательный параметр);
- -<код>-тип элемента для записи аналогично используемому. в команде **fscanf**.

Таблица 6.3. Флаги в описании формата команды **fprintf** 

Код	Описание
+	число выводится с принудительным выводом лидирующего
	знака числа (в т.ч. плюса)
-	выравнивание числа по левому краю, если по умолчанию —
	по правому, и наоборот
<пробел>	вывод числа с лидирующим знаком «-», если он имеется,
	и пробелом в противном случае
0	вывод нуля между знаком и первой цифрой, если
	происходит выравнивание по правому краю и задан
	параметр <ширина>
{}	опции для вывода прямоугольных таблиц

Команда **fprintf** возвращает количество символов, записанных в файл. При выводе с помощью команды **fprintf** автоматический переход на следующую строку не происходит. Для перевода строки необходимо указать символы \n. Аналогично команде **fprintf** в Maple есть команда **printf** для форматированного вывода на экран.

#### Пример 6.3.

```
> f := fopen("d:\\inet\\temp\\data2.txt", WRITE);  
> fprintf(f, "Матрица %s состоит из %d строк и %d столбцов\n", "A", 2, 2); # Возвращаемое значение 41 говорит, что было записано 42 символа, \n - перевод строки  
> fprintf(f, "%.2f %.2f\n%.2f %.2f", 2.129, 3.3, -2.23, 2.4503); # Будут записаны вещественные числа с двумя знаками после запятой  
> fclose(f);  
f := 0
42
```

20

Файл d:\inet\temp\data2.txt содержит следующие строки:

Матрица A состоит из 2 строк и 2 столбцов 2.13 3.30 -2.23 2.45

Упражнение 6.1. Создайте с помощью команд Maple в любом доступном каталоге файл «test.dat». В первую строку выведите свою фамилию, имя и отчество, разделяя их запятыми. Во вторую строку запишите дату своего рождения в формате «день.месяц(числом).год». Далее прочитайте записанные данные из «test.dat» и выведите на экран сообщение формата

$$<$$
число $>$   $<$ месяц (прописью) $>$   $<$ год $>$  года родился  $<$ Имя $>$   $<$ Отчество $>$   $<$ Фамилия $>$ .

## 6.3. Команды ввода и вывода строк и таблиц данных

Менее мощными, но в ряде случаев более удобными являются команды чтения и записи строки в файл: **readline** и **writeline** соответственно.

### ⊳ Команда **readline**.

Назначение команды: чтение строки из файла.

Формат вызова:

## readline(<файл>)

Параметр:

— <файл> — имя файла или дескриптор.

Особенности использования команды readline:

— команда читает строку до символа окончания строки (т.е. символа  $\setminus \mathbf{n}$ );

 $\triangleleft$ 

- символ окончания строки функция не возвращает;
- если строки для чтения отсутствуют (достигнут конец файла), то функция возвращает 0.

Приведем пример вывода на экран содержимого файла.

#### Пример 6.4.

Запись строки в файл производится с помощью команды writeline.

#### ⊳ Команда writeline.

Назначение команды: запись строки в файл.

Формат вызова:

```
writeline(<файл>,<список\_строк>)
```

Параметры:

- -<файл>- имя файла или дескриптор;
- -<**список строк**>- одна или несколько строк.

Команда **writeline** возвращает количество символов, записанных в файл. Каждая строка в списке начинается с новой строки.

 $\triangleleft$ 

## Пример 6.5.

```
> f := fopen("d:\\inet\\temp\\data3.txt", WRITE); 
> writeline(f, "Строка N1", "Строка N2"); 
> fclose(f); 
 f := 0 20
```

**Упражнение 6.2.** Создайте матрицу A размеров  $5 \times 7$  со случайными элементами. Сохраните в файле «matrix.dat» информацию о размерах матрицы, а также ее элементы. Загрузите из этого же файла матрицу B и сравните ее с исходной матрицей A.

Для чтения и записи больших объемов числовых данных можно использовать команды readdata, writedata.

#### ⊳ Команда **readdata**.

Назначение команды: чтение числовых данных из файла.

Формат вызова 1:

$$readdata(<$$
файл $>,<$ формат $>)$ 

Формат вызова 2:

Формат вызова 3:

#### Параметры:

- -<файл>- имя файла или дескриптор;
- <формат> ключевое слово integer, float или string или список этих слов, задающих формат файла;
- **<число\_колонок>** положительное целое число, указывающее количество колонок данных в файле.

Особенности команды readdata:

файл, из которого читаются данные, должен состоять из числовых данных, организованных в несколько колонок;

◁

- фактически **readdata** это применение команды **readline** и затем команды **sscanf** для разбора прочитанной строки по формату;
- при использовании формата 3 считается, что в файле находятся вещественные числа;
- если в файле одна колонка, то возвращается список значений, в противном случае список списков.

Приведем пример использования команды **readdata** для чтения таблицы данных.

### Пример 6.6.

Файл d:\inet\temp\matrix.dat содержит следующие 4 колонки данных:

```
1 2.34 4.15 3.54
2 3.74 -1.06 7.22
3 -0.45 10.93 6.34
4 7.03 -0.88 3.43
5 8.0 4.93 0.75
   > f := fopen("d:\\inet\\temp\\matrix.dat", READ);
   > readdata(f, 4); # Чтение 4 колонок как вещественных чисел
   > fclose(f);
                                  f := 0
     [[1.,\, 2.34,\, 4.15,\, 3.54],\, [2.,\, 3.74,\, -1.06,\, 7.22],\, [3.,\, -0.45,\, 10.93,\, 6.34],
     [4., 7.03, -0.88, 3.43], [5., 8.0, 4.93, 0.75]]
   > f := fopen("d:\\inet\\temp\\matrix.dat", READ); readdata(f,
   [integer, float, float,
   > float]); # Чтение 1-й колонки как целых, а остальных - как
   вещественных чисел
      fclose(f);
                                  f := 0
      [[1, 2.34, 4.15, 3.54], [2, 3.74, -1.06, 7.22], [3, -0.45, 10.93, 6.34],
      [4, 7.03, -0.88, 3.43], [5, 8.0, 4.93, 0.75]]
   > f := fopen("d:\\inet\\temp\\matrix.dat", READ);
   > readdata(f, integer); # Чтение только 1-й колонки как целых
   > fclose(f);
                                  f := 0
                               [1, 2, 3, 4, 5]
```

Komanda writedata позволяет записывать большие объемы числовых данных в файл.

## ⊳ Команда writedata.

Назначение команды: запись числовых данных в файл. Формат вызова:

writedata(<файл>,<данные>,<формат>,<проц\_нечисл>)

Параметры:

- -<файл>- имя файла или дескриптор;
- <**данные**> список, вектор, матрица или список списков значений;
- **<формат>** формат данных аналогично команде **readdata** (необязательный параметр);
- **<проц\_нечисл>** процедура, указывающая, как записывать нечисловые данные (необязательный параметр).

Особенности команды readdata:

- обыкновенные дроби перед записью преобразуются к формату чисел с плавающей запятой;
- параметр **<проц\_нечисл>** обычно используется для записи комплексных чисел.

Приведем пример использования команды writedata для сохранения содержимого матрицы в файл.

#### Пример 6.7.

- > with(linalg):
- > a := randmatrix(3, 5); # Создаем матрицу 3x5 из случайных элементов

$$a := \begin{bmatrix} -85 & -55 & -37 & -35 & 97 \\ 50 & 79 & 56 & 49 & 63 \\ 57 & -59 & 45 & -8 & -93 \end{bmatrix}$$

- > f := fopen("d:\\inet\\temp\\data4.txt", WRITE);
- > writedata(f, a);
- > fclose(f);

$$f := 0$$

Файл d:\inet\temp\data4.txt содержит следующие строки:

-85 -55 -37 -35 97 50 79 56 49 63 57 -59 45 -8 -93

Упражнение 6.3. Создайте матрицу размера  $5 \times 4$ , состоящую из случайных чисел из отрезка [-10;10] с одним знаком после запятой. Сохраните ее в файле с помощью команды writedata. Загрузите эту матрицу из файла с помощью команды readdata.

При обработке статистической информации, в частности, при построении моделей регрессии, удобной может оказаться специальная команда уже рассматривавшегося ранее пакета **stats**. Такая команда — **importdata** — содержится в одноименном подпакете **importdata**.

## ⊳ Команда **importdata**.

Назначение команды: чтение статистических данных из файла. Формат вызова:

importdata(<имя\_файла>, <число\_потоков>)

#### Параметры:

- **<имя** файла> имя файла (не дескриптор!);
- <**число\_потоков**> количество потоков, на которые необходимо разбить данные по умолчанию 1 (необязательный параметр).

Особенности команды **importdata**:

- файл, указанный в данной команде, должен содержать последовательность чисел, разделенных друг от друга пробелами или символами перевода строки;
- если в строке встретился символ # (решетка), то оставшаяся часть строки считается комментарием;
- если какое-то значение отсутствует, то вместо него должен стоять символ \* (звездочка);
- если количество потоков равно 1, то команда возвращает весь считанный набор данных как последовательность выражений. В противном случае результатом будет являться последовательность списков, в которые последовательно будут заноситься значения (первое значение из файла в первый список, второе во второй и т.д. циклически).

#### Пример 6.8.

Файл d:\inet\temp\datastat.txt содержит следующие строки:

```
# Начало файла - комментарий 1.04 -4.56 4.38 -2.10 9.83 5.34 3.34*0.97 # В этой строке имеется пропущенное значение # Конец файла - комментарий > with(stats, importdata): # Подключаем подпакет importdata пакета stats > T:= importdata("d:\\inet\\temp\\datastat.txt", 3); T:=[1.04, -2.10, 3.34], [-4.56, 9.83, missing], [4.38, 5.34, 0.97]
```

Команда **importdata** предназначена лишь для упрощения операции чтения статистических данных из файлов. В более сложных случаях вместо данной команды могут быть использованы рассмотренные выше команды для чтения данных из файлов, например, **fscanf**.

## 6.4. Контрольные вопросы к разделу 6

- 1. Работа с файлами в Maple, их открытие и закрытие. Сравните технологию работы с файлами в Maple и в других языках программирования.
- 2. Форматированное и построчное чтение данных из файла.
- 3. Форматированная и построчная запись данных в файл.
- 4. Чтение и запись таблиц числовых данных в файл.
- 5. Загрузка статистических данных из файла, ее особенности.

#### Заключение

Изложенный материал в данном пособии является лишь базовой информацией, необходимой для начала работы с математическим пакетом Maple. Приведенный материал позволит студентам быстро освоить базовые идеи и приемы работы в Maple, позволит решить многие задачи и лабораторные работы, встречающиеся в курсе моделирования. Конечно, для более эффективной работы следует более основательно изучить справочную информацию, поставляемую с пакетом Maple, а также примеры решения многих задач, приведенные на сайте, посвященном данном пакету.

За пределами данного пособия остались многие возможности математического пакета Maple: функциональность различных подключаемых пакетов (для работы с графами и сетями и т.д.), взаимодействие с другими приложениями (например, с пакетом Matlab), вызов функций, написанных на других языках программирования (С), разработка программ Maple с графическим интерфейсом (маплеты) и многое другое. Будем надеяться, что мощь Мaple не останется незамеченной и пригодится вам как в учебных и исследовательских целях, так и в дальнейшей производственной деятельности.

# Предметный указатель

Векторы	col82
Объявление	convert91
Создание со случайными элемен-	det79
тами84	diff40
Выражения	error63
Упрощение	eval28
Высказывание	evalb
Вычисление значений 28, 35	evalf
	evalm
Генерация	extend
Псевдослучайных чисел93	fclose
Случайных чисел в соответствии с	fopen
законом распределения97	for
Градиент86	fprintf
Группа вычислений7	fscanf
Добавление перед курсором8	gc66
Добавление после курсора8	grad
The hands are a server and a server a server and a server a server and a server a server and a server a server and a serve	if
Дифференцирование40	importdata
Замена подвыражений	int
алгебраическая30	inverse
синтаксическая29	
	jacobian
Инициализация датчика псевдослу-	linsolve
чайных чисел	Matrix
Интегрирование41	matrix
Квантиль100	maximize
Команда	minimize
'if'55	nops
->	op
:=	packages
?14	print51
about	proc 60
	rand
additionally	randmatrix83
algsubs	random
anames	randomize95
array52	randvector
assigned21	readdata111
assume23	readline
coeff	

return67	Обращение 79
row	Объявление
seq46	Определитель 79
simplify	Транспонирование 77
solve33	Матрицы и векторы
statevalf	Команды для работы 77
submatrix80	Обращение к элементам74
subs 30	Операции76
table50	Умножение78
time	Множество49
transpose	Hanapayampa 10
try67	Неравенство
type 42	Область ввода
unapply37	Область вывода7
unassign	Оператор
value	Условный54
vector76	Сокращенная форма55
whattype 44	присваивания
with 70	цикла57
writedata	управление 59
writeline	Поможе 70
Команды	Пакеты70
Выполнение	linalg
Завершение11	LinearAlgebra88
Переход на следующую строку 12	stats
Правила набора10	VectorCalculus88
Константы	Подключение70
предопределенные16	Палитры
строковые	Переменная пакета Digits 17, 18
Коэффициент при выражении42	Переменные
Массивы	Последовательность выражений. 44
Математические функции26	Создание
Матрица Якоби (якобиан)87	Предположения относительно пере-
Матрицы	менных
Выделение подматрицы80	32
Выделение столбцов 82	Преобразование
Выделение строк	выражения в функцию37
Генерация со случайными элемен-	матриц различных типов 91
тами	форматов91
96	функции в выражение
Добавление строк и столбцов83	чисел17
, , , , , , , , , , , , , , , , , , ,	1110001

Процедуры
Создание60
Рабочий лист       6         Режимы ввода       8         команд Марle в виде математической символики       9         команд Марle в стандартном режиме       9         текстовой информации       8         текстовой математической символики       9
Секция
Решение       85         Сообщение об ошибке       63         Список       47         Справочная система       13         Ссылка на предыдущие результаты       12
Таблицы       50         Точность отображения результатов       18
Уравнение       32         Решение       33, 35
Файлы       3акрытие       104         Запись       107         больших объемов числовых данных       112         строки       110         Открытие       103         Чтение       105         больших объемов числовых данных       111         статистических данных       113         строки       109
Функции работы с файдами 105
раооты с сраидами — — ПО

создание60
Функция
Оптимизация38
Функция (функциональный опера-
тор)
Числа16

## Библиографический список

- 1. Говорухин В., Цибулин Б. Компьютер в математическом исследовании.— СПб: Питер, 2001.-624c.
- 2. Матросов А.В. Мар<br/>le 6. Решение задач высшей математики и механики.— СПб.: БХВ, 2001.— 528<br/>с.
- 3. Прохоров Г.В., Колбеев В.В., Желнов К.И., Леденев М.А. Математический пакет Maple V Release 4: Руководство пользователя.— Калуга: Облиздат, 1998.-200c.
- 4. http://www.maplesoft.com

## УЧЕБНОЕ ПОСОБИЕ

## Основы использования математического пакета MAPLE в моделировании

ДЛЯ СТУДЕНТОВ СПЕЦИАЛЬНОСТИ 230101 «Вычислительные машины, комплексы, системы и сети»

САРАЕВ Павел Викторович

Редактор: Т.И. Сячинова

Компьютерный набор: П.В. Сараев

Подписано в печать 28.08.06 Формат  $60 \times 84 / 16$ . Бумага для множительных аппаратов Усл. печ. л.  $7{,}44$ . Тираж 500 экз. Зак. №  $213/\Pi\Phi$