Е.М. Артемова, И.А. Бизяев

СИСТЕМА КОМПЬЮТЕРНОЙ МАТЕМАТИКИ MAPLE: ТЕОРИЯ И РЕШЕНИЕ ЗАДАЧ

Часть 2

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Удмуртский государственный университет»
Институт математики, информационных технологий и физики
Кафедра теоретической и экспериментальной физики

Е.М. Артемова, И.А. Бизяев

СИСТЕМА КОМПЬЮТЕРНОЙ МАТЕМАТИКИ MAPLE: ТЕОРИЯ И РЕШЕНИЕ ЗАДАЧ

Часть 2

Учебно-методическое пособие

Ижевск
2023

УДК 519.6:004(075.5)
ББК 22.195.3я73
А861

Рекомендовано к изданию Учебно-методическим советом УдГУ

Пособие подготовлено в Уральском математическом центре (соглашение № 075-02-2023-933).

Рецензент: канд. физ.-мат. наук, ст. науч. сотрудник ИМ УдмФИЦ УрО РАН М.Р. Королева

Артемова Е.М., Бизяев И.А.

А861 Система компьютерной математики Maple: теория и решение задач : учеб.-метод. пособие в 2 ч. : Ч. 2. – Ижевск : Удмуртский университет, 2023. – 91 с.

Пособие представляет собой практическое руководство по изучению возможностей пакета аналитических вычислений Maple. Подробные теоретические сведения чередуются с практическими заданиями. Последовательное изучение тем и выполнение заданий позволит постепенно освоить основные команды математической системы Maple.

Пособие предназначено для студентов, аспирантов, преподавателей физико-математических направлений университетов, а также для широкого круга исследователей при первоначальном ознакомлении с пакетом. Кроме того, может использоваться как справочник по основным командам Maple.

УДК 519.6:004(075.5)
ББК 22.195.3я73

Содержание

Предисловие

Содержание второй части данного учебно-методического пособия составляют разделы, изучаемые студентами Удмуртского государственного университета (УдГУ) при освоении дисциплины «Аналитические методы вычислений». Приведенные в данной части методического пособия материалы могут быть полезны при изучении дисциплин, требующих дополнительных компьютерных вычислений, например, «Уравнения тепломассопереноса», «Гидродинамика», «Нелинейные уравнения математической физики», «Вычислительная математика в физических задачах». При написании преследовалась цель описать возможности пакета Maple, по возможности в наиболее простой и ясной форме.

Материал второй части данного пособия разделен на четыре основные части. Первая (пятая) посвящена работе с командами линейной алгебры в Maple, описаны часто используемые пакеты линейной алгебры и рассмотрены примеры их использования. Во второй (шестой) изложены основы программирования в Maple, а также описаны правила создания пользовательских функций и процедур. В этой части приведены примеры использования циклов и условного оператора при решении задач математической физики. Третий (седьмой) раздел посвящен решению дифференциальных уравнений, описан синтаксис команд для построения точного, приближенного и численного решения дифференциальных уравнений (как обыкновенных, так и в частных производных). В четвертой (восьмой) части приведены разобранные примеры решения известных механических задач с использованием графических возможностей пакета Maple. Кроме краткого изложения теоретического материала, в данном пособии представлены примеры решения типовых задач по каждому разделу, а также перечень задач для самостоятельного решения.

Пособие предназначено для студентов бакалавриата, магистратуры, аспирантов, преподавателей физико-математических направлений университетов, а также для неспециалистов

в работе с пакетом Maple для первоначального ознакомления. Отдельные разделы пособия будут полезны научным сотрудникам в области математики, химии, биохимии и т. д.

Для изучения некоторых разделов данного пособия читателю будет полезно ознакомиться с содержанием ранее изданных учебно-методических пособий [1, 2], в которых изложены основы синтаксиса других языков программирования. А также с пособиями [3, 4, 5], в которых приведены фрагменты Maple-кода для решения задач математической физики и анализа динамических систем.

Необходимость издания данного пособия обусловлена отсутствием современных методических пособий по данной тематике в библиотечном фонде УдГУ. В основу пособия положены лекции, читаемые авторами в Институте математики, информационных технологий и физики в УдГУ для студентов направлений «Физика» и «Прикладные математика и физика». Содержание данного пособия соответствует учебной программе дисциплины «Аналитические методы вычислений», связанной с изучением пакета Maple и его использованием для решения задач.

В пособии учтены требования Федерального государственного образовательного стандарта к содержанию курса для направлений подготовки бакалавров «Физика» и «Прикладные математика и физика». Методическое пособие рассчитано на средний уровень физико-математической подготовки студентов и необходимо для освоения следующих компетенций:

– способность использовать в профессиональной деятельности базовые знания фундаментальных разделов математики, создавать математические модели типовых профессиональных задач и интерпретировать полученные результаты с учетом границ применимости моделей;

– способность использовать базовые теоретические знания для решения профессиональных задач;

– способность применять на практике базовые общепрофессиональные знания и умения теории и методов физических исследований (в соответствии с профилем подготовки).

5. Команды линейной алгебры

В данном разделе в основном (за исключением пункта 5.3) рассмотрены функции, расположенные в пакете `LinearAlgebra` (если не оговорено другое). Также для справки будут приводиться команды линейной алгебры из пакета `linalg`. Поэтому прежде, чем использовать ниже приведенные команды, следует подключить указанные пакеты:

```
> with(LinearAlgebra):
> with(linalg):
```

Заметим, что пакет `linalg` считается устаревшим и функции из этого пакета имеют меньше опций, по сравнению с функциями из пакета `LinearAlgebra`. Часто функции из пакетов `linalg` и `LinearAlgebra` возвращают разные структуры данных, из-за чего с результатом функций пакета `linalg` могут работать только функции из этого же пакета, аналогично для пакета `LinearAlgebra`. Поэтому для решения задач линейной алгебры желательно выбирать только один из рассматриваемых пакетов.

Для понимания, какая функция в каком пакете располагается, отметим, что команды/функции из пакета `linalg` начинаются со строчной буквы, а из пакета `LinearAlgebra` — с заглавной.

5.1. Работа с векторами

Для создания вектора используется команда/функция `Vector` (базовая команда, не требующая подключения дополнительных пакетов) со следующим синтаксисом:

```
> Vector([v1, v2, ..., vN]);
```

или

```
> Vector(N, opts);
```

Здесь `[v1, v2, ..., vN]` — список компонент N-мерного вектора, `N` — размер создаваемого вектора, `opts` — дополнительные опции, которые читателю предлагается изучить самостоятельно. Если при создании вектора указан только его размер, то по умолчанию он будет заполнен нулями.

Для обращения к компонентам (координатам) вектора используются квадратные скобки `[]`, внутри которых указывается номер координаты. Следует помнить, что нумерация компонент вектора в Maple начинается с **единицы**.

Пример создания векторов $a = (-1,0,5)^T$ и $b = (5, 10, 5)^T$:

```
> a := Vector([-1, 0, 5]);
```

$$a := \begin{bmatrix} -1 \\ 0 \\ 5 \end{bmatrix}$$

```
> b := Vector(3);
```

$$b := \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

```
> b[1] := 5: b[2] := 10: b[3] := 5:
> b;
```

$$b := \begin{bmatrix} 5 \\ 10 \\ 5 \end{bmatrix}$$

Часто при работе с векторами необходимо, чтобы они были заполнены не заданными числовыми значениями, а некоторыми символами. Для этого используется дополнительная опция `symbol = x`, где `x` — символ, который будет использоваться в качестве обозначения компонент вектора. При этом необходимо убедиться, что ранее переменная `x` нигде не была объявлена.

Пример создания векторов X и Y, заполненных символьно:

```
> X := Vector(2, symbol = x);
```

$$X := \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

```
> Y := Vector(3, symbol = y);
```

$$Y := \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Заметим, что имя переменной-вектора не должно совпадать с именем, указанным в опции `symbol`, поскольку в этом случае возникает следующая ошибка:

```
> X := Vector(2, symbol = X);
Error, recursive assignment
```

Далее рассмотрим возможные операции над векторами.

1. Для сложения, вычитания векторов, а также умножения вектора на число используются стандартные арифметические операции, описанные в разделе 2.2 первой части пособия.

Примеры сложения векторов $a = (5, 2)^T$, $b = (-8, 12)^T$, умножения вектора b на 20 и вычисления выражения $15a - b$:

```
> a := Vector([5, 2]);
```

$$a := \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

```
> b := Vector([-8, 12]);
```

$$b := \begin{bmatrix} -8 \\ 12 \end{bmatrix}$$

```
> a + b;
```

$$\begin{bmatrix} -3 \\ 14 \end{bmatrix}$$

```
> 20 * b;
```

$$\begin{bmatrix} -160 \\ 240 \end{bmatrix}$$

```
> 15 * a - b;
```

$$\begin{bmatrix} 83 \\ 18 \end{bmatrix}$$

2.1. Для вычисления скалярного произведения может использоваться функция `DotProduct` из пакета `LinearAlgebra` со следующим синтаксисом:

```
> DotProduct(V1, V2, conjugate = s);
```

Здесь `V1, V2` — векторы, скалярное произведение которых вычисляется, `conjugate = s` — опция комплексного сопряжения вектора `V1`, параметр `s` может иметь значение `true` (по умолчанию) или `false`. По умолчанию считается, что векторы `V1, V2` комплексные, и по определению при перемножении один из векторов должен быть комплексно-сопряжен. В случае, когда мы считаем векторы `V1, V2` действительными, в качестве опции комплексного сопряжения необходимо указывать `conjugate = false`.

Пример вычисления скалярного произведения векторов $a = (1, 2, 3)^T$ и $b = (-1, 0, 1)^T$:

```
> a := Vector([1, 2, 3]);
```

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

```
> b := Vector([-1, 0, 1]);
```

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

```
> with(LinearAlgebra):
> DotProduct(a, b);
```

$$2$$

Пример вычисления скалярного произведения векторов v и u, заполненных символьно:

```
> v := Vector(4, symbol = V):
> u := Vector(4, symbol = U):
> with(LinearAlgebra):
> DotProduct(v, u);
```

$$\overline{V}_1 U_1 + \overline{V}_2 U_2 + \overline{V}_3 U_3 + \overline{V}_4 U_4$$

```
> DotProduct(v, u, conjugate = false);
```

$$V_1 U_1 + V_2 U_2 + V_3 U_3 + V_4 U_4$$

2.2. Аналогичная команда/функция для вычисления скалярного произведения из пакета `linalg` называется `dotprod`. Эта команда имеет следующий синтаксис:

```
> dotprod(V1, V2);
```

Здесь `V1, V2` — векторы, скалярное произведение которых вычисляется. Заметим, что данная команда не имеет дополнительных опций для отключения комплексного-сопряжения.

Пример вычисления скалярного произведения векторов $x = (1+i, 2i)^T$,$y = (-1, 3+i)^T$:

```
> x := Vector([1+I, 2*I]);
```

$$x := \begin{bmatrix} 1+\mathrm{I} \\ 2\mathrm{I} \end{bmatrix}$$

```
> y := Vector([-1, 3+I]);
```

$$y := \begin{bmatrix} -1 \\ 3+\mathrm{I} \end{bmatrix}$$

```
> with(linalg):
> dotprod(x, y);
```

$$1 + 5\mathrm{I}$$

2.3. Для вычисления внутреннего произведения векторов (скалярное произведение без комплексного сопряжения) используется функция `innerprod` из пакета `linalg`, синтаксис которой следующий:

```
> innerprod(V1, V2);
```

Здесь `V1, V2` — векторы, внутреннее произведение которых вычисляется.

Пример вычисления внутреннего произведения векторов $x = (1+i, 2i)^T$,$y = (-1, 3+i)^T$:

```
> x := Vector([1+I, 2*I]):
> y := Vector([-1, 3+I]):
> with(linalg):
> innerprod(x, y);
```

$$-3 + 5\mathrm{I}$$

Таким образом, для вычисления скалярного произведения могут использоваться команды: `DotProduct`, `dotprod`, `innerprod`. Причем, если у команды `DotProduct` не указана дополнительная опция или указана опция `conjugate = true`, то она работает почти как функция `dotprod`, если у команды `DotProduct` указана опция `conjugate = false`, то она работает как команда `innerprod`.

3. Для вычисления векторного произведения векторов может использоваться либо команда/функция `CrossProduct` из пакета `LinearAlgebra`, либо команда `crossprod` из пакета `linalg`, синтаксис которых совпадает:

```
> CrossProduct(V1, V2);
> crossprod(V1, V2);
```

Здесь `V1, V2` — векторы, векторное произведение которых вычисляется. Следует помнить, что результатом функции векторного произведения является вектор и эта функция определена только для **трехмерных векторов**.

Пример вычисления векторного произведения для векторов K и L, которые заполнены символьными значениями:

```
> K := Vector(3, symbol = k);
```

$$K := \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix}$$

```
> L := Vector(3, symbol = l);
```

$$L := \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix}$$

```
> with(LinearAlgebra): with(linalg):
> CrossProduct(K, L);
```

$$\begin{bmatrix} k_2 l_3 - k_3 l_2 \\ -k_1 l_3 + k_3 l_1 \\ k_1 l_2 - k_2 l_1 \end{bmatrix}$$

```
> crossprod(K, L);
```

$$\begin{bmatrix} k_2 l_3 - k_3 l_2 & -k_1 l_3 + k_3 l_1 & k_1 l_2 - k_2 l_1 \end{bmatrix}$$

4. Для вычисления нормы вектора может использоваться либо команда `Norm` из пакета `LinearAlgebra`, либо команда `norm` из пакета `linalg`, синтаксис которых совпадает:

```
> Norm(V, N);
> norm(V, N);
```

Здесь `V` — вектор, `N` — номер вычисляемой нормы. Напомним, что длина вектора соответствует второй норме, то есть `N = 2`.

Пример вычисления длины вектора $z = (1, 2, 3, 4, 5)$ и его максимальной (бесконечной) нормы:

```
> z := Vector([1, 2, 3, 4, 5]);
```

$$z := \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

```
> with(LinearAlgebra): with(linalg):
> Norm(z, 2); norm(z, 2);
```

$$\sqrt{55}$$
$$\sqrt{55}$$

```
> Norm(z, infinity); norm(z, infinity);
```

$$5$$
$$5$$

5.2. Работа с матрицами

Для создания матрицы используется команда `Matrix`, которая является базовой, то есть не требует подключения сторонних пакетов, и имеет следующий синтаксис:

```
> Matrix([[m11, ..., m1N], ..., [mM1, ..., mMN]]);
```

или

```
> Matrix(M, N, opts);
```

Здесь `[[m11, ..., m1N], ..., [mM1, ..., mMN]]` — компоненты матрицы, где компоненты каждой строки описываются в отдельных квадратных скобках, `M` — количество строк матрицы, `N` — количество столбцов матрицы, `opts` — дополнительные опции. Если при создании матрицы указаны только ее размеры, то по умолчанию она будет заполнена нулями.

Для обращения к компонентам матрицы так же, как и для вектора, используются квадратные скобки `[]`, внутри которых через запятую указываются номера столбца и строки. Нумерация строк и столбцов в Maple начинается с единицы.

Пример создания матриц $A = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$:

```
> A := Matrix([[0, -1], [-1, 0]]);
```

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

```
> B := Matrix(2, 2):
> B[1, 1] := 1: B[2, 2] := 1:
> B;
```

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

Для автоматического заполнения матрицы символами используется опция `symbol = x`, где `x` — символ, который будет использоваться в качестве обозначения компонент матрицы. При этом необходимо убедиться, что ранее переменная `x` нигде не была объявлена.

Пример создания матриц C и K, заполненных символьно:

```
> C := Matrix(3, 3, symbol = c);
```

$$C := \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix}$$

```
> K := Matrix(2, 4, symbol = k);
```

$$K := \begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} & k_{1,4} \\ k_{2,1} & k_{2,2} & k_{2,3} & k_{2,4} \end{bmatrix}$$

Рассмотрим возможные операции над матрицами:

1. Для сложения, вычитания, умножения на число матриц используются стандартные арифметические операции, описанные в разделе 2.2 первой части пособия. Следует помнить, что операции сложения и вычитания матриц определены только для матриц одинаковых размеров.

Пример вычисления следующих выражений $A + B$, $-15A$, $C = A - 10B$, где

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} -1 & 3 \\ 0 & 5 \end{pmatrix}:$$

```
> A := Matrix([[1, 2], [3, 4]]):
> B := Matrix([[-1, 3], [0, 5]]):
> A + B;
```

$$\begin{bmatrix} 0 & 5 \\ 3 & 9 \end{bmatrix}$$

```
> -15 * A;
```

$$\begin{bmatrix} -15 & -30 \\ -45 & -60 \end{bmatrix}$$

```
> A - 10 * B;
```

$$\begin{bmatrix} 11 & -28 \\ 3 & -46 \end{bmatrix}$$

2. Для перемножения матриц и умножения матрицы на вектор могут использоваться следующие функции/команды:

1) `Multiply` из пакета `LinearAlgebra`, синтаксис этой команды следующий:

   `> Multiply(A, B);`
   Здесь `A, B` — перемножаемые матрицы или матрица и вектор.

2) `multiply` из пакета `linalg`, синтаксис этой команды следующий:

   `> multiply(A, B);`
   Здесь `A, B` — перемножаемые матрицы или матрица и вектор.

3) Базовая команда `evalm` со следующим синтаксисом:

   `> evalm(A &* B);`
   Здесь `A, B` — перемножаемые матрицы или матрица и вектор.

Пример перемножения символьных матриц M и N:

```
> M := Matrix(2, 3, symbol = m);
```

$$M := \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \end{bmatrix}$$

```
> N := Matrix(3, 2, symbol = n);
```

$$N := \begin{bmatrix} n_{1,1} & n_{1,2} \\ n_{2,1} & n_{2,2} \\ n_{3,1} & n_{3,2} \end{bmatrix}$$

```
> with(LinearAlgebra): with(linalg):
> Multiply(M, N);
```

$$\begin{bmatrix} m_{1,1}n_{1,1} + m_{1,2}n_{2,1} + m_{1,3}n_{3,1} & m_{1,1}n_{1,2} + m_{1,2}n_{2,2} + m_{1,3}n_{3,2} \\ m_{2,1}n_{1,1} + m_{2,2}n_{2,1} + m_{2,3}n_{3,1} & m_{2,1}n_{1,2} + m_{2,2}n_{2,2} + m_{2,3}n_{3,2} \end{bmatrix}$$

```
> multiply(M, N);
```

$$\begin{bmatrix} m_{1,1}n_{1,1} + m_{1,2}n_{2,1} + m_{1,3}n_{3,1} & m_{1,1}n_{1,2} + m_{1,2}n_{2,2} + m_{1,3}n_{3,2} \\ m_{2,1}n_{1,1} + m_{2,2}n_{2,1} + m_{2,3}n_{3,1} & m_{2,1}n_{1,2} + m_{2,2}n_{2,2} + m_{2,3}n_{3,2} \end{bmatrix}$$

```
> evalm(M &* N);
```

$$\begin{bmatrix} m_{1,1}n_{1,1} + m_{1,2}n_{2,1} + m_{1,3}n_{3,1} & m_{1,1}n_{1,2} + m_{1,2}n_{2,2} + m_{1,3}n_{3,2} \\ m_{2,1}n_{1,1} + m_{2,2}n_{2,1} + m_{2,3}n_{3,1} & m_{2,1}n_{1,2} + m_{2,2}n_{2,2} + m_{2,3}n_{3,2} \end{bmatrix}$$

При перемножении матриц необходимо следить за размерами перемножаемых матриц, в противном случае Maple выведет сообщение об ошибке.

Пример умножения матрицы $A = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}$ на вектор $b = \begin{pmatrix} 5 \\ 1 \end{pmatrix}$:

```
> A := Matrix([[1, 1], [0, 2]]):
> b := Vector([5, 1]):
> with(LinearAlgebra): with(linalg):
> Multiply(A, b);
```

$$\begin{bmatrix} 6 \\ 2 \end{bmatrix}$$

```
> multiply(A, b);
```

$$\begin{bmatrix} 6 & 2 \end{bmatrix}$$

```
> evalm(A &* b);
```

$$\begin{bmatrix} 6 & 2 \end{bmatrix}$$

3. Для транспонирования матрицы можно использовать либо команду `Transpose` из пакета `LinearAlgebra`, либо команду `transpose` из пакета `linalg`. Синтаксис этих команд следующий:

```
> Transpose(A);
> transpose(A);
```

Здесь `A` — транспонируемая матрица. Также в пакете `LinearAlgebra` существует отдельная команда `HermitianTranspose` для эрмитова транспонирования матрицы,

то есть транспонирования с сопряжением, синтаксис этой команды аналогичен синтаксису выше указанных команд:

```
> HermitianTranspose(A);
```

Здесь `A` — транспонируемая матрица.

Пример вычисления A^T и B^T для матриц

$$A = \begin{pmatrix} 1 & 3 & 7 \\ 0 & 2 & 8 \\ 0 & 0 & 10 \end{pmatrix}, \quad B = \begin{pmatrix} 2i & 1 \\ 1+i & -3i \end{pmatrix}$$

```
> A := Matrix([[1, 3, 7], [0, 2, 8], [0, 0, 10]]):
> B := Matrix([[2*I, 1], [1 + I, -3*I]]):
> with(LinearAlgebra): with(linalg):
> Transpose(A);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 3 & 2 & 0 \\ 7 & 8 & 10 \end{bmatrix}$$

```
> transpose(B);
```

$$\begin{bmatrix} 2\mathrm{I} & 1+\mathrm{I} \\ 1 & -3\mathrm{I} \end{bmatrix}$$

```
> HermitianTranspose(B);
```

$$\begin{bmatrix} -2\mathrm{I} & 1-\mathrm{I} \\ 1 & 3\mathrm{I} \end{bmatrix}$$

4. Определитель матрицы можно найти с помощью функции `Determinant` из пакета `LinearAlgebra` или функции `det` из пакета `linalg`, синтаксис которых следующий:

```
> Determinant(A, method = m);
> det(A);
```

Здесь `A` — матрица, определитель которой вычисляется, `method = m` — дополнительная опция, определяющая алгоритм вычисления определителя матрицы, с которой читателю предлагается ознакомиться самостоятельно.

Пример вычисления определителя матрицы $A = \begin{pmatrix} 4 & 10 \\ 5 & -5 \end{pmatrix}$:

```
> A := Matrix([[4, 10], [5, -5]]):
> with(LinearAlgebra): with(linalg):
> Determinant(A);
```

$$-70$$

```
> det(A);
```

$$-70$$

Матрицы могут содержать не только числовые значения, но и символьные выражения, например формулы. Пример вычисления определителя матрицы $B = \begin{pmatrix} \cos\beta & \cos\alpha \\ \sin\beta & \sin\alpha \end{pmatrix}$:

```
> B := Matrix([[cos(beta), cos(alpha)],
               [sin(beta), sin(alpha)]]):
> Determinant(B);
```

$$\cos(\beta)\sin(\alpha) - \cos(\alpha)\sin(\beta)$$

Следует заметить, что функции `Determinant` и `det` не упрощают полученные выражения, поэтому следует использовать функции упрощения, описанные в разделе 2.3 первой части пособия:

```
> combine(Determinant(B));
```

$$\sin(-\beta + \alpha)$$

5. Для вычисления обратной матрицы могут использоваться команда `MatrixInverse` из пакета `LinearAlgebra` или команда `inverse` из пакета `linalg`. Синтаксис этих команд следующий:

```
> MatrixInverse(A);
> inverse(A);
```

Здесь `A` — матрица, для которой строится обратная. В случае, когда матрица `A` оказывается вырожденной, то есть имеет нулевой определитель, функции `MatrixInverse`, `inverse` выведут сообщение об ошибке.

Пример вычисления обратных матриц J^{-1} и F^{-1}, где

$$J = \begin{pmatrix} 1 & 1 & 0 \\ 2 & 4 & 1 \\ 0 & 0 & 2 \end{pmatrix}, \quad F = \begin{pmatrix} 2 & 3 \\ 4 & 6 \end{pmatrix}:$$

```
> A := Matrix([[1, 1, 0], [2, 4, 1], [0, 0, 2]]):
> B := Matrix([[2, 3], [4, 6]]):
> with(LinearAlgebra): with(linalg):
> MatrixInverse(A);
```

$$\begin{bmatrix} 2 & -\frac{1}{2} & \frac{1}{4} \\ -1 & \frac{1}{2} & -\frac{1}{4} \\ 0 & 0 & \frac{1}{2} \end{bmatrix}$$

```
> invA := inverse(A);
```

$$invA := \begin{bmatrix} 2 & -\frac{1}{2} & \frac{1}{4} \\ -1 & \frac{1}{2} & -\frac{1}{4} \\ 0 & 0 & \frac{1}{2} \end{bmatrix}$$

```
> multiply(A, invA);
```

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

```
> MatrixInverse(B);
Error, (in MatrixInverse) singular matrix
> inverse(B);
Error, (in linalg:-inverse) singular matrix
> Determinant(B);
```

$$0$$

6. Ранг матрицы можно определить с помощью функции `Rank` из пакета `LinearAlgebra` или функции `rank` из пакета `linalg`. Синтаксис этих команд/функций следущий:

```
> Rank(A);
> rank(A);
```

Здесь `A` — матрица, ранг которой вычисляется.

Пример вычисления ранга матриц

$$V = \begin{pmatrix} -7 & 1 & 2 \\ 2 & 1 & -1 \\ 3 & 0 & -1 \\ 2 & 7 & -3 \end{pmatrix}, \quad U = \begin{pmatrix} x & 1 & 0 \\ 0 & z & 1 \\ xy & y & 1 \end{pmatrix}:$$

```
> V := Matrix([[-7, 1, 2], [2, 1, -1],
               [3, 0, -1], [2, 7, -3]]):
> U := Matrix([[x, 1, 0], [0, z, 1], [x*y, y, 1]]):
> with(LinearAlgebra): with(linalg):
> Rank(V); rank(V);
                          2
                          2
> Rank(U); rank(U);
                          3
                          3
```

7. Для определения собственных чисел матрицы могут использоваться следующие команды/функции:

1) `Eigenvalues` из пакета `LinearAlgebra`, синтаксис которой следующий:

   ```
   > Eigenvalues(A, opts);
   ```

   Здесь `A` — матрица, собственные значения которой определяются, `opts` — дополнительные опции. Из всех имеющихся дополнительных опции отметим две опции:

   - `implicit = s` — указывает, в каком виде будут представлены собственные числа. Если параметр `s = true`, то Maple представит собственные числа

в алгебраической форме через радикалы и функцию `RootOf`. Если параметр `s = false` (по умолчанию), то собственные числа будут представлены в виде числовых выражений, в этом случае удобно дополнительно пользоваться функцией `evalf`, описанной в разделе 2.3 первой части пособия;

- `output = s` — указывает, в каком формате собственные числа матрицы будут выведены. Возможные значения параметра `s`: `Vector[row]` (вывод в виде вектора-строки), `Vector[column]` (по умолчанию, вывод в виде вектора-столбца), `list` (вывод в виде списка).

2) `eigenvalues` из пакета `linalg`, синтаксис которой следующий:

```
> eigenvalues(A);
```

Здесь `A` — матрица, собственные значения которой определяются.

Пример вычисления собственных чисел матриц

$$A = \begin{pmatrix} 5 & 7 & 1 & 0 \\ 1 & 1 & 2 & 1 \\ 0 & 0 & 8 & 0 \\ 9 & 0 & 0 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{pmatrix}:$$

```
> A := Matrix([[5, 7, 1, 0], [1, 1, 2, 1],
               [0, 0, 8, 0], [9, 0, 0, 4]]):
> B := Matrix([[1, 2, 3], [3, 2, 1], [2, 1, 3]]):
> with(LinearAlgebra): with(linalg):
> Eigenvalues(A, implicit = true);
```

$$\begin{bmatrix} 8 \\ RootOf(\_Z^3 - 10\_Z^2 + 22\_Z - 55, index = 1) \\ RootOf(\_Z^3 - 10\_Z^2 + 22\_Z - 55, index = 2) \\ RootOf(\_Z^3 - 10\_Z^2 + 22\_Z - 55, index = 3) \end{bmatrix}$$

```
> evalf(eigenvalues(A));
```

$$8., 8.125509032,$$
$$0.937245484 + 2.427010014\mathrm{I}, 0.937245484 - 2.427010014\mathrm{I}$$

```
> eigenvalues(B);
```

$$6, \sqrt{2}, -\sqrt{2}$$

```
> Eigenvalues(B, implicit = true, output = list);
```

$$[6, RootOf(\_Z^2 - 2, index = 1), RootOf(\_Z^2 - 2, index = 2)]$$

```
> Eigenvalues(B);
```

$$\begin{bmatrix} 6 \\ \sqrt{2} \\ -\sqrt{2} \end{bmatrix}$$

8. Для определения собственных векторов матрицы могут использоваться следующие команды/функции:

1) `Eigenvectors` из пакета `LinearAlgebra`, синтаксис которой следующий:

   `> Eigenvectors(A, opts);`
   Здесь `A` — матрица, собственные векторы которой определяются, `opts` — дополнительные опции, совпадающие с опциями для команды/функции `Eigenvalues`.

2) `eigenvectors` из пакета `linalg`, синтаксис которой следующий:

   `> eigenvectors(A);`
   Здесь `A` — матрица, собственные векторы которой определяются.

Отметим, что обе функции возвращают не только собственные векторы, но и собственные числа, им соответствующие, и их кратность.

Пример вычисления собственных векторов матрицы

$$B = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{pmatrix}:$$

```
> B := Matrix([[1, 2, 3], [3, 2, 1], [2, 1, 3]]):
> with(LinearAlgebra): with(linalg):
> evalf(Eigenvectors(B));
```

$$\begin{bmatrix} 6. \\ 1.414213562 \\ -1.414213562 \end{bmatrix}, \begin{bmatrix} 1. & -0.03886827682 & -3.675417430 \\ 1. & -1.508049884 & 2.936621305 \\ 1. & 1. & 1. \end{bmatrix}$$

```
> eigenvectors(B);
```

$$\left[\sqrt{2}, 1, \left\{\left[\frac{9\sqrt{2}}{7} - \frac{13}{7} \quad \frac{5}{7} - \frac{11\sqrt{2}}{7} \quad 1\right]\right\}\right],$$
$$\left[-\sqrt{2}, 1, \left\{\left[-\frac{9\sqrt{2}}{7} - \frac{13}{7} \quad \frac{5}{7} + \frac{11\sqrt{2}}{7} \quad 1\right]\right\}\right],$$
$$\left[6, 1, \left\{\left[1 \quad 1 \quad 1\right]\right\}\right]$$

9. Получить жорданову форму матрицы можно с помощью функции `JordanForm` из пакета `LinearAlgebra`. Синтаксис этой команды/функции следующий:

```
> JordanForm(A);
```

Здесь `A` — матрица, представляемая в жордановой форме.

Пример вычисления жордановой формы матриц

$$S = \begin{pmatrix} 0 & -2 & -2 & -2 \\ -3 & 1 & 1 & -3 \\ 1 & -1 & -1 & 1 \\ 2 & 2 & 2 & 4 \end{pmatrix}, \quad W = \begin{pmatrix} 5, 7 \\ 1, 7 \end{pmatrix}:$$

```
> S := Matrix([[0, -2, -2, -2], [-3, 1, 1, -3],
               [1, -1, -1, 1], [2, 2, 2, 4]]):
> W := Matrix([[5, 7], [1, 7]]):
> with(LinearAlgebra):
```

```
> JordanForm(S);
```

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

```
> JordanForm(W);
```

$$\begin{bmatrix} 6-2\sqrt{2} & 0 \\ 0 & 6+2\sqrt{2} \end{bmatrix}$$

5.3. Векторные и тензорные операции

Векторный/тензорный анализ — раздел математики, распространяющий методы математического анализа на векторы/ тензоры в двух или более измерениях. Рассмотрим наиболее распространенные команды/функции векторного и тензорного анализа, реализованные в Maple.

В данном разделе будут рассмотрены функции из пакетов `VectorCalculus` и `linalg`.

1. Для вычисления градиента функции могут использоваться функции `Gradient` из пакета `VectorCalculus` или `grad` из пакета `linalg`, синтаксис которых следующий:

```
> Gradient(f, coordinates);
> grad(f, v, coords = s);
```

Здесь `f` — алгебраическое выражение, `coordinates` — список переменных, по которым вычисляется градиент, и указание системы координат (по умолчанию декартова), `v` — список переменных, по которым вычисляется градиент, `coords = s` — дополнительная опция, указывающая систему координат (по умолчанию декартова).

Пример вычисления градиента функций $\cos(xyz)+y^2z$ и $r\sin\phi$ (в сферических координатах):

```
> f := cos(x * y * z) + y^2 * z;
```

$$f := \cos(xyz) + y^2 z$$

```
> g := r * sin(phi);
```

$$g := r\sin(\phi)$$

```
> with(VectorCalculus): with(linalg):
> Gradient(f, [x, y, z]);
```

$$(-yz\sin(xyz))\overline{e}_x + (-zx\sin(xyz) + 2yz)\overline{e}_y + (-xy\sin(xyz) + y^2)\overline{e}_z$$

```
> grad(f, [x, y, z]);
```

$$\begin{bmatrix} -yz\sin(xyz) & -zx\sin(xyz) + 2yz & -xy\sin(xyz) + y^2 \end{bmatrix}$$

```
> Gradient(g, spherical[r, theta, phi]);
```

$$(\sin(\phi))\overline{e}_\rho + (0)\overline{e}_\theta + \left(\frac{\cos(\phi)}{\sin(\theta)}\right)\overline{e}_\phi$$

```
> grad(g, [r, theta, phi], coords = spherical);
```

$$\begin{bmatrix} \sin(\phi) & 0 & \dfrac{\cos(\phi)}{\sin(\theta)} \end{bmatrix}$$

2. Дивергенция вектора может быть вычислена с помощью функции `diverge` из пакета `linalg`. Синтаксис этой команды/функции следующий:

```
> diverge(f, v, coords = s);
```

Здесь `f` — вектор, дивергенция которого вычисляется, `v` — список переменных, по которым вычисляются производные, `coords = s` — дополнительная опция, указывающая систему координат (по умолчанию декартова). Следует заметить, что для вычисления дивергенции векторное поле объявляется с помощью функции `vector` из пакета `linalg` (синтаксис этой функции полностью совпадает с функцией `Vector`).

Пример вычисления дивергенции векторных полей $a = (x, y, e^z)^T$, $b = (\rho\cos\phi, \rho)^T$ (в полярных координатах):

```
> with(linalg):
> a := vector([x, y, exp(z)]);
```

$$a := \begin{bmatrix} x & y & e^z \end{bmatrix}$$

```
> b := vector([rho * cos(phi), rho]);
```

$$b := \begin{bmatrix} \rho\cos(\phi) & \rho \end{bmatrix}$$

```
> diverge(a, [x, y, z]);
```

$$2 + e^z$$

```
> diverge(b, [rho, phi], coords = polar);
```

$$2\cos(\phi)$$

3. Для вычисления ротора векторного поля может использоваться команда `curl` из пакета `linalg`. Синтаксис этой команды/функции следующий:

```
> curl(f, v, coords = s);
```

Здесь `f` — вектор, ротор которого вычисляется, `v` — список переменных, по которым вычисляются производные, `coords = s` — дополнительная опция, указывающая систему координат (по умолчанию декартова). Так же, как и для функции `diverge`, векторное поле объявляется с помощью функции `vector` из пакета `linalg`.

Необходимо помнить, что функция вычисления ротора, как и функция векторного произведения, определена только для трехмерных векторов.

Пример вычисления ротора векторов $p = (z^3, y^2, x)^T$ и $k = (\rho, \rho\phi, z\cos\phi)^T$ (в цилиндрических координатах):

```
> with(linalg):
> p := vector([z^3, y^2, x]);
```

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

```
> k := vector([rho, rho * phi, z * cos(phi)]);
```

$$k := \begin{bmatrix} \rho & \rho\phi & z\cos(\phi) \end{bmatrix}$$

```
> curl(p, [x, y, z]);
```

$$\begin{bmatrix} 0 & 3z^2 - 1 & 0 \end{bmatrix}$$

```
> curl(k, [rho, phi, z], coords = cylindrical);
```

$$\begin{bmatrix} -\dfrac{z\sin(\phi)}{\rho} & 0 & 2\phi \end{bmatrix}$$

4. Для вычисления якобиана в Maple реализованы функция `Jacobian` в пакете `VectorCalculus` и функция `jacobian` из пакета `linalg`. Синтаксис этих команд/функции следующий:

```
> Jacobian(f, v = p);
> jacobian(f, v);
```

Здесь `f` — вектор алгебраических выражений, `v` — список переменных, по которым вычисляются производные. Также для функции `Jacobian` есть возможность указать точку `p`, в которой вычисляется якобиан. Если точка `p` не указана, то якобиан будет вычислен в общем виде, без каких-либо подстановок значений.

Пример вычисления якобиана $(x^2 + z, e^y \cos z, xyz)^T$ в точке $x_0 = (0, 1, \pi)$:

```
> f := Vector([x^2 + z, exp(y) * cos(z), x * y * z]):
> with(VectorCalculus): with(linalg):
> Jacobian(f, [x, y, z] = [0, 1, Pi]);
```

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & -\mathrm{e} & 0 \\ \pi & 0 & 0 \end{bmatrix}$$

```
> J := jacobian(f, [x, y, z]);
```

$$\begin{bmatrix} 2x & 0 & 1 \\ 0 & \mathrm{e}^y\cos(z) & -\mathrm{e}^y\sin(z) \\ yz & xz & xy \end{bmatrix}$$

```
> simplify(subs({x = 0, y = 1, z = Pi}, evalm(J)));
```

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & -\mathrm{e} & 0 \\ \pi & 0 & 0 \end{bmatrix}$$

5. Для построения матрицы Гессе или гессиана (матрицы вторых производных) можно использовать либо функцию `Hessian` из пакета `VectorCalculus`, либо функцию `hessian` из пакета `linalg`. Синтаксис этих команд/функций следующий:

```
> Hessian(f, v = p);
> hessian(f, v);
```

Здесь `f` — алгебраическое выражение, `v` — список переменных, по которым вычисляются производные. Также для функции `Hessian` есть возможность указать точку `p`, в которой вычисляется матрица Гессе. Если точка `p` не указана, то гессиан будет вычислен в общем виде, без каких-либо подстановок значений.

Пример вычисления матрицы Гессе для функции $g = x^2y + 3xy^2$ в точке $p = (1, 1)$:

```
> g := x^2 * y + 3 * x * y^2:
> with(VectorCalculus): with(linalg):
> Hessian(g, [x, y] = [1, 1]);
```

$$\begin{bmatrix} 2 & 8 \\ 8 & 6 \end{bmatrix}$$

```
> H := hessian(g, [x, y]);
```

$$\begin{bmatrix} 2y & 2x + 6y \\ 2x + 6y & 6x \end{bmatrix}$$

```
> simplify(subs({x = 1,y = 1}, evalm(H)));
```

$$\begin{bmatrix} 2 & 8 \\ 8 & 6 \end{bmatrix}$$

5.4. Другие функции

1. Для создания векторов или матриц, заполненных случайными числами, в Maple используются команды `RandomVector` и `RandomMatrix` из пакета `LinearAlgebra` соотвественно. Эти команды имеют следующий синтаксис:

```
> RandomVector(N, generator = a..b, options);
```

```
> RandomMatrix(M, K, generator = a..b, options);
```

Здесь `N` — количество элементов или размерность вектора, `M, K` — количество строк и столбцов матрицы соотвественно, `generator = a..b` — опция, задающая диапазон значений случайных чисел, `options` — дополнительные опции, которые читателю предлагается изучить самостоятельно.

Пример заполнения пятимерного вектора a целыми случайными числами в диапазоне $[-12, 5]$ и матрицы A размерами 3×3 целыми случайными числами в диапазоне $[15, 200]$:

```
> with(LinearAlgebra):
> a := RandomVector(5, generator = -12..5);
```

$$a := \begin{bmatrix} 2 \\ 5 \\ -4 \\ -11 \\ -6 \end{bmatrix}$$

```
> A := RandomMatrix(3, 3, generator = 15..200);
```

$$A := \begin{bmatrix} 175 & 81 & 177 \\ 68 & 51 & 192 \\ 119 & 33 & 101 \end{bmatrix}$$

Пример заполнения трехмерного вектора b вещественными случайными числами в диапазоне $[-1, 3]$ и матрицы B размерами 3×2 вещественными случайными числами в диапазоне $[-5, 1]$:

```
> with(LinearAlgebra):
> b := RandomVector(3, generator = -1.0..3.0);
```

$$b := \begin{bmatrix} -0.499268939581529292 \\ 2.28761305495585843 \\ 2.97627397845559782 \end{bmatrix}$$

```
> B := RandomMatrix(3, 2, generator = -5.0..1.0);
```

$$B := \begin{bmatrix} 0.525248769880235500 & -2.25206508730711707 \\ -0.236150173424267251 & -2.54761304752091799 \\ -2.07458643706233836 & -4.78135252653631859 \end{bmatrix}$$

2. Матричную экспоненту можно вычислить с помощью функций `MatrixExponential` из пакета `LinearAlgebra` и `exponential` из пакета `linalg`, синтаксис которых следующий:

```
> MatrixExponential(A, t, options);
> exponential(A, t);
```

Здесь `A` — квадратная матрица, `t` — необязательный параметр, `options` — дополнительные опции.

Пример вычисления e^B, где

$$B = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 2 \\ 1 & 0 & 0 \end{pmatrix}:$$

```
> B := Matrix([[0, 0, 1], [1, 0, 2], [1, 0, 0]]):
> with(LinearAlgebra): with(linalg):
> MatrixExponential(B);
```

$$\begin{bmatrix} \frac{\mathrm{e}}{2} + \frac{\mathrm{e}^{-1}}{2} & 0 & \frac{\mathrm{e}}{2} - \frac{\mathrm{e}^{-1}}{2} \\ \frac{3\mathrm{e}}{2} + \frac{\mathrm{e}^{-1}}{2} - 2 & 1 & \frac{3\mathrm{e}}{2} - \frac{\mathrm{e}^{-1}}{2} - 1 \\ \frac{\mathrm{e}}{2} - \frac{\mathrm{e}^{-1}}{2} & 0 & \frac{\mathrm{e}}{2} + \frac{\mathrm{e}^{-1}}{2} \end{bmatrix}$$

```
> exponential(B, t);
```

$$\begin{bmatrix} \frac{\mathrm{e}^{t}}{2} + \frac{\mathrm{e}^{-t}}{2} & 0 & \frac{\mathrm{e}^{t}}{2} - \frac{\mathrm{e}^{-t}}{2} \\ \frac{3\mathrm{e}^{t}}{2} + \frac{\mathrm{e}^{-t}}{2} - 2 & 1 & \frac{3\mathrm{e}^{t}}{2} - \frac{\mathrm{e}^{-t}}{2} - 1 \\ \frac{\mathrm{e}^{t}}{2} - \frac{\mathrm{e}^{-t}}{2} & 0 & \frac{\mathrm{e}^{t}}{2} + \frac{\mathrm{e}^{-t}}{2} \end{bmatrix}$$

```
> MatrixExponential(B, t);
```

$$\begin{bmatrix} \frac{e^t}{2}+\frac{e^{-t}}{2} & 0 & \frac{e^t}{2}-\frac{e^{-t}}{2} \\ \frac{3e^t}{2}+\frac{e^{-t}}{2}-2 & 1 & \frac{3e^t}{2}-\frac{e^{-t}}{2}-1 \\ \frac{e^t}{2}-\frac{e^{-t}}{2} & 0 & \frac{e^t}{2}+\frac{e^{-t}}{2} \end{bmatrix}$$

3. Для вычисления характеристического уравнения матрицы могут использоваться функции `CharacteristicPolynomial` из пакета `LinearAlgebra` и `charpoly` из пакета `linalg`. Синтаксис этих команд/функций следующий:

```
> CharacteristicPolynomial(A, lambda);
> charpoly(A, lambda);
```

Здесь `A` — квадратная матрица, `lambda` — имя переменной, которое будет использовано в полиноме.

Пример построения характеристического полинома матриц

$$K = \begin{pmatrix} 7 & 0 & -10 \\ 5 & -1 & 2 \\ 0 & 2 & -3 \end{pmatrix}, \quad L = \begin{pmatrix} \cos\alpha & \sin\beta \\ \sin\alpha & \cos\beta \end{pmatrix}:$$

```
> K := Matrix([[7, 0, -10], [5, -1, 2], [0, 2, -3]]):
> L := Matrix([[cos(alpha), sin(beta)],
               [sin(alpha), cos(beta)]]):
> with(LinearAlgebra): with(linalg):
> CharacteristicPolynomial(K, lambda);
```

$$\lambda^3 - 3\lambda^2 - 29\lambda + 107$$

```
> charpoly(K, x);
```

$$x^3 - 3x^2 - 29x + 107$$

```
> pol := CharacteristicPolynomial(L, lambda);
```

$$pol := \lambda^2 + (-\cos(\beta) - \cos(\alpha))\lambda - \sin(\alpha)\sin(\beta) + \cos(\beta)\cos(\alpha)$$

```
> combine(pol);
```

$$\lambda^2 - \lambda\cos(\beta) - \lambda\cos(\alpha) + \cos(\alpha+\beta)$$

```
> combine(charpoly(L, zeta));
```

$$\cos(\alpha+\beta) - \zeta\cos(\beta) - \cos(\alpha)\zeta + \zeta^2$$

4. В задачах, связанных, например, с вычислением показателей Ляпунова, возникает необходимость ортогонализации векторов. Для ортогонализации Грама-Шмидта в Maple существует функция `GramSchmidt` из пакета `LinearAlgebra`. Синтаксис этой команды следующий:

```
> GramSchmidt([V1, V2, ..., Vn], options);
```

Здесь `[V1, V2, ..., Vn]` — список векторов, `options` — дополнительные опции. Из дополнительных опций отметим опцию `conjugate = s` и опцию `normalized = s`. Опция `conjugate = s` отвечает за комплексное сопряжение и нужна при работе с комплексными векторами, параметр `s` может иметь два значения: `true` (по умолчанию) или `false`. Опция `normalized = s` указывает, будут возвращаемые вектора нормированы или нет, параметр `s` может иметь два значения: `true` или `false` (по умолчанию).

Пример ортогонализации векторов $V_1 = (2,-1,0)^T$, $V_2 = (0,1,3)^T$, $V_3 = (-2,1,-3)^T$:

```
> V1 := Vector([2, -1, 0]):
> V2 := Vector([0, 1, 3]):
> V3 := Vector([-2, 1, -3]):
> with(LinearAlgebra):
> GramSchmidt([V1, V2, V3]);
```

$$\left[\begin{bmatrix}2\\-1\\0\end{bmatrix}, \begin{bmatrix}\frac{2}{5}\\\frac{4}{5}\\3\end{bmatrix}, \begin{bmatrix}\frac{18}{49}\\\frac{36}{49}\\-\frac{12}{49}\end{bmatrix}\right]$$

```
> GramSchmidt([V1,V2,V3], normalized = true);
```

$$\left[\begin{bmatrix}\frac{2\sqrt{5}}{5}\\ -\frac{\sqrt{5}}{5}\\ 0\end{bmatrix}, \begin{bmatrix}\frac{2\sqrt{5}}{35}\\ \frac{4\sqrt{5}}{35}\\ \frac{3\sqrt{5}}{7}\end{bmatrix}, \begin{bmatrix}\frac{3}{7}\\ \frac{6}{7}\\ -\frac{2}{7}\end{bmatrix}\right]$$

5. Для построения LU и QR разложений матрицы могут использоваться команды `LUDecomposition` и `QRDecomposition` из пакета `LinearAlgebra` соотвественно. Читателю предлагается самостоятельно изучить эти команды/функции.

Задачи

5.1. Найти $\det A$, $\det B$, $A + B$, $A \cdot B$, $A - B$, где A и B — матрицы вида

$$A = \begin{pmatrix}5 & 2\\ 7 & 3\end{pmatrix}, \quad B = \begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}.$$

5.2. Даны матрица Q и векторы a, b вида

$$Q = \begin{pmatrix}3 & 4 & -5\\ 8 & 7 & -2\\ 2 & -1 & 9\end{pmatrix}, \quad a = (1, 1, 8), \quad b = (-5, 0, 3).$$

Найти Q^T, Q^{-1}, $Q^{-1}Q$, $Qa - Q^T b$, $[a, b]$, (a, b), $|Qa|$.

5.3. Решить систему, не используя функцию `solve`

$$\begin{cases}2x + 3y + 5z = 10\\ 3x + 7y + 4z = 3\\ x + 2y + 2z = 3\end{cases}$$

5.4. Проверить тождества:

$$[[x, y], z] + [[y, z], x] + [[z, x], y] = 0,$$
$$[Ma, Mb] = \det M (M^{-1})^T [a, b],$$

где x, y, z, a, b — векторы, M — матрица.

6. Программирование в Maple

6.1. Условный оператор

1. Условный оператор, как и во многих языках программирования, в Maple начинается с ключевого слова `if`. Синтаксис наиболее полного условного оператора имеет следующий вид:

```
> if condition1 then
  commands1;
  elif condition2 then
  commands2;
  elif condition3 then
  ...
  else
  commandsN;
  end if;
```

Следует обратить внимание, что весь условный оператор записывается в одной области ввода, то есть только первая строка начинается с символа `[>` . Для того чтобы писать команды в одной области, но при этом не в одну строку, необходимо использовать комбинацию клавиш **Shift+Enter** для перехода на следующую строку в рамках одного поля ввода.

Рассмотрим подробно работу приведенного выше кода. Выполнение условного оператора начинается с проверки первого указанного условия *condition*1, которое может быть как простым, так и составным. В зависимости от результата проверки условия *condition*1 идет ветвление:

- если *condition*1 оказывается верным, то есть условие выполняется, то Maple перейдет к выполнению команд *commands*1, после чего условный оператор закончит работу;

- если *condition*1 оказывается неверным, то есть условие не выполняется, то Maple не переходит к выполнению команд *commands*1, а переходит к проверке второго указанного условия *condition*2.

Так же, как и в случае для *condition*1, в зависимости от результата проверки условия *condition*2 идет ветвление: если *condition*2 верно, то Maple выполнит *commands*2 и завершит работу условного оператора, если *condition*2 неверно, то будет проверяться условие *condition*3, и т. д. Если все проверяемые условия *condition*1, *condition*2 и т. д. оказались неверными, то Maple перейдет к выполнению *commands*N, после чего закончит работу условного оператора.

Ветви с дополнительными условиями `elif`—`then`, а также ветвь `else` не являются обязательными для корректной работы условного оператора. Другими словами, в зависимости от решаемой задачи указанные ветви могут быть пропущены. При этом ключевое слово `then` и строка `end if`, указывающая на окончание условного оператора, не могут быть пропущены, их отсутствие является синтаксической ошибкой.

Примеры **некорректных** условных операторов (отсутствие ключевого слова `then` и строки `end if`):

```
> if 5 > 3
  a := sin(Pi/4);
  end if;
Error, missing operator or ';'
```

```
> if 5 > 3
  a := sin(Pi/4);
Warning, premature end of input, use <Shift> + <Enter>
to avoid this message.
```

Как видно из примеров, Maple не выводит сообщения с конкретным названием ошибки, а только сообщает, что в написанном коде что-то неправильно. Для корректной работы вышеприведенный код необходимо привести к следующему виду:

```
> if 5 > 3 then
  a := sin(Pi/4);
  end if;
```

$$a := \frac{\sqrt{2}}{2}$$

Следует обратить внимание на то, какой символ стоит в конце `end if`: точка запятой «;» или двоеточие «:». Если после `end if` стоит «;», то результат всех выполненных команд будет выведен в рабочий лист не зависимо от указанного символа двоеточия «:» после этих команд. Если же после `end if` стоит «:», то ничего выведено не будет, также будет игнорироваться вывод результатов команд, в конце которых указана «;». В этом случае удобно воспользоваться функцией `print`, в которой указаны значения, которые необходимо вывести на рабочий лист.

Пример вычисления функции $f(x)$ в точке x_0, где функция $f(x)$ имеет следующий вид:

$$f(x) = \begin{cases} x^2 + x, & x > 5 \\ x^3 - 3x, & x < -1 \\ x \cos x, & \text{иначе} \end{cases}$$

```
> x0 := 1.34:
> if x0 > 5 then
  f := subs(x = x0, x^2 + x);
  elif x0 < -1 then
  f := subs(x = x0, x^3 - 3 * x);
  else
  f := subs(x = x0, x * cos(x));
  end if:
> evalf(f);
```

0.3065287625

2. Работая с условными операторами, необходимо задавать простые или сложные условия, в которых используются операторы сравнения. Существующие операторы сравнения и их запись в Maple приведены в таблице 1.

Как известно, сложными условиями называют условия, составленные из простых с помощью логических операторов. Логический оператор «И» записывается в Maple как `and`, а логический оператор «ИЛИ» как `or`.

Таблица 1. Операторы сравнения

Оператор	Название	Запись в Maple
$>$, $<$	операторы «больше», «меньше»	`>`, `<`
$\geqslant$, $\leqslant$	операторы «больше или равно», «меньше или равно»	`>=`, `<=`
$=$	оператор «равно»	`=`
$\neq$	оператор «неравно»	`<>`

Пример условного оператора `if` со сложным условием, который вычисляет следующую функцию

$$g(x) = \begin{cases} \operatorname{sh} x, & \text{если } x \in [-1, 5) \text{ или } x = -4 \\ \operatorname{ch} x, & \text{если } x \in (-10, -5] \text{ или } x = 8 \\ \operatorname{th} x, & \text{иначе} \end{cases}$$

```
> x := -1.25461:
> if (x >= -1 and x < 5) or x = -4 then
  g := sinh(x);
  print(g);
  elif (x > -10 and x <= -5) or x = 8 then
  g := cosh(x);
  print(g);
  else
  g := tanh(x);
  print(g);
  end if:
```

$$-0.8495713078$$

6.2. Циклы

1. Цикл со счетчиком в Maple, как и во многих языках программирования, начинается с ключевого `for` и имееет следующий синтаксис:

```
> for counter from start by step to finish do
  commands;
  end do;
```

или

```
> for counter in [value1, value2, ...] do
  commands;
  end do;
```

Здесь *counter* — имя переменной-счетчика, *start* и $finish$ — начальное и конечное значения счетчика, *step* — шаг, с которым изменяется переменная *counter* (если не указан фрагмент `by` *step*, то по умолчанию шаг берется равным единице), [*value*1, *value*2, ...] — список значений, которые принимает счетчик (не обязательно числовые). В теле цикла *commands* могут быть вложенные циклы и условные операторы. Обратим внимание, что цикл `for` так же, как и условный оператор, записывается в одной области ввода, то есть каждая строка не начинается с символа `[>` .

Рассмотрим подробно работу приведенных выше кодов:

- в первом случае работа цикла `for` начинается с того, что переменной-счетчику *counter* присваивается значение *start*, после чего выполняются команды в теле цикла *commands*. Выполнив эти команды, переменная-счетчик увеличивается на указанный шаг *step* (по умолчанию единица) и проверяется, не стало ли значение переменной *counter* больше $finish$. Если *counter* не превосходит $finish$, то выполняется еще одна итерация цикла и счетчик опять увеличивается на величину шага *step*, иначе цикл завершается;

- во втором случае работа цикла `for` начинается с того, что переменной-счетчику *counter* присваивается первое значение из списка *value*1, после чего выполняются команды в теле цикла *commands*. Выполнив эти команды, переменной-счетчику присваивается второе значение из списка *value*2, после чего опять выполняются команды в теле цикла *commands*. Цикл продолжается до тех пор, пока не будут перебраны все значения в списке [*value*1, *value*2, ...].

Для цикла `for` так же, как и для условного оператора `if`, следует обращать внимание, какой символ стоит в конце цикла после `end do`: «`;`» (точка с запятой) или «`:`» (двоеточие). В случае, если

прописано `end do;`, то будут выводится результаты работы всех команд в теле цикла на каждой итерации. В случае, если прописано `end do:`, то вывод в рабочий лист производиться не будет (если в теле цикла не прописана команда `print`).

Пример подсчета суммы первых N натуральных чисел разными способами:

```
> N := 7: sumN := 0:
> for i from 1 to N do
  sumN := sumN + i;
  end do;
```

$$sumN := 1$$
$$sumN := 3$$
$$sumN := 6$$
$$sumN := 10$$
$$sumN := 15$$
$$sumN := 21$$
$$sumN := 28$$

```
> N := 7: sumN := 0:
> for i from 1 to N do
  sumN := sumN + i;
  end do:
> sumN;
```

$$28$$

```
> sumN := 0:
> for i in [1,2,3,4,5,6,7] do
  sumN := sumN + i;
  end do:
> sumN;
```

$$28$$

Пример вычисления суммы $1 + n^2 + n^4 + ... + n^k$, где $k = 10$:

```
> s := 0: k := 10:
> for i from 0 by 2 to k do
  s := s + n^i;
 end do:
```

```
> s;
```

$$n^{10} + n^8 + n^6 + n^4 + n^2 + 1$$

2. Цикл с предусловием в Maple начинается с ключевого слова `while` и имеет следующий синтаксис:

```
> while conditions do
  commands;
  end do;
```

Здесь *conditions* — условие (простое или составное) работы цикла, *commands* — команды, в теле цикла. Обратим внимание, что цикл `while` записывается в одной области ввода.

Приведенный выше код начинает свою работу с проверки условия *conditions*. Если данное условие **верно**, то программа переходит к выполнению команд *commands*. После выполнения команд повторяется проверка условия *conditions*. В тот момент, когда условие *conditions* перестанет быть верными, цикл прекратится.

Пример определения номера первого нулевого элемента в случайном массиве:

```
> with(LinearAlgebra):
> A := RandomVector(8, generator = -2..2);
```

$$\begin{bmatrix} -1 \\ -1 \\ 0 \\ -1 \\ -1 \\ -2 \\ 0 \\ -2 \end{bmatrix}$$

```
> i := 1:
> while A[i]<>0 do
  i := i + 1;
  end do:
> i;
```

3

3. Цикл с постусловием в Maple имеет ключевой слово `until` и имеет следующий синтаксис:

```
> do
  commands;
  until conditions;
```

Здесь *conditions* — условие (простое или составное) завершения работы цикла, *commands* — команды в теле цикла. Обратим внимание, что цикл `until` записывается в одной области ввода.

Приведенный выше код гарантированно один раз выполняет команды тела цикла *commands*. После чего идет проверка условия *conditions*. Если данное условие **неверно**, то программа снова переходит к выполнению команд *commands*. В момент, когда условие *conditions* становятся верным, цикл прекращается.

Пример определения номера первого нулевого элемента в случайном массиве:

```
> with(LinearAlgebra):
> A := RandomVector(8, generator = -2..2);
```

$$\begin{bmatrix} -1 \\ -1 \\ 0 \\ -1 \\ -1 \\ -2 \\ 0 \\ -2 \end{bmatrix}$$

```
> i := 0:
> do
  i := i + 1;
  until A[i]=0:
> i;
```

3

4. Для всех выше перечисленных циклов определена команда `break`, которая прерывает работу цикла.

6.3. Создание собственных функций и процедур

1. Как правило, функции используются при описании громоздких или повторяющихся математических выражений. Имя функции всегда совпадает с именем переменной, в которой она хранится. Синтаксис объявления функции следующий:

```
> var := (x1, ..., xN) -> expression;
```

Здесь `var` — имя переменной, оно же имя функции, `(x1, ..., xN)` — список аргументов функции, `expression` — выражение, зависящее от переменных `x1, ..., xN`.

Для вызова объявленной функции необходимо указать имя переменной/функции `var` и в скобках указать значения аргументов функции `(val1, ..., valN)`, при которых ее необходимо вычислить:

```
> var(val1, ..., valN);
```

В результате, в выражение `expression` будут подставлены значения `x1 = val1, ..., xN = valN`.

Функции оказываются удобными при построении графиков функций с несколькими параметрами, анимаций, а также при численном анализе функции.

Пример объявления функции $g(x, y, z) = \cos e^{x+y+z} - \sin \mathrm{ch}(xyz)$ и вычисления ее значения в следующих точках $(x, y, z) = (0, 2, -4)$, $(x, y, z) = (1, 2.5, -0.45)$, $(x, y, z) = (10, -0.2, 4)$:

```
> g := (x, y, z) -> cos(exp(x+y+z)) - sin(cosh(x*y*z));
```

$$g := (x, y, z) \mapsto \cos(\mathrm{e}^{x+y+z}) - \sin(\cosh(xyz))$$

```
> g(0, 2, -4);
```

$$\cos(\mathrm{e}^{-2}) - \sin(1)$$

```
> g(1, 2.5, -0.45);
```

$$-1.631727606$$

```
> g(10, -0.2, 4);
```

$$-1.813032741$$

2. Процедуры в Maple обозначаются ключевым словом `proc` и имеют следующий синтаксис объявления:

```
> ProcName := proc(x1, ..., xN)
  commands;
  end proc;
```

Здесь $ProcName$ — имя процедуры, `(x1, ..., xN)` — список входных параметров, $commands$ — команды, выполняемые в теле процедуры (в том числе циклы и условные операторы). Обратим внимание, что процедура так же, как и циклы и условный оператор, записывается в одной области ввода.

Если в конце `end proc` стоит точка с запятой «`;`», то на рабочий лист будут продублированы объявление процедуры и все команды, указанные внутри нее. Если в конце `end proc` стоит двоеточие «`:`», то ничего выведено не будет.

Для вызова объявленной процедуры необходимо указать имя процедуры $ProcName$, после чего в скобках указать значения входных параметров `(val1, ..., valN)`:

```
> ProcName(val1, ..., valN);
```

Следует помнить, что при выборе имени процедуры так же, как и для имени переменной, запрещено использовать зарезервированные слова языка Maple, например, `for`, `end`, `and` или имена существующих команд Maple и т. д.

Пример объявления процедуры для нахождения суммы двух чисел:

```
> sum2Num := proc(a, b)
  a + b;
  end proc;
```

$$sum2Num := \mathbf{proc}(a, b)\ a + b;\ \mathbf{end\ proc}$$

```
> sum2Num(5, 10);
```

15

Часто при работе с процедурами возникает необходимость объявления дополнительных переменных внутри процедуры. Эти переменные принято заранее объявлять либо локальными (`local`), существующими только внутри процедуры, либо глобальными

(`global`), существующими во всем исполняемом файле. Имена локальных и глобальных переменных перечисляются после ключевых слов `local` и `global` соответственно. Если тип переменной не указан, то Maple выведет предупреждение о том, что переменная по умолчанию будет считаться локальной.

Пример процедуры, вычисляющей среднее арифметическое по массиву A размера N:

```
> with(LinearAlgebra);
> N := 5: A := RandomVector(N, generator = 0..5);
```

$$\begin{bmatrix} 2 \\ 5 \\ 2 \\ 1 \\ 4 \end{bmatrix}$$

```
> MidVal := proc(A, N)
  local i, sum;
  sum := 0;
  for i from 1 to N do
  sum := sum + A[i];
  end do:
  evalf(sum / N);
  end proc:
> MidVal(A, N);
```

2.800000000

В случае, когда процедура должна возвращать несколько значений или значение какой-то конкретной переменной, используется команда `return`, после которой через запятую указываются переменные и/или выражения, которые процедура должна вернуть. После срабатывания команды `return` процедура завершает работу. Если же `return` не прописана в теле процедуры, то по умолчанию будет возвращаться значение последней выполненной команды в процедуре.

Пример процедуры, которая возвращает сумму, разность и произведение двух чисел:

```
> multFunc := proc(a, b)
  local s, r, m;
  s := a + b; r := a - b; m := a * b;
  return s, r, m;
  end proc:
> multFunc(5, 0);
```

$$5, 5, 0$$

3. Достаточно часто результат работы процедуры необходимо отобразить на графике. Отрисовка данных, полученных из процедуры через функцию/команду `plot` (если речь идет о двумерном графике), несколько отличается от построения графика обычной функции. Синтаксис команды `plot` для процедуры с одним входным параметром:

```
> plot(procName, a..b, opts);
```

Синтаксис команды `plot` для процедуры с несколькими входными параметрами:

```
> plot('procName(x1, ..., xN)', xk = a..b, opts);
```

Здесь `procName` — имя процедуры, результат работы которой рисуется, `x1, ..., xN` — входные параметры процедуры, `a..b` — диапазон значений параметра `xk`, `opts` — дополнительные опции, описанные в разделе 4.1.1. первой части пособия.

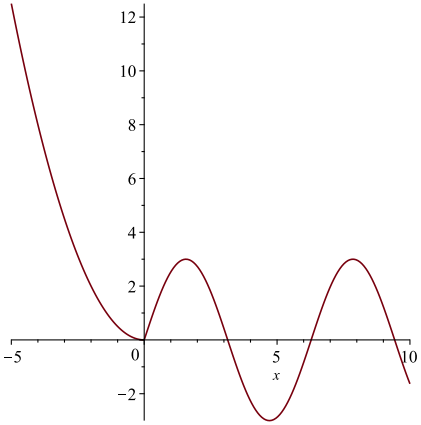
Пример построения графика кусочной функции

$$f(x) = \begin{cases} ax^2, & x \leqslant 0 \\ b \sin x, & x > 0 \end{cases},$$

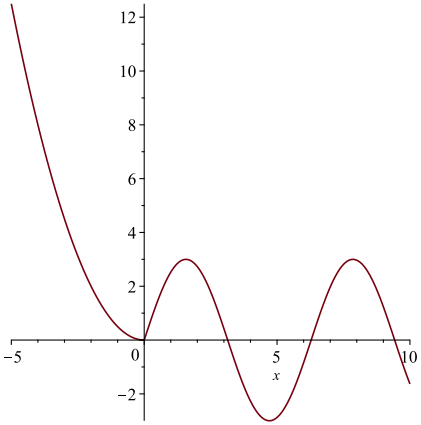
заданной через процедуру (при $a = 0.5$, $b = 3$):

```
> f := proc(a, b, x)
  if x<=0 then
  a * x^2;
  else  b * sin(x);
  end if:
  end proc:
> plot(f(0.5, 3, x), x = -5..10);
Error, (in f) cannot determine if this expression is
true or false: x <= 0
```

```
> plot('f(0.5, 3, x)', x = -5..10);
```




```
> f := proc(x)
  if x<=0 then 0.5 * x^2;
  else 3 * sin(x);
  end if:
  end proc:
> plot(f, -5..10);
```



6.4. Примеры решения задач

1. Найти первые N нулей функции Бесселя $J_0(x)$.

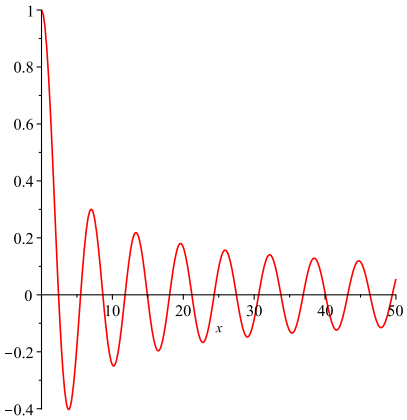
Создадим функцию, которая будет возвращать значение функции $J_0(x)$:

```
> restart;
> f := (x) -> BesselJ(0, x);
```

$$f := x \mapsto \mathrm{BesselJ}(0, x)$$

Построим график этой функции, чтобы убедиться, что функция возвращает корректные значения, и визуализировать точки пересечения с осью Ox:

```
> P := plot(f(x), x = 0..50, color = red);
```



Уравнение $J_0(x) = 0$ является трансцендентным, следовательно, его придется решать численно с помощью функции `fsolve`, описанной в разделе 3.4 первой части пособия. Из графика видно, что функция имеет множество точек пересечения с осью Ox, поэтому необходимо локализовать эти участки (условно $[a, b]$) и уже на каждом участке численно искать решение уравнения $J_0(x) = 0$. Для этого напишем следующий цикл:

```
> N := 15:
> a := 0: delta := 0.1:
```

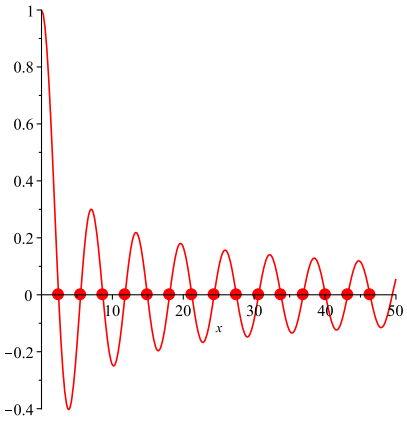
```
> for i from 1 to N do
  while f(a)*f(a+delta)>0 do
  a := a + delta:
  end do:
  x0[i] := fsolve(f(x) = 0, x = a..a+delta);
  a := a + delta;
  end do:
> x0[1]; x0[2];
```

$$2.404825558$$

$$5.520078110$$

Для того чтобы убедиться, что найденные значения действительно являются нулями функции $J_0(x)$, отметим их на графике:

```
> with(plots):
> display(P, pointplot([seq([x0[k], 0], k = 1..N)],
          color = red, symbol = solidcircle,
          symbolsize = 20));
```



2. Построить уравнения движения системы, которая описывается следующим лагранжианом:

$$L = x\sqrt{1 + \dot{x}^2 + \dot{y}^2} + y.$$

Создадим переменную L, в которую запишем лагранжиан системы:

```
> restart;
> L := x * sqrt(1 + vx + vy) + y;
```

$$L := x\sqrt{1 + vx^2 + vy^2} + y.$$

Здесь `vx` обозначает производную $\dot{x}$, а `vy` — производную $\dot{y}$. Для построения уравнений движения системы воспользуемся уравнениями Лагранжа-Эйлера:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_k}\right) - \frac{\partial L}{\partial q_k} = 0,$$

где $q = (x, y)$, $\dot{q} = (\dot{x}, \dot{y})$. Для вычисления производной $\dfrac{d}{dt}$ воспользуемся правилом дифференцирования

$$\frac{dA}{dt} = \sum_k \frac{\partial A}{\partial q_k}\dot{q}_k$$

и объявим следующую процедуру:

```
> q := Vector([x, y, vx, vy]):
> dq := Vector([vx, vy, ax, ay]):
> dt := proc(A)
  local sum := 0, i;
  for i from 1 to 4 do
  sum:=sum + diff(A, q[i]) * dq[i];
  end do:
  simplify(sum);
  end proc:
```

Теперь, используя эту процедуру, получим уравнения движения системы:

```
> eqX := simplify(dt(diff(L, vx)) - diff(L, x));
```

$$eqX := \frac{-vy^4 + (x\,ax - vx^2 - 2)vy^2 - x\,vx\,vy\,ay + x\,ax - vx^2 - 1}{(vx^2 + vy^2 + 1)^{3/2}}$$

```
> eqY := dt(diff(L, vy)) - diff(L, y);
```

$$eqY := \frac{vx\,vy^3 + (-ax\,vx\,x + vx^3 + vx)vy + x\,ay(vx^2 + 1)}{(vx^2 + vy^2 + 1)^{3/2}} - 1$$

Задачи

6.1. Через цикл заполнить матрицу следующим образом:

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 3 & 1 \\ 4 & 4 & 4 & 4 \end{pmatrix}$$

6.2. Даны вещественные числа a, b, c. Определить, имеет ли уравнение $ax^2 + bx + c = 0$ вещественные корни?

6.3. Вычислить сумму $1 + \dfrac{x}{1!} + \dfrac{x^2}{2!} + \ldots + \dfrac{x^n}{n!}$.

6.4. Вывести на экран следующую последовательность (через цикл `while`):

$$\sin^5(5x), \quad \sin^4(4x), \quad \sin^3(3x), \quad \sin^2(2x), \quad \sin x, \quad 1.$$

6.5. Написать процедуру для вычисления интеграла с переменным верхним пределом.

$$\int_0^t \sin \tau e^{\tau} d\tau,$$

t — входной параметр процедуры.

6.6. Написать рекурсивную процедуру для вычисления факториала (для вызова процедурой самой себя использовать команду `thisproc()`, которую читателю предлагается изучить самостоятельно).

6.7. Построить с использованием функции на одном графике:

$$x^3 + y^3 = 3axy, \quad a = 1, 2, 3.$$

6.8. Создать анимацию и нарисовать несколько графиков при разных значениях параметра c:

$$(x^2 + y^2)^2 - 2c^2(x^2 - y^2) = a^4 - c^4.$$

7. Решение дифференциальных уравнений

7.1. Обыкновенные дифференциальные уравнения

Для построения аналитического, приближенного или численного решения дифференциального уравнения или системы дифференциальных уравнений используется команда/функция `dsolve`, синтаксис которой следующий:

```
> dsolve(ODE, vars, options);
```

Здесь `ODE` — дифференциальное уравнение или система дифференциальных уравнений, а также начальные или граничные условия, если они заданы, `vars` — неизвестные функции, относительно которых разрешаются уравнения, `options` — дополнительные опции.

Далее рассмотрим варианты построения решений и дополнительные опции, для них существующие.

7.1.1. Аналитическое решение

1. Для построения аналитического решения дифференциального уравнения или системы дифференциальных уравнений **без начальных или граничных условий** используется следующий синтаксис команды `dsolve`:

```
> dsolve(diff_eq, var);
> dsolve({diff_eq1, ..., diff_eqN}, {var1, ..., varN});
```

Здесь `diff_eq` — дифференциальное уравнение, `{diff_eq1, ..., diff_eqN}` — система дифференциальных уравнений, `var` — неизвестная функция, `{var1, ..., varN}` — неизвестные функции, относительно которых решается система уравнений.

При записи дифференциальных уравнений необходимо явно указывать зависимость неизвестной функции от независимой переменной.

Пример записи дифференциальных уравнений $y' + xy^2 = \operatorname{tg} x$ и $y^{(IV)} + 2 = y$ в Maple:

```
> eq1 := diff(y(x), x) + x * y(x)^2 = tan(x);
```

$$eq1 := \frac{\mathrm{d}}{\mathrm{d}x} y(x) + xy(x)^2 = \tan(x)$$

```
> eq2 := diff(y(x), x$4) + 2 = y(x);
```

$$eq2 := \frac{\mathrm{d}^4}{\mathrm{d}x^4} y(x) + 2 = y(x)$$

Также следует обратить внимание, что производные, входящие в уравнения, записываются с помощью функции `diff`, которая была рассмотрена в разделе 3.1 первой части пособия.

Пример решения дифференциального уравнения $x^2y' + xy = 1$ и системы уравнений:

$$\begin{cases} \dot{x} = 2x + y, \\ \dot{y} = 4y - x \end{cases}$$

```
> eq := x^2 * diff(y(x), x) + x * y(x) = 1;
```

$$eq := x^2 \left(\frac{\mathrm{d}}{\mathrm{d}x} y(x) \right) + xy(x) = 1$$

```
> dsolve(eq, y(x));
```

$$y(x) = \frac{\ln(x) + \_C1}{x}$$

```
> eqs := {diff(x(t), t) = 2*x(t) + y(t),
          diff(y(t), t) = 4*y(t) - x(t)};
```

$$eqs := \left\{ \frac{\mathrm{d}}{\mathrm{d}t} x(t) = 2x(t) + y(t),\ \frac{\mathrm{d}}{\mathrm{d}t} y(t) = 4y(t) - x(t) \right\}$$

```
> sol := dsolve(eqs, {x(t), y(t)});
```

$$sol := \left\{ x(t) = \mathrm{e}^{3t}(\_C2\, t + \_C1),\ y(t) = \mathrm{e}^{3t}(\_C2\, t + \_C1 + \_C2) \right\}$$

Для проверки построенного решения можно воспользоваться функцией `odetest`, синтаксис которой следующий:

```
> odetest(sol, ODE);
```

Здесь `sol` обозначает решение дифференциального уравнения или системы дифференциальных уравнений `ODE`. Если построенное решение корректно, то функция `odetest` вернет значение 0:

```
> odetest(sol, eqs);
```

$$\{0\}$$

2. Для построения аналитического решения дифференциального уравнения или системы дифференциальных уравнений **с начальными или граничными условиями** используется следующий синтаксис команды `dsolve`:

```
> dsolve({diff_eq, cond}, var);
> dsolve({diff_eq1, ..., diff_eqN, cond1, ..., condN},
         {var1, ..., varN});
```

Здесь `diff_eq` и `diff_eq1, ..., diff_eqN` — дифференциальное уравнение и система дифференциальных уравнений, `cond` и `cond1, ..., condN` — начальные или граничные условия, `var` и `{var1, ..., varN}` — неизвестные функции.

Запись начальных/граничных условий в Maple осуществляется по следующим правилам:

1) Начальное условие на функцию, например, вида $y(0) = 1$ записывается как `y(0) = 1`.

2) Начальное условие на первую производную функции, например, вида $y'(0) = 1$ записывается как `D(y)(0) = 1`.

Пример решения уравнения $y'' = \lambda y$, $y(0) = 1$, $y'(1) = 0$ и системы дифференциальных уравнений

$$\begin{cases} \dot{x} = 2x + y, \\ \dot{y} = 4y - x \end{cases}, \quad x(0) = 1,\ y(0) = 2.$$

```
> dsolve({diff(y(x), x, x) = lambda * y(x),
          y(0) = 1, D(y)(1) = 0}, y(x));
```

$$y(x) = \frac{\mathrm{e}^{-\sqrt{\lambda}}\mathrm{e}^{\sqrt{\lambda}x}}{\mathrm{e}^{\sqrt{\lambda}} + \mathrm{e}^{-\sqrt{\lambda}}} + \frac{\mathrm{e}^{\sqrt{\lambda}}\mathrm{e}^{-\sqrt{\lambda}x}}{\mathrm{e}^{\sqrt{\lambda}} + \mathrm{e}^{-\sqrt{\lambda}}}$$

```
> eqs := diff(x(t), t) = 2*x(t) + y(t),
          diff(y(t), t) = 4*y(t) - x(t);
```

$$eqs := \frac{\mathrm{d}}{\mathrm{d}t}x(t) = 2x(t) + y(t),\ \frac{\mathrm{d}}{\mathrm{d}t}y(t) = 4y(t) - x(t)$$

```
> ics := x(0) = 1, y(0) = 2;
```

$$ics := x(0) = 1, y(0) = 2;$$

```
> sol := dsolve({eqs, ics}, {x(t), y(t)});
```

$$sol := \left\{x(t) = \mathrm{e}^{3t}(t+1),\ y(t) = \mathrm{e}^{3t}(t+2)\right\}$$

Для проверки полученных решений также используется команда/функция `odetest`, синтаксис которой описан выше.

К полученному решению можно обратиться по имени переменной, в которую оно записано, а также применять к нему команды/функции, описанные в разделе 3 первой части пособия. Примеры использования этих команд:

```
> subs(t = 1, sol);
```

$$\left\{x(1) = 2\mathrm{e}^3,\ y(1) = 3\mathrm{e}^3\right\}$$

```
> rhs(sol[1]);
```

$$\mathrm{e}^{3t}(t+1)$$

Решение дифференциального уравнения не всегда можно представить через элементарные функции. В некоторых случаях Maple может выразить решение через специальные функции.

Пример решения уравнения Бесселя нулевого порядка $y'' + \dfrac{1}{x}y' + y = 0$:

```
> dsolve(diff(y(x), x$2) + 1/x * diff(y(x), x)
             + y(x) = 0, y(x));
```

$$y(x) = \_C1\mathrm{BesselJ}(0, x) + \_C2\mathrm{BesselY}(0, x)$$

Следует помнить, что возможности пакета Maple ограничены, и для большинства дифференциальных уравнений невозможно получить общее решение в квадратурах. Когда команда `dsolve` не может построить точное решение, на рабочий лист ничего не выводится и Maple по умолчанию переходит к выполнению других команд. В таких случаях решение возможно построить приближенно через степенные ряды или численно.

7.1.2. Приближенное решение

Для построения приближенного решения используется команда/функция `dsolve` с опцией `series`. При этом **необходимо** указывать начальные условия для решаемого дифференциального уравнения, в противном случае решение построить не удастся. Синтаксис `dsolve` следующий:

```
> dsolve({diff_eq, cond}, var, series);
> dsolve({diff_eq1, ..., diff_eqN, cond1, ..., condN}
         {var1, ..., varN}, series);
```

Здесь `diff_eq` и `diff_eq1, ..., diff_eqN` — дифференциальное уравнение и система дифференциальных уравнений, `cond` и `cond1, ..., condN` — начальные или краевые условия, `var` и `{var1, ..., varN}` — неизвестные функции.

Пример решения дифференциального уравнения $y'' = \cos(xy)$ с начальными условиями $y(0) = 1$, $y'(0) = 1$:

```
> ODE := {diff(y(x), x$2) = cos(x*y(x)),
            y(0) = 1, D(y)(0) = 1};
```

$$ODE := \left\{ \frac{\mathrm{d}^2}{\mathrm{d}x^2} y(x) = \cos(xy(x)), y(0) = 1, \mathrm{D}(y)(0) = 1 \right\}$$

```
> dsolve(ODE, y(x));
> dsolve(ODE, y(x), series);
```

$$y(x) = 1 + x + \frac{1}{2}x^2 - \frac{1}{24}x^4 - \frac{1}{20}x^5 + \mathrm{O}(x^6)$$

Пример решения нелинейной системы дифференциальных уравнений:

$$\begin{cases} \dot{x} = e^y \\ \dot{y} = \sin(tx) \end{cases}$$
$$x(0) = 0,\ y(0) = 1.$$

```
> ODE := {diff(y(t), t) = sin(t*x(t)),
          diff(x(t), t) = exp(y(t)),
             y(0) = 1, x(0) = 0};
```

$$ODE := \left\{ \frac{\mathrm{d}}{\mathrm{d}t} x(t) = e^{y(t)}, \frac{\mathrm{d}}{\mathrm{d}t} y(t) = \sin(tx(t)), x(0) = 1, y(0) = 1 \right\}$$

```
> dsolve(ODE, {x(t), y(t)}, series);
```

$$\left\{x(t) = \mathrm{e}t + \frac{1}{12}\mathrm{e}^2 t^4 + \mathrm{O}(t^6),\ y(t) = 1 + \frac{1}{3}\mathrm{e}t^3 + \mathrm{O}(t^6)\right\}$$

7.1.3. Численное решение

1. Для численного решения дифференциальных уравнений используется команда/функция `dsolve` с опцией `numeric`. Для построения численного решения **необходимо** указывать начальные условия для решаемого дифференциального уравнения, в противном случае решение построить не удастся. Синтаксис `dsolve` следующий:

```
> dsolve({diff_eq, cond}, var, numeric, opts);
> dsolve({diff_eq1, ..., diff_eqN, cond1, ..., condN}
         {var1, ..., varN}, numeric, opts);
```

Здесь `diff_eq` и `diff_eq1, ..., diff_eqN` — дифференциальное уравнение и система дифференциальных уравнений, `cond` и `cond1, ..., condN` — начальные условия, `var` и `{var1,...,varN}` — неизвестные функции, `opts` — дополнительные опции для настройки численного решения.

При вызове функции `dsolve` с опцией `numeric` необходимо записывать ее результат в какую-либо переменную, поскольку результатом выполнения команды `dsolve` является процедура, содержащая в себе **все** численное решение. Конкретные значения из решения уравнений можно получить уже с помощью этой процедуры (см. раздел 6.3).

Пример численного решения следующего дифференциального уравнения

$$\ddot{\omega} = \omega + t\sinh\omega, \quad \omega(0) = 0, \quad \omega'(0) = 1.$$

```
> eq := diff(omega(t),t$2)=omega(t)+t*sinh(omega(t));
```

$$eq := \frac{\mathrm{d}^2}{\mathrm{d}t^2}\omega(t) = \omega(t) + t\sinh(\omega(t))$$

```
> ics := omega(0) = 0, D(omega)(0) = 1;
```

$$ics := \omega(0) = 0, \mathrm{D}(\omega)(0) = 1$$

```
> dsol := dsolve({eq, ics}, omega(t), numeric);
```

$$dsol := \mathbf{proc}(x\_rkf45)...\mathbf{end\ proc}$$

```
> dsol(1);
```

$$\left[t = 1., \omega(t) = 1.27800188634370, \frac{\mathrm{d}}{\mathrm{d}t}\omega(t) = 2.00554447458781\right]$$

Из примера видно, что в качестве аргумента процедуры выводится строка x_rkf45. Данная строка указывает на то, какой численный метод используется для решения дифференциального уравнения, в частности, x_rkf45 означает метод Рунге-Кутты-Фельберга 4-5го порядков [6].

Рассмотрим наиболее часто используемые опции команды/функции `dsolve` при численном решении:

- `method = s` — опция для выбора численного метода решения дифференциальных уравнений. Возможные значения параметра `s`: `rkf45` (по умолчанию), `ck45`, `rosenbrock`, `bvp`, `rkf45_dae`, `ck45_dae`, `rosenbrock_dae`, `dverk78`, `lsode`, `gear` и др.

- `output = s` — опция, указывающая формат возвращаемых значений. Возможные значения параметра `s`: `procedurelist` (по умолчанию), `listprocedure`, `operator` и др.

  Разницу в работе этих опций покажем на примере решения дифференциального уравнения $y'' = e^{y+\cos x}$, $y(0) = 1$, $y'(0) = 5$:

  ```
  > ODE := {diff(y(x),x,x) = exp(y(x) + cos(x)),
            y(0) = 1, D(y)(0) = 5}:
  > ds:=dsolve(ODE,y(x),numeric);
  ```

  $$ds := \mathbf{proc}(x\_rkf45)...\mathbf{end\ proc}$$

```
> ds:=dsolve(ODE,y(x),numeric,output=listprocedure);
```

$$ds := \left[x = \mathbf{proc}(x)...\mathbf{end\ proc},\ y(x) = \mathbf{proc}(x)...\mathbf{end\ proc},\ \frac{\mathrm{d}}{\mathrm{d}x}y(x) = \mathbf{proc}(x)...\mathbf{end\ proc}\right]$$

```
> ds:=dsolve(ODE,y(x),numeric,output=operator);
```

$$ds := [x = \mathbf{proc}(x)...\mathbf{end\ proc},\ y = \mathbf{proc}(x)...\mathbf{end\ proc},\ \mathrm{D}(y) = \mathbf{proc}(x)...\mathbf{end\ proc}]$$

- `range = n1..n2` — опция, определяющая диапазон независимой переменной `[n1, n2]`, в котором вычисляется решение. Данная опция полезна в связке с функцией `odeplot`, которая будет рассмотрена далее.
- `abserr = n` — числовое значение `n` определяет предел допустимой абсолютной ошибки.
- `relerr = n` — числовое значение `n` определяет предел допустимой относительной ошибки.

2. Как правило, при численном решении дифференциального уравнения возникает необходимость визуализации полученных результатов, то есть построения графиков функций. Для этого можно воспользоваться командой/функцией `odeplot`, расположенной в пакете `plots`. Синтаксис этой команды следующий:

```
> odeplot(dif_sol, [var1(t), var2(t)], t = a..b, opts);
```

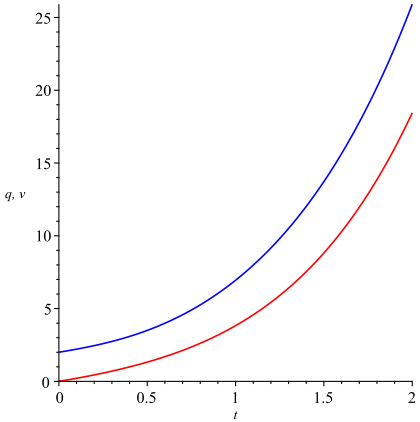
или

```
> odeplot(dif_sol, [t, var(t)], t = a..b, opts);
```

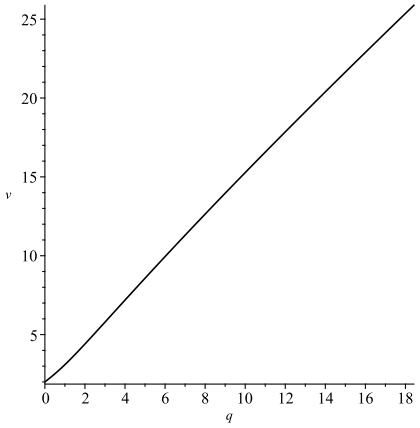
Здесь `dif_sol` — имя процедуры, содержащей численное решение дифференциального уравнения или системы уравнений, `[var1(t), var2(t)]` или `[t, var(t)]` — имена функций для построения, с указанием независимой переменной `t`, `t = a..b` — имя независимой и диапазон ее значений `[a, b]`, `opts` — дополнительные опции для графики, совпадающие с опциями для функции `plot`.

Пример решения системы дифференциальных уравнений $\dot{q} = v$, $\dot{v} = v + \ln(q^2 + 1)$ с начальными условиями $q(0) = 0$, $v(0) = 2$ и построения зависимостей $q(t)$, $v(t)$, $v(q(t))$:

```
> eq := diff(q(t), t) = v(t),
            diff(v(t), t) = v(t) + ln(1+q(t)^2):
> ds := dsolve({eq,v(0)=2,q(0)=0},{q(t),v(t)},numeric):
> with(plots):
> odeplot(ds, [[t, q(t)], [t, v(t)]],
              t = 0..2, color = [red, blue]);
```




```
> odeplot(ds, [q(t), v(t)], t = 0..2, color = black);
```



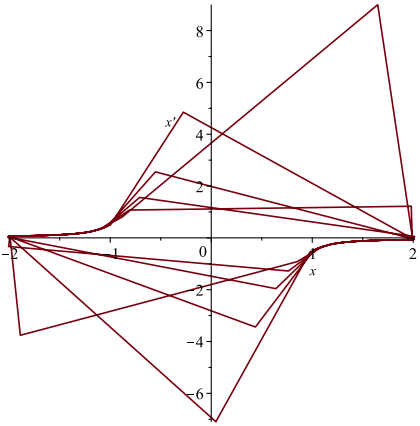
Рассмотрим некоторые опции, которые не указывались для функции/команды `plot`, однако могут быть полезны при работе с `odeplot`.

- `frame = n` — опция, задающая количество кадров в анимации. При указании этой опции функция `odeplot` будет возвращать не график указанной зависимости, а ее анимацию. В качестве параметра анимации будет выбираться независимая переменная. Читателю предлагается самостоятельно рассмотреть использование данной опции.

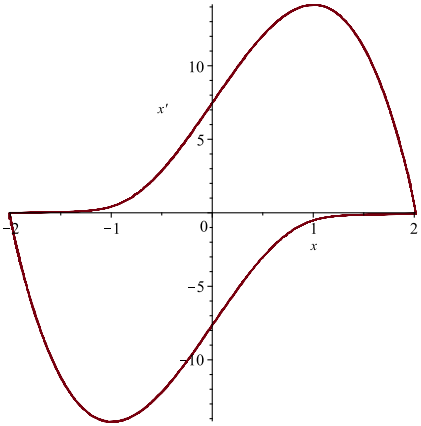
- `refine = n` — опция, указывающая, во сколько раз (в `n` раз) больше точек следует использовать для построения графика (по умолчанию `n = 1`). Данная опция может использоваться **только совместно** с опцией `range`, которая указывает диапазон независимой переменной.

  Пример использования данной опции:

```
> ode := diff(x(t), t, t)
      + 10 * (x(t)^2 - 1) * diff(x(t), t) + x(t) = 0:
> ics := x(0) = 2, D(x)(0) = 0:
> sol := dsolve({ics, ode}, numeric, range = 0..100):
> odeplot(sol, [x(t), diff(x(t), t)], t = 0..100);
```



```
> odeplot(sol, [x(t), diff(x(t), t)],
                t = 0..100, refine = 2);
```



Команда/функция `odeplot` также может использоваться для отрисовки трехмерных графиков. Читателю предлагается рассмотреть этот вопрос самостоятельно.

3. Отметим, что для построения численного решения дифференциальных уравнений можно воспользоваться функциями, описанными в разделе 4 первой части пособия. Однако предварительно необходимо извлечь численные решения в отдельные переменные с помощью опции `output=listprocedure` в команде `dsolve` и команде/функции `eval`. После чего работать с этими решениями как с явно заданными функциями, где в качестве аргумента выступает независимая переменная.

В случае, когда решение представляет собой периодическую функцию, удобно использовать именно этот способ построения графиков. Это связанно с тем, что функция `plot` имеет дополнительную опцию `discont = s`, отвечающую за обработку разрывов на графике, (параметр `s` может иметь значение `true` или `false` (по умолчанию)). При этом команда/функция `odeplot` данной опции не имеет.

Пример построения зависимости $y(x(t))$, где функции $y(t)$, $x(t)$ определяются из уравнения:

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

с начальными условиями $x(0) = -1$, $y(0) = 1$:

```
> sys := diff(x(t),t) = y(t),
         diff(y(t),t) = sin(x(t));
```

$$sys := \frac{\mathrm{d}}{\mathrm{d}t}x(t) = y(t),\ \frac{\mathrm{d}}{\mathrm{d}t}y(t) = \sin(x(t))$$

```
> ics := y(0) = 1, x(0) = -1;
```

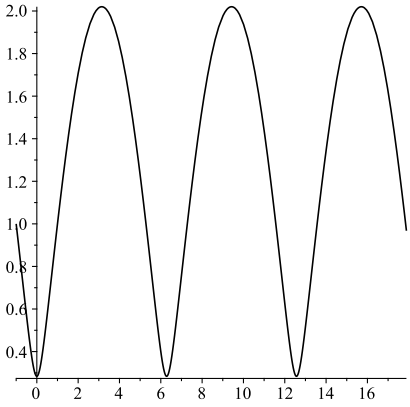
$$ics := y(0) = 1, x(0) = -1$$

```
> dsol := dsolve({sys,ics}, {x(t), y(t)}, numeric,
          output = listprocedure):
> X := eval(x(t), dsol); Y := eval(y(t), dsol);
```

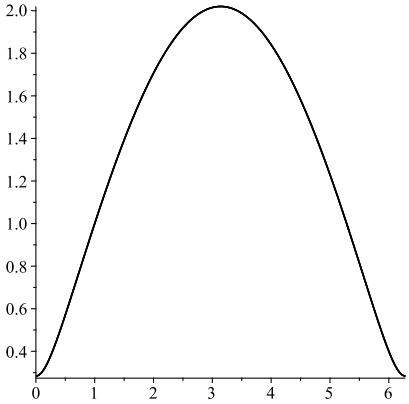
$$X := \mathbf{proc}(t)...\mathbf{end\ proc}$$

$$Y := \mathbf{proc}(t)...\mathbf{end\ proc}$$

```
> plot([X(t), Y(t), t = 0..20], color = black);
```




```
> X_period := t-> X(t) - 2 * Pi * floor(X(t)/(2*Pi));
> plot([X_period(t), Y(t), t = 0..20],
        color = black, discont = true);
```



В приведенном коде используется команда/функция `floor(x)`, которая возвращает наибольшее целое число, меньшее или равное `x`. Для расширения кругозора читателю также предлагается изучить следующие функции: `trunc`, `round`, `frac`, `ceil`.

7.1.4. Построение фазового портрета

Для исследования дифференциальных уравнений второго порядка (или системы двух уравнений первого порядка) часто строят фазовый портрет. Фазовый портрет исследуемой системы — это совокупность фазовых траекторий для всевозможных начальных условий.

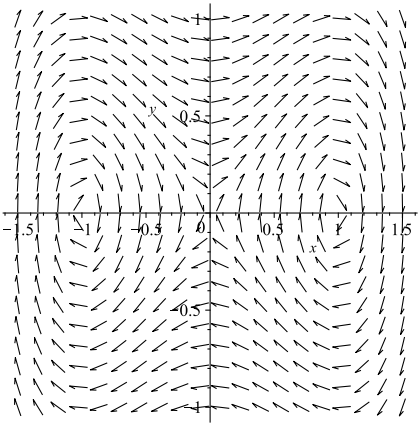
Существует три подхода к построению фазового портрета в пакете Maple. Далее все они будут рассмотрены для следующей системы:

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

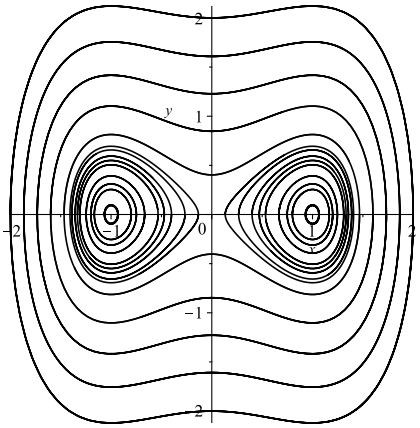
1. С использованием функции/команды `DEplot` из пакета `DEtools`. Данная команда строит поле скорости указанной системы уравнений, если не указаны начальные условия. Указав набор начальных условий, функция `DEplot` нарисует соответствующие им фазовые траектории. Опции этой команды/функции можно посмотреть в **Справке**.

Приведем пример построения фазового портрета системы с использованием команды DEplot:

```
> eqs := diff(x(t),t)=y(t), diff(y(t),t)=x(t)-x(t)^3:
> with(DEtools):
> DEplot({eqs}, [x(t), y(t)], t = 0..1,
            x = -1.5..1.5, y = -1..1, color = black);
```

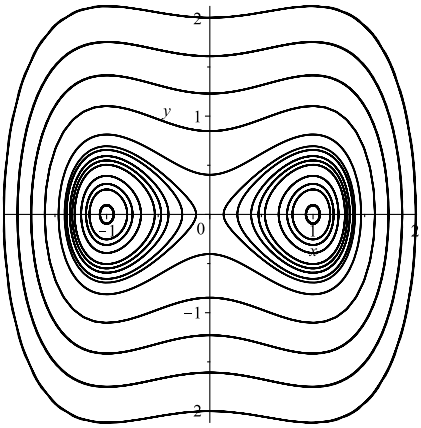



```
> ics := [seq([x(0) = -2+4*i/30, y(0)=0], i=0..30)]:
> DEplot({eqs}, [x(t), y(t)], t = 0..10, ics,
          stepsize = 0.05, arrows = none,
          linecolor = black, thickness = 1);
```



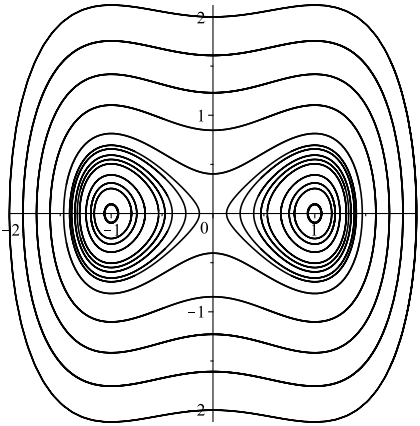
2. С использованием функции/команды `odeplot` из пакета `plots`:

```
> with(plots):
> N := 30:
> for i from 1 to N do
  sol:= dsolve({eqs, x(0) = -2 + 4 * i / N, y(0) = 0},
        numeric);
  A[i]:= odeplot(sol, [x(t), y(t)], color = black);
  end do:
> display(seq(A[i], i = 1..N));
```



3. С использованием стандартной команды `plot`:

```
> N := 30:
> for i from 1 to N do
  sol:= dsolve({eqs, x(0) = -2 + 4 * i / N, y(0) = 0},
        numeric, output = listprocedure);
  X := eval(x(t), sol); Y := eval(y(t), sol);
  A[i]:= plot([X(t), Y(t), t=-5..5], color = black);
  end do:
> display(seq(A[i], i = 1..N));
```



7.2. Уравнения в частных производных

Для построения общего аналитического или численного решения уравнения или системы уравнений в частных производных используется команда/функция `pdsolve`. Синтаксис этой команды следующий:

```
> pdsolve(PDE, vars, options);
```

или

```
> pdsolve({PDE, ics}, vars, options);
```

Здесь `PDE` — уравнение или система уравнений в частных производных, `ics` — граничные условия, `vars` — неизвестные функции, относительно которых разрешаются уравнения (могут не указываться), `options` — дополнительные опции.

7.2.1. Аналитическое решение

Рассмотрим сначала построение аналитического решения и отметим, что нахождение такого решения зачастую занимает длительное время. Для того чтобы контролировать процесс поиска решения, рекомендуется предварительно вызывать команду `infolevel[pdsolve]`, которая имеет следующий синтаксис:

```
> infolevel[pdsolve] := n:
```

Здесь `n` задает количество или уровень выводимой информации.

Параметр `n` может иметь следующие значения:

- 1 — вывод информации, которую необходимо сообщить пользователю (значение по умолчанию);
- 2, 3 — выводится общая информация, включая используемый метод или алгоритм;
- 4, 5 — вывод подробной информации о том, как решается проблема.

Пример построения общего решения уравнения

$$x\frac{\partial u}{\partial x} - y\frac{\partial u}{\partial y} = 0.$$

```
> infolevel[dsolve] := 2:
> PDE := x * diff(u(x,y), x) - y * diff(u(x,y), y) = 0;
```

$$x\left(\frac{\partial}{\partial x}u(x,y)\right) - y\left(\frac{\partial}{\partial y}u(x,y)\right) = 0.$$

```
> ans := pdsolve(PDE, u(x,y));

    Methods for first order ODEs:
    --- Trying classification methods ---
    trying a quadrature
    trying 1st order linear
    <- 1st order linear successful
    Methods for first order ODEs:
    --- Trying classification methods ---
    trying a quadrature
    trying 1st order linear
    <- 1st order linear successful
```

$$ans := u(x,y) = \_F1(yx)$$

Для проверки построенного решения можно воспользоваться функцией `pdetest`, синтаксис которой следующий:

```
> pdetest(sol, PDE);
```

Здесь `sol` обозначает решение уравнения или системы уравнений

в частных производных, записанных в переменную PDE. Если построенное решение корректно, то функция pdetest вернет значение 0:

```
> pdetest(ans, PDE);
```

$$0$$

Часто при работе с командой/функцией pdsolve оказывается полезной опция HINT = s, которая позволяет указать метод решения или вид неизвестной функции. Указанное в опции HINT выражение команда pdsolve воспринимает как отправную точку при поиске решения. Возможные значения параметра s:

- ‘+‘ — заставляет pdsolve начинать поиск решения с попытки разделения переменных в сумму;
- ‘*‘ — заставляет pdsolve начинать поиск решения с попытки представления переменных в виде произведения;
- ‘TWS‘ — поиск решения типа бегущей волны через степенной ряд $\mathrm{th}(\xi)$, где ξ представляет собой линейную комбинацию независимых переменных;
- strip — поиск решения через уравнения характеристик (только для уравнения первого порядка);
- *any algebraic expression* — заставляет pdsolve сначала искать решение в виде указанного алгебраического выражения.

Пример решения уравнения в частных производных второго порядка

$$S \cdot \frac{\partial^2 S}{\partial x \partial y} + \frac{\partial S}{\partial x}\frac{\partial S}{\partial y} = 1,$$

при поиске укажем искать решение в виде $\sqrt{f(x,y)}$:

```
> PDE := S(x, y) * diff(S(x, y), y, x) +
         diff(S(x, y), x) * diff(S(x, y), y) = 1;
```

$$S(x,y)\left(\frac{\partial^2}{\partial x \partial y}S(x,y)\right) + \left(\frac{\partial S}{\partial x}S(x,y)\right)\left(\frac{\partial S}{\partial y}S(x,y)\right) = 1$$

```
> pdsolve(PDE, HINT = sqrt(f(x, y)));
```

$$S(x,y) = \sqrt{\_F2(x) + \_F1(y) + 2yx}$$

Пример решения системы уравнений в частных производных

$$-\frac{\partial^2 F}{\partial r^2} + \frac{\partial^2 F}{\partial s^2} + \frac{\partial H}{\partial r} + \frac{\partial G}{\partial s} + s = 0,$$
$$\frac{\partial^2 F}{\partial r^2} + 2\frac{\partial^2 F}{\partial s \partial r} + \frac{\partial^2 F}{\partial s^2} - \frac{\partial H}{\partial r} + \frac{\partial G}{\partial s} - r = 0.$$

```
> sys := {-diff(F(r, s), r, r) + diff(F(r, s), s, s) +
          diff(H(r), r) + diff(G(s), s) + s = 0,
          diff(F(r, s), r, r) + 2*diff(F(r, s), r, s)+
          diff(F(r, s), s, s) - diff(H(r), r)+
          diff(G(s), s) - r = 0}:
> pdsolve(sys, {F, H, G});
```

$$\left\{F(r,s) = \_F1(s) + \_F2(r) + \_F3(s-r) - \frac{r^2(r-3s)}{12},\right.$$
$$G(s) = -\frac{s^2}{4} - \frac{d}{ds}\_F1(s) + \_C2,$$
$$\left.H(r) = -\frac{r^2}{4} + \frac{d}{dr}\_F2(r) + \_C1\right\}$$

7.2.2. Численное решение

Для построения численного решения в функции/команде `pdsolve` необходимо указать опцию `numeric`, а также начальные и граничные условия к решаемым уравнениям. Синтаксис `pdsolve` следующий:

```
> pdsolve(PDE, ICs, numeric, options);
```

Здесь `PDE` — уравнение или система уравнений в частных производных, `ICs` — начальные и граничные условия,`options` — дополнительные опции. Для численного решения функция `pdsolve` имеет достаточно много дополнительных опции `options`, рассмотрим некоторые из них:

- `time=s` и `range=a..b` — опции, позволяющие определить переменную времени `s` и указать диапазон ее значений `a..b`;

- `spacestep = n` — опция, указывающая величину шага по пространственной сетке. По умолчанию имеет значение $\frac{1}{20}$;
- `timestep = n` — опция, указывающая величину шага по времени;
- `abstol = n` — опция, указывает верхнюю границу локальной оценки временных и пространственных ошибок на один шаг;
- `mintimestep = n` — опция, определяющая минимальное значение временного шага, необходимого для вычисления решения. По умолчанию равно $\frac{1}{1000}$ от начального временного шага;
- `maxtimestep = n` — опция, определяющая максимальное значение временного шага.

При вызове функции `pdsolve` с опцией `numeric` необходимо записывать ее результат в какую-либо переменную, поскольку результатом выполнения команды `pdsolve` является так называемый **модуль**, содержащий в себе все численное решение[1]. Конкретные значения из решения уравнений можно получить с помощью этого модуля, как будет показано в примерах ниже.

Пример численного решения уравнения теплопроводности на ограниченном интервале

$$\begin{cases} \dfrac{\partial T}{\partial t} = \dfrac{\partial^2 T}{\partial x^2}, \ x \in [0, 2] \\ T(x, 0) = 1 \\ T(0, t) = 0 \\ T'(2, t) = 0 \end{cases}$$

```
> PDE := diff(T(x, t), t) = diff(T(x, t), x, x);
```

$$PDE := \frac{\partial}{\partial t} T(x, t) = \frac{\partial^2}{\partial x^2} T(x, t)$$

[1] В рамках данного методического пособия модули в Maple не рассматриваются, читателю предлагается изучить эту тему самостоятельно.

```
> ICs := {T(0, t) = 0, T(x, 0) = 1, D[1](T)(2, t) = 0}:
> pds := pdsolve(PDE, ICs, numeric);
```

$$pds := \textbf{module}()...\textbf{endmodule}$$

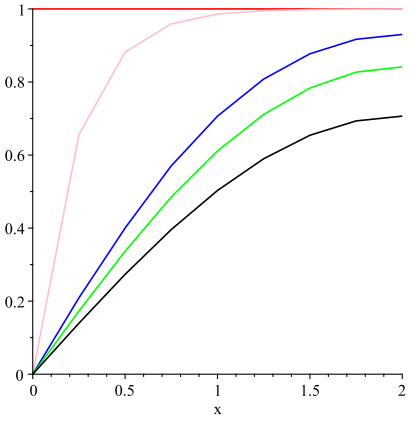
Для того чтобы обратиться к полученному модулю, используется следующий синтаксис:

```
> pds:-module_command(arg);
```

Здесь `pds` — имя переменной, в которую записан модуль, `module_command` — название вызываемой команды в модуле (возможные команды: `plot`, `plot3d`, `animate`, `value` и `settings`)[2], `arg` — аргументы команды `module_command`.

Пример построения графиков распределения температуры (из примера выше) в разные моменты времени:

```
> p1 := pds:-plot(t = 0, color = red):
  p2 := pds:-plot(t = 0.1, color = pink):
  p3 := pds:-plot(t = 0.5, color = blue):
  p4 := pds:-plot(t = 0.7, color = green):
  p5 := pds:-plot(t = 1, color = black):
> with(plots):
> display(p1, p2, p3, p4, p5);
```



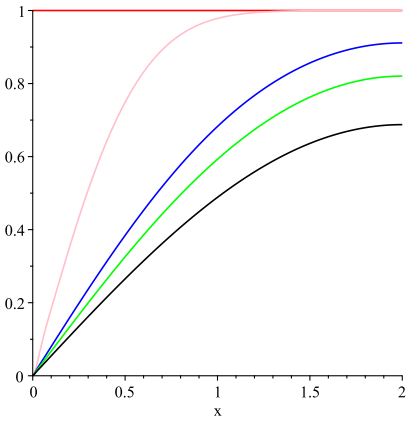
[2]Синтаксис использования команд модуля, полученного из `pdsolve`, подробно приведен в **Справке**.

Из рисунка видно, что линии немного ломанные, это связанно с тем, что значения шага по времени и по пространственной сетке равны значениям по умолчанию. Определим эти значения вручную с использованием опций `spacestep` и `timestep`:

```
> pds := pdsolve(PDE, ICs, numeric,
                 spacestep = 0.01, timestep = 0.01);
```

$$pds := \textbf{module}()...\textbf{endmodule}$$

```
> p1 := pds:-plot(t = 0, color = red):
  p2 := pds:-plot(t = 0.1, color = pink):
  p3 := pds:-plot(t = 0.5, color = blue):
  p4 := pds:-plot(t = 0.7, color = green):
  p5 := pds:-plot(t = 1, color = black):
> display(p1, p2, p3, p4, p5);
```



Пример численного решения волнового уравнения с непрерывным начальным распределением:

$$\begin{cases} \dfrac{\partial u}{\partial t} = -\dfrac{1}{2}\dfrac{\partial u}{\partial x} \\ u(x,0) = 0.5e^{-16(x-0.5)^2} \\ u(0,t) = 0.01 \end{cases}$$

и с начальным распределением, имеющим разрывы:

$$\begin{cases} \dfrac{\partial u}{\partial t} = -\dfrac{1}{2}\dfrac{\partial u}{\partial x} \\ u(x,0) = \begin{cases} 0, & x \leqslant 0.5 \text{ или } x \geqslant 1 \\ 0.5, & \text{иначе} \end{cases} \\ u(0,t) = 0.01 \end{cases}$$

```
> PDE := diff(u(x, t), t) = -0.5 * diff(u(x, t), x);
```

$$PDE := \frac{\partial}{\partial t}u(x,t) = -0.5\frac{\partial}{\partial x}u(x,t)$$

```
> IC1 := {u(0,t)=0.01, u(x,0)=exp(-16*(x-0.5)^2)/2};
```

$$IC1 := \left\{u(0,t) = 0.01, u(x,0) = \frac{\mathrm{e}^{-16(x-0.5)^2}}{2}\right\}$$

```
> IC2 := {u(0, t) = 0.01,
          u(x, 0)=piecewise(x<=0.5 or x>=1, 0, 0.5)}:
> pds1 := pdsolve(PDE, IC1, numeric, time = t,
          range = 0..2, spacestep = 0.01,
          timestep = 0.01);
```

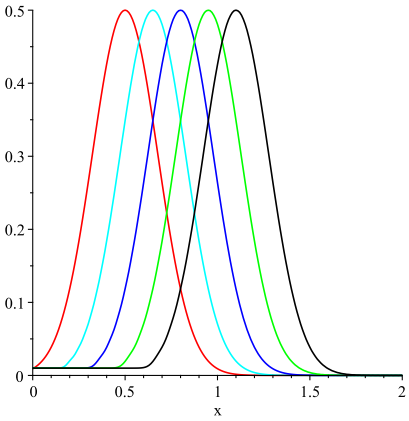
$$pds1 := \textbf{module}()...\textbf{endmodule}$$

```
> pds2 := pdsolve(PDE, IC2, numeric, time = t,
          range = 0..2, spacestep = 0.01,
          timestep = 0.01);
```

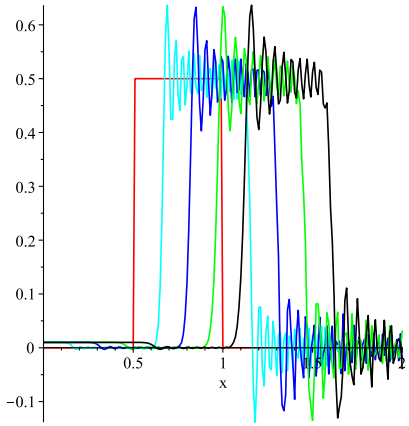
$$pds2 := \textbf{module}()...\textbf{endmodule}$$

```
> with(plots):
> p1 := pds1:-plot(t = 0, color = red):
  p2 := pds1:-plot(t = 0.3, color = cyan):
  p3 := pds1:-plot(t = 0.6, color = blue):
```

```
  p4 := pds1:-plot(t = 0.9, color = green):
  p5 := pds1:-plot(t = 1.2, color = black):
> display(p1, p2, p3, p4, p5);
```




```
> p1 := pds2:-plot(t = 0, color = red):
  p2 := pds2:-plot(t = 0.3, color = cyan):
  p3 := pds2:-plot(t = 0.6, color = blue):
  p4 := pds2:-plot(t = 0.9, color = green):
  p5 := pds2:-plot(t = 1.2, color = black):
> display(p1, p2, p3, p4, p5);
```



Задачи

7.1. Решить следующие дифференциальные уравнения:

$$y' + y\cos x = e^{-\sin x}, \quad y' + y^2 - y\sin 2x - \cos 2x = 0,$$
$$y' + y^2\sin x = \frac{2\sin x}{\cos^2 x}, \quad y'' - y'\operatorname{ctg} x + y\sin^2 x = 0.$$

7.2. Решить задачу Коши и проверить полученное решение:

$$x^2y' = 2xy + 3,\ y(1) = 1, \quad y' - \tfrac{3y}{x} = x^3 + x, y(1) = 2,$$
$$\begin{cases} \dot{x} = -2x + 4y \\ \dot{y} = -x + 3y \end{cases},\ x(0) = 3,\ y(0) = 0,$$
$$\begin{cases} \dot{x} = 2x - 5y + 3 \\ \dot{y} = 5x - 6y + 1 \end{cases},\ x(0) = 6,\ y(0) = 5.$$

7.3. Построить фазовый портрет следующих систем:

$$\begin{cases} \dot{x} = -x \\ \dot{y} = 2x - 2y \end{cases} \qquad \ddot{x} - 2\dot{x}(1 - x^2) + x = 0.$$

7.4. Построить численное решение уравнения теплопроводности на ограниченном интервале:

$$\begin{cases} \dfrac{\partial T}{\partial t} = \dfrac{\partial^2 T}{\partial x^2},\ x \in [-1, 1] \\ T(x, 0) = 1 \\ T(-1, t) = 0 \\ T(1, t) = 0 \end{cases}$$

7.5. Найти общее решение следующих уравнений:

$$(x + 2y)\frac{\partial u}{\partial x} - y\frac{\partial u}{\partial y} = 0, \quad x\frac{\partial f}{\partial x} + 2y\frac{\partial f}{\partial y} = yx^2 + f,$$
$$x\frac{\partial u}{\partial x} + y\frac{\partial u}{\partial y} + z\frac{\partial u}{\partial z} = 0, \quad (g - x)^2\frac{\partial g}{\partial x} + xg\frac{\partial g}{\partial y} = xy.$$

8. Решение различных задач в Maple

8.1. Качение шара по плоскости [7]

Рассмотрим качение уравновешенного шара радиуса R и массой m по плоскости[3]. Запишем в Maple эти параметры и присвоим им единичные значения, а также введем переменную `Inertia`, в которой будет храниться тензор инерции рассматриваемого шара:

```
> restart;
> with(plots):
> R := 1: m := 1:
> Inertia := Matrix([[3, 0, 0], [0, 2, 0], [0, 0, 4]]):
```

Для описания движения шара необходимо определить неподвижную $Oxyz$ и подвижную $Cx_1x_2x_3$, связанную с шаром, системы координат. Указанные системы координат выберем как показано на рис. 1.

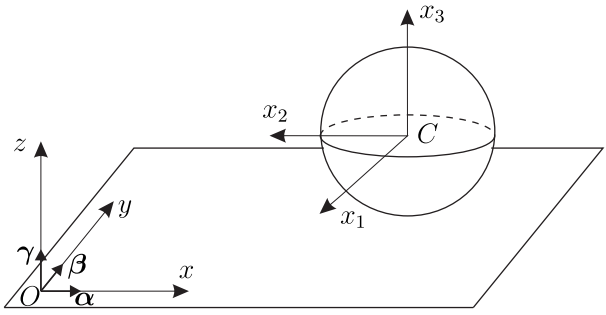


Рис. 1. Шар на плоскости

На рисунке 1 введены векторы $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, которые обозначают орты неподвижных осей, спроецированные на подвижные оси Cx_1, Cx_2, Cx_3, и характеризуют ориентацию шара. Вектор, определяющий центр шара, в выбранной неподвижной системе координат задается как $(x,y,R)^T$.

[3] Приведенный в этом разделе Maple-код можно найти по ссылке: https://drive.google.com/file/d/1wfB-p0QoeaUVBnKgHbvAc0F0FH6yXB58/view?usp=sharing

Нарисуем шар и опорную плоскость в начальный момент времени, для этого создадим переменные, хранящие в себе орты $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$ и переменную с координатами центра шара:

```
> centr := [0, 0, R]:
> alpha0 := [1, 0, 0]:
  beta0 := [0, 1, 0]:
  gamm0 := [0, 0, 1]:
```

Команды для построения шара (полупрозрачным, серым цветом) и плоскости $z = 0$:

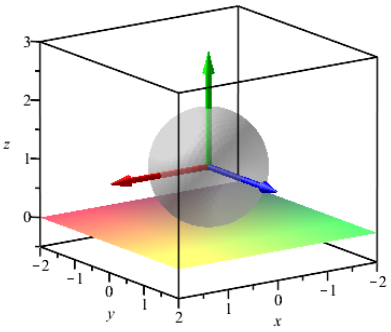
```
> sphere := implicitplot3d(x^2 + y^2 + (z - R)^2 = R^2,
          x = -2..2, y = -2..2, z = -0.5..2,
          grid = [50, 50, 50], transparency = 0.5,
          color = gray):
> plane := implicitplot3d(z = 0, x = -2..2,
          y = -2..2, z = -0.5..2):
```

Команды для построения ортов $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$:

```
> a1 := arrow(centr, 2 * R * alpha0, color = red):
  a2 := arrow(centr, 2 * R * beta0, color = blue):
  a3 := arrow(centr, 2 * R * gamm0, color = green):
```

Визуализация рассматриваемой системы в начальный момент времени:

```
> display(plane, sphere, a1, a2, a3);
```



Уравнения, описывающие движение шара по инерции на плоскости, имеют следующий вид:

$$\dot{\boldsymbol{\alpha}} = \boldsymbol{\alpha} \times \boldsymbol{\Omega}, \quad \dot{\boldsymbol{\beta}} = \boldsymbol{\beta} \times \boldsymbol{\Omega}, \quad \dot{\boldsymbol{\gamma}} = \boldsymbol{\gamma} \times \boldsymbol{\Omega},$$
$$\dot{x} = R(\boldsymbol{\beta}, \boldsymbol{\Omega}), \quad \dot{y} = -R(\boldsymbol{\alpha}, \boldsymbol{\Omega}),$$
$$(\mathbf{I}\boldsymbol{\Omega})^{\cdot} + m\boldsymbol{r} \times (\dot{\boldsymbol{\Omega}} \times \boldsymbol{r}) = (\mathbf{I}\boldsymbol{\Omega}) \times \boldsymbol{\Omega},$$

где $\boldsymbol{r} = -R\boldsymbol{\gamma}$ — радиус-вектор из центра шара в точку контакта, $\mathbf{I}$ — тензор инерции (записан в переменную `Inertia`), x, y — координаты точки контакта в неподвижной системе координат, $\boldsymbol{\Omega} = (\omega_1, \omega_2, \omega_3)^T$ — вектор угловой скорости шара.

Запишем уравнения движения шара в Maple с использованием команд линейной алгебры:

```
> alpha := Vector([alpha1(t), alpha2(t), alpha3(t)]):
> beta := Vector([beta1(t), beta2(t), beta3(t)]):
> gamm := Vector([gamma1(t), gamma2(t), gamma3(t)]):
> Omega := Vector([omega1(t), omega2(t), omega3(t)]):
> r := -R * gamm:
> with(LinearAlgebra):
> equations := seq(diff(alpha[i], t) =
    (CrossProduct(alpha, Omega))[i], i = 1..3),
  seq(diff(beta[i], t) =
    (CrossProduct(beta, Omega))[i], i = 1..3),
  seq(diff(gamm[i], t) =
    (CrossProduct(gamm, Omega))[i], i = 1..3),
  diff(x(t), t) =
    R * DotProduct(beta, Omega, conjugate = false),
  diff(y(t), t) =
    -R * DotProduct(alpha, Omega, conjugate = false),
  seq(diff((Multiply(Inertia, Omega))[i], t) +
  m*CrossProduct(r,CrossProduct(diff(Omega,t),r))[i] =
    CrossProduct(Multiply(Inertia, Omega), Omega)[i],
    i = 1..3):
```

Будем решать эту систему уравнений численно, для этого нужно задать начальные условия:

```
> ics := seq(subs(t=0, alpha[i]) = alpha0[i], i=1..3),
```

```
seq(subs(t=0, beta[i]) = beta0[i], i=1..3),
seq(subs(t=0, gamm[i]) = gamm0[i], i=1..3),
x(0) = 0, y(0) = 0,
omega1(0) = 0, omega2(0) = 3, omega3(0) = 3:
```

Численное решение системы дифференциальных уравнений:

```
> dsol := dsolve({equations, ics},
                 numeric, output = listprocedure):
```

Извлечем фазовую переменную из переменной `dsol`:

```
> Alpha1 := eval(alpha1(t), dsol):
  Alpha2 := eval(alpha2(t), dsol):
  Alpha3 := eval(alpha3(t), dsol):
> Beta1 := eval(beta1(t), dsol):
  Beta2 := eval(beta2(t), dsol):
  Beta3 := eval(beta3(t), dsol):
> Gamma1 := eval(gamma1(t), dsol);
  Gamma2 := eval(gamma2(t), dsol);
  Gamma3 := eval(gamma3(t), dsol);
```

$$\Gamma 1 := \mathbf{proc}(t)...\mathbf{end\ proc}$$
$$\Gamma 2 := \mathbf{proc}(t)...\mathbf{end\ proc}$$
$$\Gamma 3 := \mathbf{proc}(t)...\mathbf{end\ proc}$$

```
> X := eval(x(t), dsol);
  Y := eval(y(t), dsol);
```

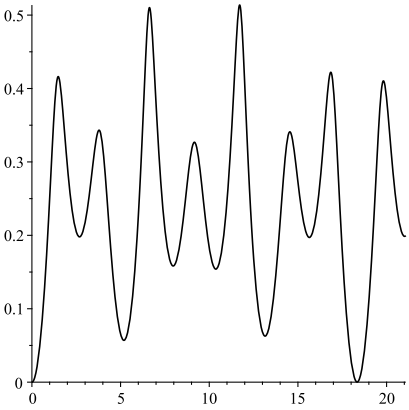
$$X := \mathbf{proc}(t)...\mathbf{end\ proc}$$
$$Y := \mathbf{proc}(t)...\mathbf{end\ proc}$$

```
> Omega1 := eval(omega1(t), dsol);
  Omega2 := eval(omega2(t), dsol);
  Omega3 := eval(omega3(t), dsol);
```

$$\Omega 1 := \mathbf{proc}(t)...\mathbf{end\ proc}$$
$$\Omega 2 := \mathbf{proc}(t)...\mathbf{end\ proc}$$
$$\Omega 3 := \mathbf{proc}(t)...\mathbf{end\ proc}$$

Траектория точки контакта имеет вид:

```
> plot([X(t), Y(t), t = 0..10], color = black);
```

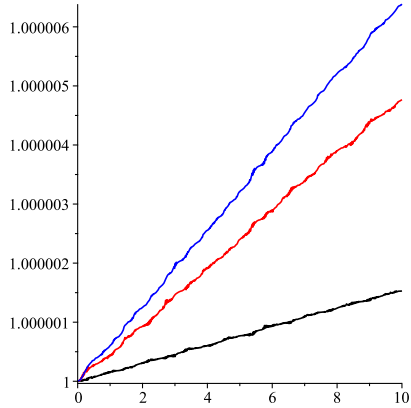


Уравнения, описывающие движения шара, имеют следующие геометрические интегралы:

$$(\boldsymbol{\alpha},\boldsymbol{\alpha}) = 1, \quad (\boldsymbol{\beta},\boldsymbol{\beta}) = 1, \quad (\boldsymbol{\gamma},\boldsymbol{\gamma}) = 1.$$

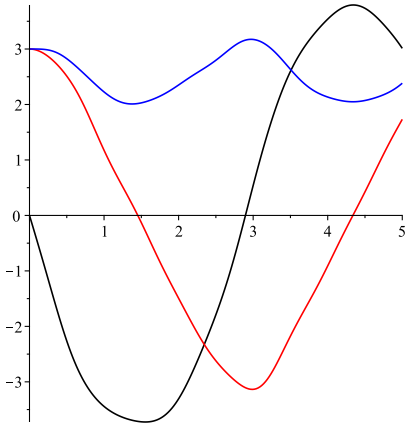
Построим графики изменения этих интегралов с течением времени, чтобы убедиться в корректности решения уравнений:

```
> plot([
  [t,abs(Gamma1(t)^2+Gamma2(t)^2+Gamma3(t)^2),t=0..10],
  [t,abs(Beta1(t)^2+Beta2(t)^2+Beta3(t)^2),t=0..10],
  [t,abs(Alpha1(t)^2+Alpha2(t)^2+Alpha3(t)^2),t=0..10]],
  color = [black, red, blue]);
```



Построение зависимостей угловых скоростей от времени:

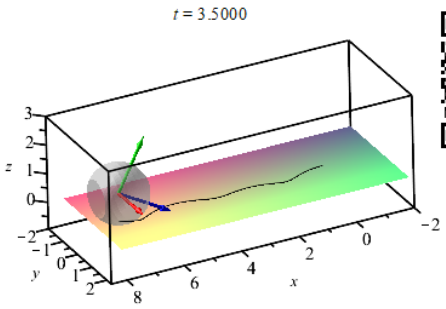
```
> plot([[t, Omega1(t), t = 0..5],
        [t, Omega2(t), t = 0..5],
        [t, Omega3(t), t = 0..5]],
        color = [black, red, blue]);
```



Построение анимации движения:

```
> T := 3.5:
> A_sphere := animate(implicitplot3d,
          [(x - X(t))^2 + (y - Y(t))^2 + (z - R)^2 = R^2,
          x = -2..8, y = -2..2, z = -0.5..2,
          grid = [50, 50, 50], transparency = 0.5,
          color = gray], t = 0..T):
> trajectory := animate(spacecurve, [[X(t), Y(t), 0],
          t = 0..time, color = black], time = 0..T):
> plane := implicitplot3d(z = 0, x = -2..2,
          y = -2..2, z = -0.5..2):
> a1 := animate(arrow, [[X(t), Y(t), 0] + centr,
        2 * R * [Alpha1(t), Beta1(t), Gamma1(t)],
        color = red], t = 0..T):
  a2 := animate(arrow, [[X(t), Y(t), 0] + centr,
        2 * R * [Alpha2(t), Beta2(t), Gamma2(t)],
        color = blue], t = 0..T):
```

```
    a3 := animate(arrow, [[X(t), Y(t), 0] + centr,
          2 * R * [Alpha3(t), Beta3(t), Gamma3(t)],
          color = green], t = 0..T):
> display(plane, trajectory, A_sphere, a1, a2, a3);
```



8.2. Отображение Пуанкаре

Отображение Пуанкаре является важным инструментом исследования динамических систем. Рассмотрим процедуру построения отображения Пуанкаре[4] на примере следующей системы дифференциальных уравнений:

$$\dot{x} = v, \quad \dot{v} = -bv - \sin x + \sin(\Omega t + p),$$

где Ω, p и b — некоторые постоянные. Запишем эти уравнения:

```
> restart;
> with(plots):
> eqs := diff(x(t), t) = v(t),
      diff(v(t), t) = -b*v(t)-sin(x(t))+sin(Omega*t+p);
```

Определим константы Ω, p и b для дальнейшего численного решения этой системы уравнений:

```
> Omega := 0.8; b := 0.05: p := 0.1;
```

$$\Omega := 0.8$$

$$p := 0.1$$

[4] Приведенный в этом разделе Maple-код можно найти по ссылке: https://drive.google.com/file/d/1f3Lm2kKvAuFUZnnytMDOr6z2Qu7VDcjP/view?usp=sharing

Найдем численное решение системы дифференциальных уравнений и извлечем его компоненты в отдельные процедуры:

```
> dsol := dsolve({eqs, v(0) = -2.5, x(0) = 0},
                 numeric, output = listprocedure,
                 maxfun = 5000000):
> X := eval(x(t), dsol);
> V := eval(v(t), dsol);
```

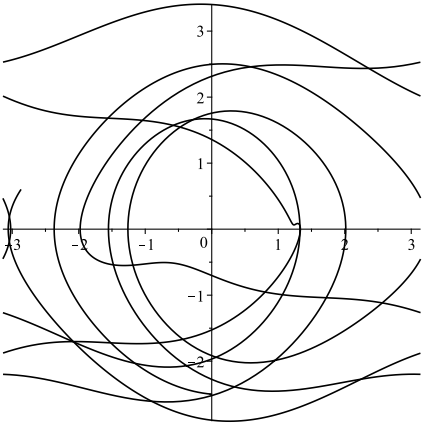
Поскольку фазовая переменная $x(t)$ является периодической, то создадим функцию, которая «переодизует» $x(t)$ в пределах $[-\pi, \pi]$:

```
> X_period := t -> X(t) - 2*Pi*floor((Pi+X(t))/(2*Pi));
```

$$X\_period := t \mapsto X(t) - 2\pi \left\lfloor \frac{\pi + X(t)}{2\pi} \right\rfloor$$

Рассматриваемая система дифференциальных уравнений неавтономна, следовательно, ее траектории на плоскости (x, v) могут самопересекаться. Поэтому исследование системы с помощью фазового портрета оказывается неинформативным. Построение фазовой траектории на плоскости (x, v):

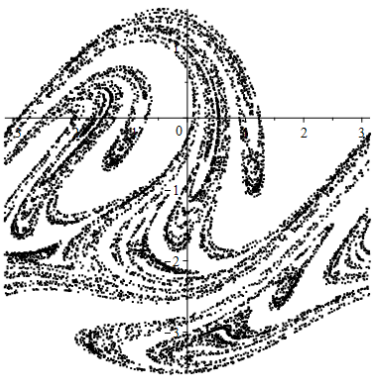
```
> plot([X_period(t), V(t), t = 0..50],
                      discont = true, color = black);
```



Для анализа таких систем традиционно используется метод отображения Пуанкаре, который заключается в том, чтобы

фиксировать значения переменных $x(t)$ и $v(t)$ при прохождении независимой переменной t значений $T = \frac{2\pi i}{\Omega}$. Цикл для построения точек отображения (N задает количество точек):

```
> N := 10000:
> for i from 1 to N do
  tau := 2 * Pi * i / Omega;
  A[i] := pointplot([X_period(tau), V(tau)],
  color = black, symbol = point, symbolsize = 20);
  end do:
> display(seq(A[l], l = 1..N));
```



8.3. Анализ уравнения Матье [8]

Уравнением Матье называется дифференциальное уравнение второго порядка вида[5]:

$$\ddot{y} + (\kappa + \delta \cos t)y = 0, \quad \kappa = \text{const}, \quad \delta = \text{const}.$$

Для анализа это уравнение удобнее представить в виде системы уравнений первого порядка:

$$\begin{cases} \dot{y} = v, \\ \dot{v} = -(\kappa + \delta \cos t)y. \end{cases}$$

[5] Приведенный в этом разделе Maple-код можно найти по ссылке: https://drive.google.com/file/d/1tvhGGt4BzY6VYhzhxR3by7MYaWRTkdwL/view?usp=sharing

Очевидно, что данная система уравнений имеет частное решение

$$v = 0, \quad y = 0.$$

Проанализируем устойчивость данного решения в зависимости от значений параметров κ и δ. Для этого воспользуемся теорией устойчивости систем с периодическими коэффициентами.

Запишем в Maple рассматриваемую систему уравнений и вычислим ее якобиан $\mathbf{J}$ в точке, соответствующей частному решению:

```
> restart;
> with(plots):
> eq := diff(y(t), t) = v(t),
   diff(v(t), t) = -(kappa + delta * cos(t)) * y(t):
> with(VectorCalculus):
> with(LinearAlgebra):
> J := Jacobian([v, -(kappa + delta * cos(t)) * y],
                [y, v] = [0, 0]);
```

$$J := \begin{bmatrix} 0 & 1 \\ -\kappa - \delta\cos(t) & 0 \end{bmatrix}$$

Для оценки устойчивости решения необходимо построить матрицу монодромии, которая вычисляется как фундаментальная матрица решении $\mathbf{Q}$ в момент времени $t = 2\pi$ (для рассматриваемой задачи). Уравнение для определения фундаментальной матрицы решений имеет вид:

$$\dot{\mathbf{Q}} = \mathbf{J}\mathbf{Q}, \quad \mathbf{Q}(0) = \mathrm{E},$$

где E — единичная матрица.

Зададим некоторые значения параметров κ, δ в Maple и рассчитаем для них матрицу фундаментальных решений:

```
> subParam := {kappa = 1, delta = 0.5}:
> Q := Matrix([[q11(t), q12(t)], [q21(t), q22(t)]]);
```

$$Q := \begin{bmatrix} q11(t) & q12(t) \\ q21(t) & q22(t) \end{bmatrix}$$

```
> dQ := map(diff, Q, t);
```

$$dQ := \begin{bmatrix} \frac{\mathrm{d}}{\mathrm{d}t} q11(t) & \frac{\mathrm{d}}{\mathrm{d}t} q12(t) \\ \frac{\mathrm{d}}{\mathrm{d}t} q21(t) & \frac{\mathrm{d}}{\mathrm{d}t} q22(t) \end{bmatrix}$$

```
> sys := seq(seq(dQ[i, j] = subs(subParam,
         Multiply(J, Q)[i, j]), i = 1..2), j = 1..2);
```

$$sys := \frac{\mathrm{d}}{\mathrm{d}t} q11(t) = q21(t), \frac{\mathrm{d}}{\mathrm{d}t} q21(t) = (-1 - 0.5\cos(t))q11(t),$$
$$\frac{\mathrm{d}}{\mathrm{d}t} q12(t) = q22(t), \frac{\mathrm{d}}{\mathrm{d}t} q22(t) = (-1 - 0.5\cos(t))q12(t)$$

```
> ics := q11(0) = 1, q12(0) = 0, q21(0) = 0, q22(0) = 1:
> dsol := dsolve({sys, ics},
          numeric, output = listprocedure):
> Q11 := eval(q11(t), dsol); Q12 := eval(q12(t), dsol);
  Q21 := eval(q21(t), dsol); Q22 := eval(q22(t), dsol);
```

$$Q11 := \mathbf{proc}(t)...\mathbf{end\ proc}$$
$$Q12 := \mathbf{proc}(t)...\mathbf{end\ proc}$$
$$Q21 := \mathbf{proc}(t)...\mathbf{end\ proc}$$
$$Q22 := \mathbf{proc}(t)...\mathbf{end\ proc}$$

Матрица монодромии состоит из следующих значений:

```
> T0 := 2 * Pi:
> Monondromy := Matrix([[Q11(T0), Q12(T0)],
                        [Q21(T0), Q22(T0)]]);
```

$$Monondromy := \begin{bmatrix} 1.01065766237453 & 0.0474637633434652 \\ 0.451459363351136 & 1.01065754886783 \end{bmatrix}$$

Согласно теории Флоке, решение $y = 0$, $v = 0$ будет устойчивым, если след матрицы монодромии будет по модулю меньше двух. Вычислим это значение:

```
> abs(Trace(Monondromy));
```

$$2.02131521124237$$

Получается, что при $\kappa = 1$ и $\delta = 0.5$ частное решение будет неустойчивым.

Оформим вышеприведенный код в процедуру, которая принимает на вход значения параметров κ и δ и возвращает модуль следа матрицы монодромии:

```
> Lambda := proc(kappa, delta)
  local T, J, Q, dQ, sys, j, i, ics, dsol,
       Q11, Q12, Q21, Q22, Monondromy;
  T := 2*Pi;
  J := Jacobian([v, -(kappa + delta * cos(t)) * y],
       [y, v] = [0, 0]);
  Q := Matrix([[q11(t), q12(t)], [q21(t), q22(t)]]);
  dQ := map(diff, Q, t);
  sys := seq(seq(dQ[i, j] =
        Multiply(J, Q)[i, j], i = 1..2), j = 1..2);
  ics :=q11(0) = 1,q12(0) = 0,q21(0) = 0,q22(0) = 1;
  dsol := dsolve({sys, ics}, numeric,
        output = listprocedure);
  Q11 :=eval(q11(t), dsol); Q12 :=eval(q12(t), dsol);
  Q21 :=eval(q21(t), dsol); Q22 :=eval(q22(t), dsol);
  Monondromy := Matrix([[Q11(T), Q12(T)],
        [Q21(T), Q22(T)]]);
  return abs(Trace(Monondromy));
  end proc:
```

Проверим корректность работы этой процедуры:

```
> Lambda(1, 0.5);
```

$$2.02131521124237$$

```
> Lambda(3, 0.5);
```

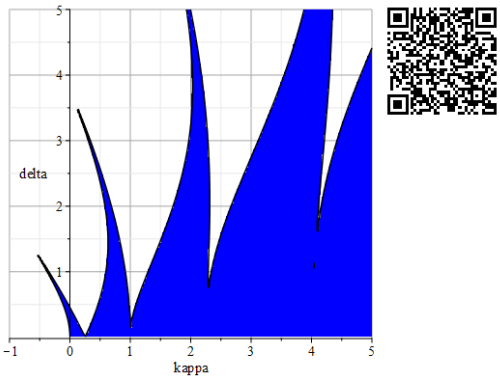
$$0.266386148280301$$

```
> Lambda(0, 2);
```

$$52.0851018464673$$

Построим область устойчивости частного решения с помощью функции `inequal` из пакета `plots`:

```
> inequal('Lambda(kappa, delta) <= 2', kappa = -1..5,
      delta = 0.01..5, grid = [150, 150], color = blue);
```



Рассмотренный в этом разделе анализ устойчивости может быть использован при анализе устойчивости такой системы, как маятник с колеблющейся точкой подвеса (маятник Капицы [9]).

Задачи

8.1. Построить анимацию, визуализирующую эффект Джанибекова.

8.2. Построить уравнения движения двойного маятника без колебаний точки подвеса. Проанализировать полученные уравнения с помощью отображения Пуанкаре.

8.3. Проанализировать систему, описывающую движения маятника с колеблющейся точкой подвеса (маятник Капицы).

8.4. Математическим бильярдом называют замкнутую систему, ограниченную бортами произвольной формы без луз. По этому столу без трения движется точечный шар, абсолютно упруго отражающийся от бортов. Построить анимацию бильярда в эллипсе (для одного шара, для N шаров).

Список команд

1) `Vector` — объявление вектора.
2) `DotProduct`, `dotprod`, `innerprod` — вычисление скалярного/внутреннего произведения векторов.
3) `CrossProduct`, `crossprod` — вычисление векторного произведения векторов.
4) `Norm`, `norm` — вычисление нормы.
5) `Matrix` — объявление матрицы.
6) `Multiply`, `multiply` — произведение матриц.
7) `Transpose`, `transpose` — транспонирование матрицы.
8) `HermitianTranspose` — эрмитово транспонирование матрицы.
9) `Determinant`, `det` — вычисление определителя матрицы.
10) `MatrixInverse`, `inverse` — вычисление обратной матрицы.
11) `Rank`, `rank` — вычисление ранга матрицы.
12) `Eigenvalues`, `eigenvalues` — вычисление собственных чисел матрицы.
13) `Eigenvectors`, `eigenvectors` — вычисление собственных векторов матрицы.
14) `JordanForm` — определение жордановой формы матрицы.
15) `Gradient`, `grad` — вычисление градиента функции.
16) `diverge` — вычисление дивергенции вектора.
17) `curl` — вычисление ротора векторного поля.
18) `Jacobian`, `jacobian` — вычисление якобиана.

19) `Hessian`, `hessian` — построение матрицы Гессе (гессиана).

20) `RandomVector` — создание вектора, заполненного случайными числами.

21) `RandomMatrix` — создание матрицы, заполненной случайными числами.

22) `MatrixExponential`, `exponential` — вычисление матричной экспоненты.

23) `CharacteristicPolynomial`, `charpoly` — построение характеристического полинома матрицы.

24) `GramSchmidt` — ортогонализация векторов.

25) `dsolve` — решение обыкновенных дифференциальных уравнений.

26) `odetest` — проверка решения обыкновенного дифференциального уравнения.

27) `odeplot` — построение численных решении обыкновенных дифференциальных уравнений.

28) `DEplot` — построение поле скорости системы дифференциальных уравнений.

29) `pdsolve` — решение дифференциальных уравнений в частных производных.

30) `pdetest` — проверка решения дифференциального уравнения в частных производных.

Список литературы

[1] Ветчанин Е. В., Артемова Е. М. Программирование в системах GNU Octave и MatLAB: учеб.-метод. пособие. — Ижевск: Изд-во «Удмуртский университет», 2019. — 99 с.

[2] Ветчанин Е. В., Артемова Е. М. Процедурное программирование на языках C/C++: учеб.-метод. пособие. — Ижевск: Изд-во «Удмуртский университет», 2020. — 124 с.

[3] Лебедев, В. Г., Иванова, Т. Б., Васькин, В. В., Пивоварова, Е. Н. Методы математической физики. Параболические уравнения: учеб.-метод. пособие. — Ижевск: Изд-во «Удмуртский университет», 2022. — 123 с.

[4] Лебедев В. Г., Иванова Т. Б., Васькин В. В. Уравнения теплопереноса: учеб.-метод. пособие. — Ижевск: Изд-во «Удмуртский университет», 2018. — 104 с.

[5] Килин А. А. Введение в теорию точечных отображений. Динамический хаос: учеб. пособие. — Ижевск: Изд-во «Удмуртский университет», 2021. — 100 с.

[6] Вержбицкий В. М. Основы численных методов: учебник для вузов. — М.: Изд-во «Высшая школа», 2002. — 840 с.

[7] Чаплыгин С. А. О катании шара по горизонтальной плоскости // Матем. сб., – 1903. – Т.24. – С. 139–168. (См. также: Чаплыгин С. А. Собр. соч.: Т. 1. М.–Л.: ОГИЗ, 1948. С.76–101.)

[8] Холостова О. В. Задачи динамики твердых тел с вибрирующим подвесом. — Издательство «Институт компьютерных исследований», 2016. — 308 с.

[9] Капица П. Л. Маятник с вибрирующим подвесом // Успехи физических наук. – 1951. – Т.44. – №5. – С. 7-20.

Учебное издание

Артемова Елизавета Марковна
Бизяев Иван Алексеевич

СИСТЕМА КОМПЬЮТЕРНОЙ МАТЕМАТИКИ MAPLE: ТЕОРИЯ И РЕШЕНИЕ ЗАДАЧ

Часть 2

Учебно-методическое пособие

Авторская редакция

Подписано в печать 30.06.2023. Формат 60x84 1/16.
Усл. печ. л. 5.29. Уч. изд. л. 4.91.
Тираж 33 экз. Заказ № 1132.

Издательский центр «Удмуртский университет»
426034, г. Ижевск, Ломоносова, 4Б, каб. 021
Тел.: + 7 (3412) 916-364, E-mail: editorial@udsu.ru

Типография Издательского центра «Удмуртский университет»
426034, Ижевск, ул. Университетская, 1, корп. 2.
Тел. 68-57-18