ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Корольков О. Г. и др.

Maple: теория и практика

(незавершённая версия)

Учебное пособие

к спецкурсу по специальности 010500 (510200) –
Прикладная математика и информатика

ВОРОНЕЖ 2006

Утверждено научно-методическим советом
факультета прикладной математики, информатики и механики
протокол №4 от 12 декабря 2005 года

Авторы: Корольков О. Г. и др.

Учебное пособие подготовлено на кафедре вычислительной математики
факультета прикладной математики, информатики и механики
Воронежского государственного университета.
Рекомендуется для студентов 3 курса д/о факультета ПММ.

Содержание

Введение 4
1. Знакомство с **Maple** 4
Инертные формы функций 5
2. Типы данных, операторы и функции системы **Maple** 6
Выражения и основы работы над ними 6
Контроль за типами объектов 7
Простые типы данных 8
Данные множественного типа 9
Константы 11
Переменные 11
Операторы и операнды 12
3. Средства программирования системы **Maple** . . 13
Условный оператор 13
Операторы цикла 14
Процедуры 15
Создание библиотеки процедур 16
Работа с файлами 16
4. Средства графики системы **Maple** 17
Двумерная графика 17
Трёхмерная графика 18
Расширенные средства графики 19
5. Работа с полиномами 20
Оценка степени полинома и его коэффициентов . . 20
Вычисление корней полинома 20
6. Решение уравнений и неравенств 21
Решение уравнений 21
Решение неравенств 22
Решение уравнений в численном виде 22
Решение уравнений в целых числах 24
Решение рекуррентных уравнений 24
7. Интерполирование функций 24
Полиномиальная интерполяция 24
Интерполяция сплайнами 26
Общая задача интерполирования 27
8. Дифференцирование функций 28
Вычисление пределов функций 28
Исследование функций на непрерывность . . . 28
Вычисление производных функций 29
Замена переменных 31

Введение

В 1980 году группа исследователей канадского университета Waterloo занялась проблемой создания компьютерной системы, эффективной в решении алгебраических задач и достаточно простой для того, чтобы её могли использовать не только математики и инженеры, но и студенты. Программа получила имя **Maple**. В начале 90-х у Maple появился графический интерфейс пользователя, и именно с этого времени Maple стал широко применяться в образовании.
Waterloo Maple, наряду с Wolfram Mathematica, является мощнейшей системой символьной математики (или компьютерной алгебры). Даже в тех случаях, когда вычисления носят символьный характер, расчётные алгоритмы реализуются так, чтобы получить сначала аналитический результат.
В Maple в общей сложности используется более трёх тысяч команд, однако некоторые из них (относящиеся к проблемам интегрирования, дифференцирования функций, решения уравнений и т.п.) применяются достаточно часто и составляют костяк базового языка. Некоторые команды доступны только при подключении специальных пакетов.
Данное пособие является практическим руководством и ориентировано на описание командного языка Maple. Работа с пособием предполагает знание, хотя бы в минимальном объёме, основ высшей математики. В пособии приведено много разнообразных примеров решения задач линейной алгебры, математического анализа, дифференциальных уравнений, математической физики и численных методов.
В Maple предусмотрена мощная и эффективная система справки. Справку по любой команде можно получить, наведя курсор на эту команду и затем нажав <F1>. Ещё один способ – в области ввода ввести знак вопроса и, без пробела, название интересующей пользователя команды, после чего нажать <Enter>.

1. Знакомство с Maple

При запуске Maple автоматически создаётся новый документ, который в дальнейшем будем называть рабочим листом. В области ввода пользователь видит символ начала командной строки (>) – здесь осуществляется ввод команд. Результат выполнения команд отображается в том же рабочем листе, в области вывода. Если команда заканчивается точкой с запятой (;), то результат выполнения появится в ячейке вывода, если же в конце команды поставить двоеточие (:), результат выводиться не будет.
Ввод фиксируется нажатием клавиши <Enter>. Если надо перенести ввод на новую строку, используется комбинация клавиш <Shift> + <Enter>.

В системной переменной % хранится результат предыдущей операции. С помощью одного, двух или трёх знаков % можно вызвать первое, второе или третье выражение с конца сессии.

Для присваивания переменной значения используется оператор `:=`. Maple сохраняет в памяти все определения и присваивания, которые были сделаны во всех загруженных в систему документах. Для очистки внутренней памяти Maple используют команду `restart`. В примерах пособия эта команда не используется, однако нужно иметь в виду, что решение каждой новой задачи следует начинать с команды `restart`.

Лабораторная работа №1.
Знакомство с Maple.

Для вычисления определённого интеграла используется функция `int`:

```
> int(2 * exp(-x) * sin(x), x = 0 .. infinity);
```

$$1$$

Рассмотрим последовательность команд:

```
> x := 2 * exp(-1) * sin(1);
```

$$x := e^{(-1)} \sin(1)$$

```
> int(2 * exp(-x) * sin(x), x = 0 .. infinity);
Error, (in int) wrong number (or type) of arguments
```

Где допущена ошибка? Как её исправить?

Инертные формы функций

Сравните две следующие команды:

```
> int(2 * exp(-x) * sin(x), x = 0 .. infinity);
```

$$1$$

```
> Int(2 * exp(-x) * sin(x), x = 0 .. infinity);
```

$$\int_0^{\infty} 2e^{(-x)} \sin(x) dx$$

`Int` – это так называемая инертная форма функции `int`. Имена инертных функций начинаются с большой буквы, эти функции выводят выражение в естественном математическом виде. С помощью ряда функций, например `evalf`, можно вычислить выражение, полученное инертной функцией:

```
> Limit(sin(x) / x, x = 0);
```

$$\lim_{x \to 0} \frac{\sin(x)}{x}$$

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

$$1.000000000$$

Иногда используют следующую запись:

```
> Limit((1 + 1 / n) ^ n, n = infinity) =
  limit((1 + 1 / n) ^ n, n = infinity);
```

$$\lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n = e$$

2. Типы данных, операторы и функции системы Maple

Выражения и основы работы над ними

Пользовательский интерфейс системы Maple позволяет представлять как вводимые, так и выводимые выражения в самых различных формах, в том числе и в естественном математическом виде. В данном пособии выражения записываются на Maple-языке без использования специальных средств для их представления. При наборе математических выражений удобно пользоваться палитрами математических символов (меню **View à Palettes**).

Для упрощения выражений используют функцию `simplify`, а для их расширения – функцию `expand`:

```
> simplify(sin(x)^2 + cos(x)^2);
```

$$1$$

```
> expand(sin(2 * x));
```

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

Для разложения выражения на множители используют функцию `factor`, а для комплектования по степеням – функцию `collect`:

```
> factor(x^2 + 2 * x * y + y^2);
```

$$(x+y)^2$$

```
> factor(x^2 + 2 * x + 2, complex);
```

$$(x+1.000000000+1.000000000\,I)(x+1.-1.\,I)$$

```
> g := int(x * (exp(x) - exp(-x)), x);
```

$$g := xe^x - e^x + \frac{x}{e^x} + \frac{1}{e^x}$$

```
> collect(g, exp(x));
```

$$g := (x-1)e^x + \frac{x+1}{e^x}$$

Для преобразования выражений в тождественные формы используют функцию `convert`:

```
> convert(2 * sin(I * x) + 2 * sinh(x), exp);
```

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

```
> collect(%, exp);
```

$$(1+I)e^x - \frac{1+I}{e^x}$$

```
> convert(arcsinh(x), ln);
```

$$\ln\left(x+\sqrt{x^2+1}\right)$$

Некоторые параметры функции `convert` представлены в **Приложении 1**.

Для создания последовательностей выражений используют функцию seq:

```
> seq(sin(x), x = 0 .. 5);
```

$$0, \sin(1), \sin(2), \sin(3), \sin(4), \sin(5)$$

Для оценивания выражений различного типа существует группа функций, основные из которых представлены в **Приложении 1**.

```
> eval(x^2 – x * y + y^2, [x = 1/3, y = 1/6]);
```

$$\frac{1}{12}$$

```
> evalf(pi, 20);
```

$$3.1415926535897932385$$

```
> evalr(exp(INTERVAL(0 .. ln(2))));
```

$$INTERVAL(1..2)$$

Лабораторная работа №2.
Оценивание выражений.

С помощью функции shake оценим интервал, в пределах которого находится число p :

```
> shake(Pi, 5);
```

$$INTERVAL(3.141279..3.141907)$$

Получите такой же результат, используя функцию evalr.

Обоими способами оцените интервалы, в пределах которых находятся следующие выражения:

$arctg(2)$;

$e^{(-1)} \sin(1)$;

$\cosh(p) - \sinh(p)$.

Контроль за типами объектов

Функция whattype возвращает тип объекта:

```
> whattype(Pi);
```

symbol

```
> whattype(2 + 3);
```

integer

```
> whattype(sin(x));
```

function

С помощью функции type можно выяснить, относится ли указанный объект к соответствующему типу:

```
> type(Pi, constant);
```

true

```
> type(sin(ln(x)), function);
```

true

Основные типы данных языка Maple перечислены в **Приложении 2**.

Простые типы данных

Maple работает с числами следующего типа: целыми, рациональными, вещественными с плавающей точкой и комплексными (подробный список числовых типов данных можно найти в **Приложении 2**). Указание десятичной точки в числе делает его вещественным и ведёт к переводу вычислений в режим работы с вещественными числами. При этом количеством выводимых после десятичной точки цифр можно управлять, задавая значение системной переменной `Digits`:

```
> exp(-1) * sin(1);
```

$$e^{(-1)}\,sin(1)$$

```
> Digits := 10:
  exp(-1.) * sin(1.);
```

0.3095598757

Мнимая единица обозначается в Maple как `I`.

Для контроля за числами используют уже известные функции `whattype` и `type`:

```
> type(30, numeric);
```

true

```
> type(30, integer);
```

true

```
> type(30.0, integer);
```

false

```
> whattype(30.0);
```

float

```
> type(Pi * I, complex);
```

true

Функция `convert` служит для преобразования чисел с разными основаниями, а также для перевода чисел в разные форматы (рациональная дробь или вещественное число с плавающей точкой):

```
> convert(30, binary);
```

11110

```
> convert(11110, decimal, binary);
```

30

```
> convert(1 / 5, float);
```

0.2000000000

```
> convert(0.2, rational);
```

$$\frac{1}{5}$$

```
> convert(Pi, float);
```

3.141592654

Некоторые параметры функции `convert` описаны в **Приложении 1**.

Для преобразования единиц измерения используют команду **Unit Converter** (меню **Edit à Unit Converter**).

Основные математические функции представлены в **Приложении 3**.

Данные множественного типа

Неупорядоченные наборы – множества – создаются с помощью фигурных скобок. Из множеств автоматически удаляются повторяющиеся элементы.

```
> {a, b, g, a, a, c, c, f};
```

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

```
> {5, 7 - 4, pi, exp(ln(3)), sqrt(25), 3!};
```

$\{3, 5, 6, p\}$

Операторы для работы с данными множественного типа представлены в **Приложении 6**.

```
> {a, b, g, a, a, c, c, f} intersect {a, b, c, d};
```

$\{a, b, c\}$

Упорядоченные наборы – списки – создаются с помощью квадратных скобок:

```
> s1 := [5, 7 - 4, pi, exp(ln(3)), sqrt(25), 3!];
```

$s1 := [5, 3, p, 3, 5, 6]$

```
> s2 := [[a, b], [c, d]];
  s3 := [[1, 2], [2, 5]];
```

$s2 := [[a, b], [c, d]]$

$s3 := [[1, 2], [2, 5]]$

Списки широко применяются для задания векторов и матриц:

```
> V := array(1 .. 6, s1);
```

$V := [5, 3, p, 3, 5, 6]$

```
> M1 := array(1 .. 2, 1 .. 2, s2);
  M2 := convert(s3, matrix);
```

$$M1 := \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$M2 := \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix}$$

Элементы векторов и матриц являются индексированными переменными. Допустимы операции вызова нужного элемента и присваивания ему нового значения.

```
> V[1] := V[3] * V[6] / 3:
  evalm(V);
```

$[2p, 3, p, 3, 5, 6]$

```
> evalm(2 * V);
```

$[4p, 6, 2p, 6, 10, 12]$

```
> evalm(M1 - a * M2);
```

$$\begin{bmatrix} 0 & b-2a \\ c-2a & d-5a \end{bmatrix}$$

```
> evalm(M1 &* M2);
```

$$\begin{bmatrix} a+2b & 2a+5b \\ c+2d & 2c+5d \end{bmatrix}$$

```
> evalm(M2^(-1));
```

$$\begin{bmatrix} 5 & -2 \\ -2 & 1 \end{bmatrix}$$

Функция `map` применяет заданную операцию к каждому элементу матрицы или вектора:

```
> M := convert([[x, x^2], [0, 1 + x]], matrix):
  map(sin, M);
```

$$\begin{bmatrix} \sin(x) & \sin(x^2) \\ 0 & \sin(1+x) \end{bmatrix}$$

```
> map(diff, M, x);
```

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

В ядро Maple введены минимально необходимые средства для работы с векторами и матрицами. Основной упор сделан на пакеты расширения, главным из которых является пакет `linalg`. Некоторые функции из этого пакета представлены в **Приложении 4**.

Для загрузки пакета используют команду `with`.

```
> with(linalg):
  M := matrix(2, 2, [[1, 2], [2, 5]]);
```

$$M := \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix}$$

```
> inverse(M);
```

$$\begin{bmatrix} 5 & -2 \\ -2 & 1 \end{bmatrix}$$

```
> eigenvalues(M);
```

$$3+2\sqrt{2}, 3-2\sqrt{2}$$

```
> LUdecomp(M, L = 'L', U = 'U'):
  evalm(L);
  evalm(U);
```

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

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

```
> equal(M, multiply(L, U));
```

true

Подробную информацию о пакете `linalg` можно получить, исполнив команду `?linalg`.

Лабораторная работа №3.
Решение систем линейных уравнений.

Для решения матричных уравнений используется функция `linsolve`.

Решим систему линейных уравнений $\begin{cases} x_1 + 2x_2 = 1, \\ 2x_1 + 5x_2 = 3. \end{cases}$

Данную систему можно записать в виде $\begin{pmatrix} 1 & 2 \\ 2 & 5 \end{pmatrix} \cdot x = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$.

```
> with(linalg):
  A := matrix(2, 2, [[1, 2], [2, 5]]):
  f := vector(2, [1, 3]):
  linsolve(A, f);
```

$[-1, 1]$

Решите данную систему, не используя функцию `linsolve`.

Пусть теперь: A – матрица Гильберта;
y – случайный вектор;
f – вектор, равный произведению A на y.

Убедитесь, что решение x матричного уравнения $Ax = f$ в точности совпадает с вектором y.

Преобразуйте компоненты матрицы A и вектора f к форме чисел с плавающей точкой, после чего ещё раз найдите решение уравнения $Ax = f$. Исследуйте погрешность полученного решения, изменяя размерность уравнения и значение системной переменной `Digits`.

Константы

Именованные константы Maple перечислены в **Приложении 5**. Подробную информацию о константах можно получить, исполнив команду `?constants`.

```
> constants;
```

false, *g*, ∞, *true*, *Catalan*, *FAIL*, *p*

```
> type(false, constant);
```

true

```
> type(exp(I * Pi), constant);
```

true

Переменные

Для явного указания типа переменных используется конструкция `name::type`. Для присваивания переменной значения используется оператор `:=`. Для отмены присваивания – одна из конструкций `x := 'x'` или `x := evaln(x)`.

Для придания переменным статуса предполагаемых используется функция `assume`.

```
> sqrt(x^2);
  M := matrix([[x, x^2], [exp(x), 1]]):
  map(ln, M);
```

$$\sqrt{x^2}$$

$$\begin{bmatrix} ln(x) & ln(x^2) \\ ln(e^x) & 0 \end{bmatrix}$$

```
> assume(x, positive):
  sqrt(x^2);
  M := matrix([[x, x^2], [exp(x), 1]]):
  map(ln, M);
```

$$x\sim$$

$$\begin{bmatrix} ln(x\sim) & 2ln(x\sim) \\ x\sim & 0 \end{bmatrix}$$

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

Функцию `assume` можно использовать не только для переменных, но и для целых выражений. Некоторые параметры этой функции можно найти в **Приложении 1**.

Операторы и операнды

В Maple имеется три типа операторов: бинарные (`binary`), с одним операндом (`unary`) и без операндов (`nullary`). Все операторы Maple перечислены в **Приложении 6** в порядке их выполнения. Подробную информацию об операторах можно получить, исполнив команду `?operators`.

Для задания функций используется функциональный оператор `->`:

```
> f := x -> exp(-I * x) * sin(x):
  f(Pi / 2);
```

$$-I$$

```
> g := (x, y) -> sqrt(x^2 + y^2):
  g(3, 4);
```

$$5$$

```
> simplify(g(sin(x), cos(x)));
```

$$1$$

Функция `define` служит для создания операторов. Функция `definemore` позволяет добавить свойства к уже определённому функцией `define` оператору.

```
> define(L, linear, L(1) = t):
  L(x^2 + 2 * x + 2);
```

$$L(x^2)+2L(x)+2t$$

```
> define(h, commutative):
  h(x, y) - h(y, x);
```

$$0$$

```
> define(f, diff(f(x), x) = 1 / f(x)):
  diff(f(exp(x)), x);
```

$$\frac{e^x}{f\left(e^x\right)}$$

```
> int(3 * f(x), x);
```

$$f(x)^3$$

Некоторые параметры функции `define` описаны в **Приложении 6**.

Лабораторная работа №4.
Создание операторов.

Создадим оператор, вычисляющий числа Фибоначчи:

```
> define(F, F(0) = 0, F(1) = 1,
  F(n::posint) = F(n - 2) + F(n - 1)):
  F(10);
  F(22);
```

55
17711

С помощью системной функции `time` определим время, затраченное на вычисление значения `F(22)`:

```
> time(F(22));
```

1.515

Это время оказалось довольно значительным, поскольку каждое новое число Фибоначчи вычисляется заново. Создайте оператор, вычисляющий числа Фибоначчи без использования рекурсии.

3. Средства программирования в Maple

Условный оператор

Условный оператор в Maple имеет следующую конструкцию:

```
if <Условие>
  then <Последовательность действий>
  { elif <Условие>
  then <Последовательность действий> }
  [ else <Последовательность действий> ]
end if;
> x := 5:
  if x < 0
    then print(`Negative`)
    elif x > 0
    then print(`Positive`)
    else print(`Zero`)
  end if;
```

Positive

Операторы цикла

Цикл в Maple может иметь одну из следующих конструкций:

```
for <Имя> from <Выражение>
  [ by <Выражение> ]  [ to <Выражение> ]
  [ while <Условие> ]
  do
    <Последовательность действий>
  end do;
for <Имя> in <Набор значений>
  [ while <Условие> ]
  do
    <Последовательность действий>
  end do;
while <Условие>
  do
    <Последовательность действий>
  end do;
```

```
> for i in [1, 2, 5, -1, 7, 12]
    while i > 0
    do
      print(i)
    end do;
```

1
2
5

Рассмотрим пример задания единичной матрицы:

```
> n := 3:
  A := array(1 .. n, 1 .. n):
  for i from 1 to n
    do
      for j from 1 to n
        do
          if i = j
            then A[i, j] := 1
            else A[i, j] := 0
          end if
        end do
    end do:
  evalm(A);
```

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

Процедуры

Общая форма задания процедуры в Maple:

```
<Имя процедуры> := proc(<Список параметров>)
  [ local <Список локальных переменных>; ]
  [ global <Список глобальных переменных>; ]
  [ options <Список расширяющих ключей>; ]
  [ description <Комментарий к процедуре>; ]
  <Тело процедуры>
end proc;
```

Процедура возвращает значение последнего выражения в её теле.

Реализуем процедуру вычисления модуля комплексного числа:

```
> modc := proc(z)
    sqrt(Re(z)^2 + Im(z)^2)
  end proc:
  modc(3 + 4 * I);
```

5

При подготовке процедур надо предусматривать их поведение при возможных ошибках. При выявлении ошибки предусматривается вывод соответствующего сообщения. Для этого используется функция `error`.

Реализуем процедуру вычисления квадратного корня из действительного числа с использованием функции `error`:

```
> sqroot := proc(x)
    if not (type(x, numeric) or type(x, realcons))
      or signum(x) = -1
      then error “invalid variable x: %1”, x
      else x^(1 / 2)
    end if
  end proc:
  sqroot(-2);
Error, (in sqroot) invalid variable x: -2
```

Расширяющий ключ `remember` обеспечивает занесение результатов обращений к процедуре в таблицу памяти. Этот ключ особенно полезен при реализации рекурсивных процедур.

Рассмотрим пример из *Лабораторной работы №4*. Реализуем процедуру вычисления чисел Фибоначчи, используя ключ `remember`:

```
> F := proc(n::nonnegint)
    if n < 2
      then n
      else F(n - 2) + F(n - 1)
    end if
  end proc:
  time(F(100));
```

0.

Подробную информацию о процедурах и ключах расширения процедур можно получить, исполнив команды `?proc` и `?options` соответственно.

Создание библиотеки процедур

Лабораторная работа №5.
Создание библиотеки процедур.

Создадим библиотеку процедур, работающих с комплексными числами, записанными в показательной форме. Вначале зададим пустую таблицу под будущие процедуры:

```
> ComplexExpLib := table([]):
```

Любое комплексное число z может быть записано в виде $z = abs(z) \cdot e^{i\,argument(z)} = r \cdot e^{ij}$. Будем представлять комплексное число z в качестве списка, состоящего из двух элементов: r и j.

Реализуем процедуру перевода комплексных чисел из алгебраической формы в показательную:

```
> ComplexExpLib[AlgToExp] := proc(z::algebraic)
    simplify([abs(z), argument(z)])
  end proc:
```

Реализуйте следующие процедуры:
перевод комплексных чисел из показательной формы в алгебраическую; проверка равенства двух комплексных чисел; умножение и деление двух комплексных чисел; возведение комплексного числа в n степень; вычисление k-го значения корня n степени из комплексного числа по формуле Муавра.

После исполнения команды `with(ComplexExpLib)`, можно проверить работу написанных процедур.

Сохраните библиотеку на диск с помощью команды `save`:

```
> save(ComplexExpLib, `c:/ComplexExpLib.m`):
```

Чтобы использовать созданную библиотеку процедур, нужно выполнить последовательность команд:

```
> read(`c:/ComplexExpLib.m`):
  with(ComplexExpLib):
```

Работа с файлами

Для записи данных в файл служит функция `writedata`:

```
writedata[APPEND](filename, data, format);
```

Данные могут задаваться списком, вектором или матрицей, формат данных может быть `integer`, `float` или `string`. Необязательный указатель `APPEND` используется, если данные дописываются в уже созданный файл.

```
> data := array([[1, 2, 3, 4], [5, 6, 7, 8]]):
  writedata(`c:/data.txt`, data, integer):
```

Считывание данных из файла обеспечивает функция `readdata`:

```
readdata(filename, format, n);
```

Здесь `n` – число считываемых столбцов.

```
> data := readdata(`c:/data.txt`, integer, 2);
```

$$data := [[1,2],[5,6]]$$

4. Средства графики системы Maple

Двумерная графика

Для построения двумерных графиков служит процедура `plot`.

При построении графика функции, заданной явно, процедура `plot` записывается следующим образом:

```
plot(y(x), x = x1 .. x2, y1 .. y2, options);
```

При построении графика функции, заданной параметрически, процедура `plot` записывается следующим образом:

```
plot([x(t), y(t), t = t1 .. t2],
x1 .. x2, y1 .. y2, options);
```

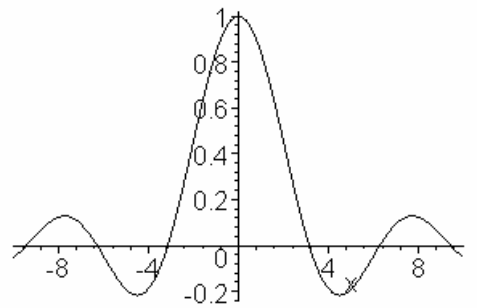
Для построения сеточной функции процедура `plot` записывается следующим образом:

```
plot(A, x1 .. x2, y1 .. y2, options);
```

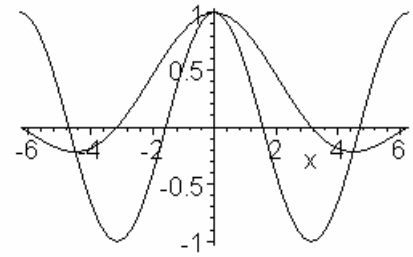
Здесь функция задана матрицей A; первый столбец матрицы – значения аргумента, второй столбец матрицы – соответствующие значения функции.

Некоторые значения параметра `options` перечислены в **Приложении 7**.

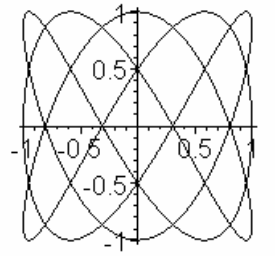
```
> plot(sin(x) / x, x = -10 .. 10, color = black);
```



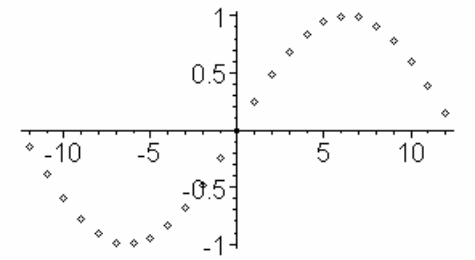

```
> plot([cos(x), sin(x) / x], x = - 2 * Pi, 2 * Pi,
  color = black);
```




```
> plot([sin(3 * t), cos(5 * t), t = 0 .. 2 * Pi],
  color = black);
```



```
> A := [[n, sin(n / 4)] $n = -12 .. 12]:
  plot(A, color = black, style = point);
```



Трёхмерная графика

Для построения трёхмерных графиков служит процедура `plot3d`.

При построении графика поверхности, заданной явно, процедура `plot3d` записывается следующим образом:

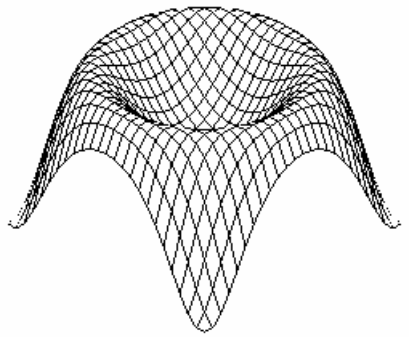
```
plot3d(z(x, y), x = x1 .. x2, y = y1 .. y2,
options);
```

При построении графика поверхности, заданной параметрически, процедура `plot3d` записывается следующим образом:

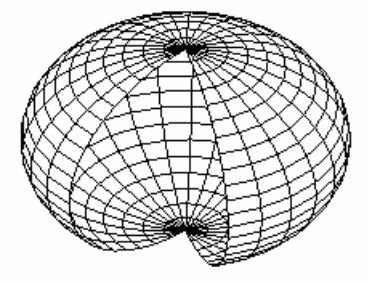
```
plot3d([x(s, t), y(s, t), z(s, t)], s = s1 .. s2,
t = t1 .. t2, options);
```

Некоторые значения параметра `options` перечислены в **Приложении 7**.

```
> plot3d(sin(x^2 + y^2), x = -Pi / 2 .. Pi / 2,
  y = -Pi / 2 .. Pi / 2, color = white);
```



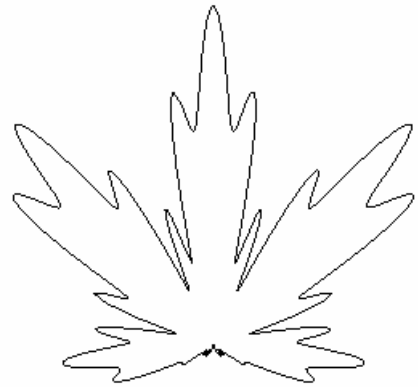
```
> plot3d(1, s = 0 .. 2 * Pi, t = 0 .. Pi,
  coords = spherical, color = white);
```



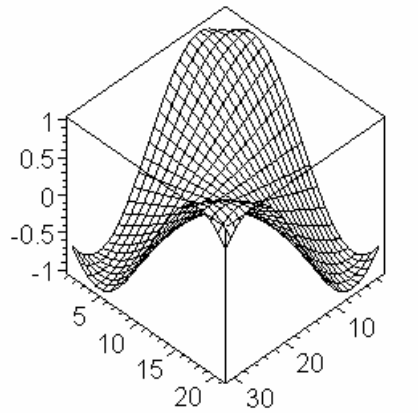
Расширенные средства графики

Часто удобно использовать функции из графического пакета `plots`. Некоторые функции из этого пакета представлены в **Приложении 7**. Подробную информацию о пакете `plots` можно получить, исполнив команду `?plots`.

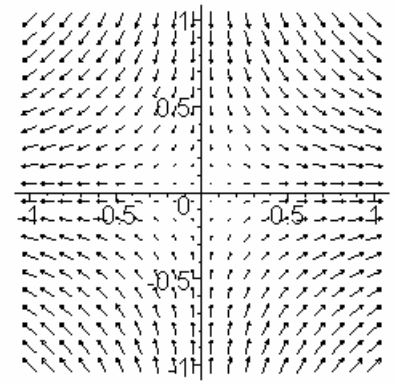
```
> with(plots):
  R := t -> 100 / (100 + (t – Pi / 2)^8) *
  (2 – sin(7 * t) – cos(30 * t) / 2):
  polarplot(R(t), t = - Pi / 2 .. 3 / 2 * Pi,
  color = black, axes = NONE);
```



```
> A := [seq([seq(sin(n * m / (20 * Pi)),
  n = -10 .. 10)], m = -15 .. 15)]:
  listplot3d(A, color = white, axes = BOXED);
```




```
> f := (x, y) -> x / sqrt(x^2 + y^2 + 1):
  g := (x, y) -> - y / sqrt(x^2 + y^2 + 1):
  fieldplot([f, g], -1 .. 1, -1 .. 1,
  arrows = SLIM);
```



5. Работа с полиномами

Оценка степени полинома и его коэффициентов

Функция `randpoly` возвращает случайный полином; функция `collect` возвращает полином, объединяя коэффициенты при степенях указанной переменной:

```
> p := randpoly([x, y], degree = 2);
  p := collect(p, x);
```

$$p := -61 + 41x - 58y - 90x^2 + 53xy - y^2$$

$$p := -90x^2 + (53y + 41)x - 61 - y^2 - 58y$$

```
> q := (x - 2 * y)^2 + 3 * x - 4 * (y^2 + y) + 5:
  q := collect(q, x);
```

$$q := x^2 + (3 - 4y)x - 4y + 5$$

Некоторые функции, используемые для оценки степени полинома и его коэффициентов, представлены в **Приложении 8**.

```
> degree(p, x);
```

$$2$$

```
> degree(q, [x, y]);
```

$$2$$

```
> coeffs(q, [x, y]);
```

$$-4y + 5, 3 - 4y, 1$$

Вычисление корней полинома

Для вычисления корней полиномов, зависящих от одной переменной, служит функция `roots`. Корни возвращаются в виде $[[r_1, k_1], \ldots, [r_n, k_n]]$, где r_i – корень полинома кратности k_i:

```
> p := 4 * x^5 + x^3 + x^2 - 3 * x + 1:
  roots(p, x);
  factor(p);
```

$$\left[\left[\frac{1}{2}, 2\right], [-1, 1]\right]$$

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

Функция `realroot` возвращает интервалы, в которых находятся действительные корни полинома:

```
> realroot(p, x);
```

$$[[0, 2], [-2, 0]]$$

Лабораторная работа №6.
Исследование полинома на устойчивость.

Исследуйте полином $L(p) = a_0 + a_1 x + \ldots + a_n x^n$ с вещественными коэффициентами на устойчивость, воспользовавшись критерием Рауса-Гурвица.

6. Решение уравнений и неравенств

Решение уравнений

Для решения уравнений в аналитическом виде используется функция `solve`:

```
> eqn := sqrt(ln(x)) = 2:
  solve(eqn, x);
```

$$e^4$$

```
> f := x -> sqrt(ln(x)) - 2:
  solve(f(x), x);
```

$$e^4$$

```
> eqn := sin(x) = 1 / 2:
  solve(eqn, x);
```

$$\frac{p}{6}$$

Чтобы функция `solve` выдавала все решения уравнения, глобальной переменной `_EnvAllSolutions` присваивают значение `true`:

```
> _EnvAllSolutions := true:
  eqn := sin(x) = 1 / 2:
  solve(eqn, x);
```

$$\frac{1}{6}p + \frac{2}{3}p\_B1\sim + 2p\_Z1\sim$$

В решениях могут встречаться следующие обозначения: `_Z` – целое число, `_NN` – целое неотрицательное число, `_B` – 0 или 1.

При решении систем уравнений они и список переменных задаются как множества. Функция `assign` обеспечивает присваивание переменным значений, полученных в результате решения системы:

```
> eqns := {x * y = 2, x + y = 3}:
  sol := solve(eqns, {x, y}):
  [x, y];
  assign(sol):
  [x, y];
  unassign('x', 'y'):
```

$$[x,y]$$

$$[1,2]$$

Функцию подстановки `subs` применяют, когда нужно проверить, является ли определённое значение решением уравнения:

```
> eqn := x^2 - 3 * x + 2 = 0:
  sol := solve(eqn, x):
  subs(x = sol[1], eqn);
  subs(x = -1, eqn);
```

$$0 = 0$$

$$6 = 0$$

Иногда результаты решения уравнений представляются в компактном виде с помощью функции `RootOf`. В этом случае все корни можно получить с помощью функции `allvalues`:

```
> eqns := {x * y = 2 * a, x + y = 2 * b}:
  solve(eqns, {x, y});
  allvalues(%);
```

$$\left\{x=-RootOf\left(\_Z^2-2\_Z\,b+2a\right)+2b, y=RootOf\left(\_Z^2-2\_Z\,b+2a\right)\right\}$$

$$\left\{x=b-\sqrt{b^2-2a}, y=b+\sqrt{b^2-2a}\right\}, \left\{x=b+\sqrt{b^2-2a}, y=b-\sqrt{b^2-2a}\right\}$$

С помощью функции `solve` можно решать и функциональные уравнения:

```
> eqn := sqrt(f(x)) = x:
  f := solve(eqn, f):
  f(x);
```

$$x^2$$

Решение неравенств

Для решения неравенств в аналитическом виде, как и для уравнений, используется функция `solve`:

```
> eqn := sqrt(ln(x)) < 2:
  solve(eqn, x);
```

$$RealRange\left(1, Open\left(e^4\right)\right)$$

К сожалению, в **Maple 8** функция `solve` устойчива при работе только с одним неравенством, и при поиске решения только по одной переменной:

```
> eqns := {x^2 + y^2 <= 1, x >= 0, y >= 0}:
  sol := solve(eqns, {x, y});
```

$$sol :=$$

Решение уравнений в численном виде

Для решения уравнений в численном виде используется функция `fsolve`:

```
> f := x -> x^4 - 4 * x^3 + 2 * x^2 - 4 * x + 1:
  fsolve(f(x), x);
```

$$0.2679491924, 3.732050808$$

Функция `fsolve` позволяет задавать интервал, на котором ищется решение, а также искать корни полинома в комплексной форме:

```
> f := x -> x^4 - 4 * x^3 + 2 * x^2 - 4 * x + 1:
  fsolve(f(x), x = 2 .. 4);
```

$$3.732050808$$

```
> fsolve(f(x), x, complex);
```

$$-1.I, 1.I, 0.2679491924, 3.732050808$$

Аналогично решаются системы уравнений:

```
> f := (x, y) -> sin(x + y) - exp(x) * y:
  g := (x, y) -> x^2 - y - 2:
  fsolve({f(x, y), g(x, y)}, {x = 0, y = 0});
```

$$\left\{x=-1.552838698, y=-0.6687012050\right\}$$

Лабораторная работа №7.
Решение уравнений в численном виде.

Создадим процедуру вычисления корней уравнения $f(x)=0$ методом половинного деления интервала:

```
> floatsolve := proc(ff, xint, epsilon)
    local x, a, b, c, f;
    x := lhs(xint);
    a := op(rhs(xint))[1];
    b := op(rhs(xint))[2];
    f := unapply(evalf(ff), x);
    if f(a) * f(b) > 0
      then error "Неверно введённый интервал"
      else if f(a) * f(b) = 0
             then
               if f(a) = 0
                 then c := a
                 else c := b
               end if
             else
               while (abs(b - a) > epsilon)
                     and (f(c) <> 0)
                 do
                   c := (a + b) / 2;
                   if f(a) * f(c) > 0
                     then a := c
                     else b := c
                   end if
                 end do
           end if;
           evalf(c);
    end if;
  end proc:
> f := x -> x^4 – 4 * x^3 + 2 * x^2 – 4 * x + 1:
  fsolve(f(x), x = 2 .. 4);
  floatsolve(f(x), x = 2 .. 4, 1E-4);
```

3.732050808
3.731994629

Модифицируйте процедуру `floatsolve` так, чтобы она объединяла в себе метод половинного деления и метод секущих. Для этого добавьте процедуре ещё один параметр, определяющий метод поиска решения, переопределите команду присваивания точке `c` значения и измените условие выхода из цикла `while`.

Убедитесь, что погрешность обоих методов имеет порядок $O(e)$.

Решение уравнений в целых числах

Для решения уравнений в целых числах используется функция `isolve`:

```
> eqn := x * y = 1:
  isolve(eqn);
```

$$\{x=-1,y=-1\},\{x=1,y=1\}$$

```
> eqn := 2 * x + 3 * y = 5:
  isolve(eqn);
```

$$\{x=1-3\_Z1,y=1+2\_Z1\}$$

Решение рекуррентных уравнений

Для решения рекуррентных уравнений используется функция `rsolve`:

```
> eqn := {f(n) = n * f(n - 1), f(0) = 1}:
  rsolve(eqn, f);
```

$$G\left(n+1\right)$$

```
> eqn := {F(n + 2) = F(n + 1) + F(n),
  F(0) = 0, F(1) = 1}:
  rsolve(eqn, F);
```

$$\frac{\sqrt{5}\left(\frac{1}{2}+\frac{\sqrt{5}}{2}\right)^{n}}{5}-\frac{\sqrt{5}\left(\frac{1}{2}-\frac{\sqrt{5}}{2}\right)^{n}}{5}$$

7. Интерполирование функций

Полиномиальная интерполяция

Функция `intepr` создаёт полином минимально необходимой степени, проходящий через заданные точки. При этом интерполируемая функция задаётся двумя векторами: вектором абсцисс и вектором ординат.

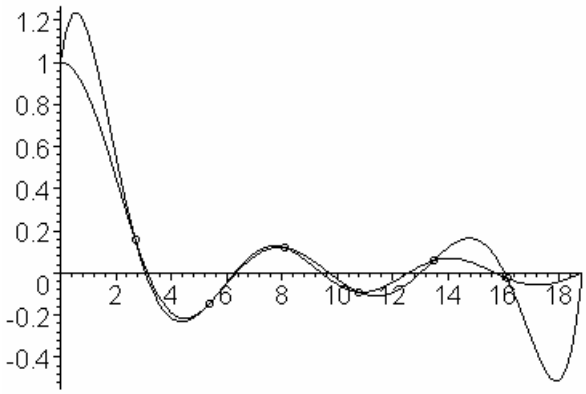
Построим интерполяционный полином для функции $y(x)=\frac{sin(x)}{x}$ на равномерной сетке на отрезке $[0,6p]$:

```
> y := x -> if x = 0
              then 1
              else sin(x) / x
            end if:
  T := 6 * Pi:
  N := 7:
  X := [(T * n / N) $n = 0 .. N]:
  Y := map(y, X):
  L := x -> interp(X, Y, x):
  evalf(L(x), 3);
```

$$1.+0.94x-1.2x^{2}+0.38x^{3}-0.056x^{4}+0.0042x^{5}-0.00016x^{6}+0.27\,10^{-5}x^{7}$$

Построим график получившегося интерполяционного полинома:

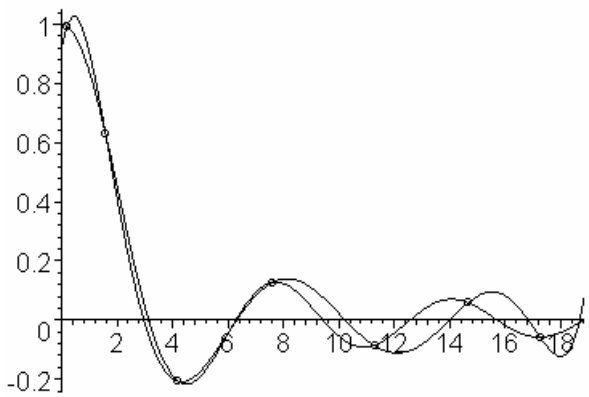
```
> A := [[T * n / N, y(T * n / N)] $n = 1 .. N]:
  plot([y, L, A], 0 .. T, color = black,
  style = [LINE, LINE, POINT]);
```



Как видно из графика, на границах области интерполирования отклонения полинома существенны. Интерполяционные свойства полинома можно улучшить, выбрав узлы интерполирования следующим образом:

$X_n = \frac{(b-a)c_n + (b+a)}{2}$, где $c_n = cos\left(\frac{(1+2n)p}{2(N+1)}\right)$ – нуль полинома Чебышева.

```
> y := x -> if x = 0
              then 1
              else sin(x) / x
            end if:
  T := 6 * Pi:
  N := 7:
  C := n -> cos((1 + 2 * n) * Pi / (2 * (N + 1))):
  X := [((T * C(n) + T) / 2) $n = 0 .. N]:
  Y := map(y, X):
  L := x -> interp(X, Y, x):
  A := [[(T * C(n) + T) / 2,
  y((T * C(n) + T) / 2)] $n = 1 .. N]:
  plot([y, L, A], 0 .. T, color = black,
  style = [LINE, LINE, POINT], symbol = CIRCLE);
```



Лабораторная работа №7.
Интерполяция методом Ньютона.

Интерполяционный полином Ньютона имеет вид

$y(x_0)+\sum_{n=1}^{N} w_{n-1}(x)\, y(X_0, X_1, ..., X_n)$, где $w_n(x)=(x-X_0)(x-X_1)...(x-X_n)$.

Здесь $y(X_0, X_1, ..., X_n)$ – разделённая разность порядка n.

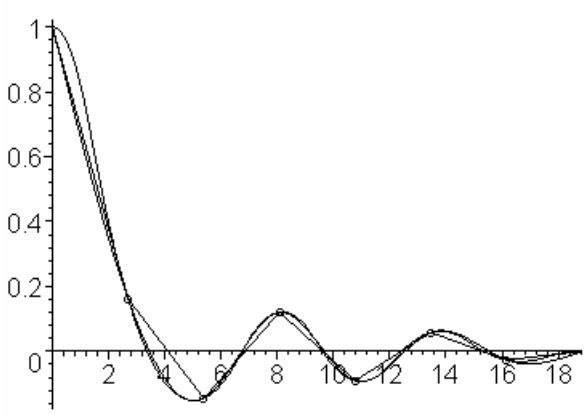
На отрезке $[0,6p]$ постройте интерполяционный полином Ньютона для функции $y(x)=\frac{sin(x)}{x}$. Убедитесь, что он совпадает с интерполяционным полиномом, построенным с помощью функции `interp`.

Интерполяция сплайнами

При интерполяции сплайнами на каждом интервале интерполируемая функция представляется в виде полинома, своего для каждого интервала. Кроме того, на сплайн накладывается условие гладкости.

В Maple для выполнения интерполяции сплайнами используется функция `spline`:

```
> y := x -> if x = 0
              then 1
              else sin(x) / x
            end if:
  T := 6 * Pi:
  N := 7:
  X := [(T * n / N) $n = 0 .. N]:
  Y := map(y, X):
  S1 := x -> spline(X, Y, x, linear):
  S2 := x -> spline(X, Y, x, quadratic):
  S3 := x -> spline(X, Y, x, cubic):
  A := [[T * n / N, y(T * n / N)] $n = 1 .. N]:
  plot([S1, S2, S3, A], 0 .. T, color = black,
  style = [LINE, LINE, LINE, POINT],
  linestyle = [DASHDOT, DASH, SOLID],
  symbol = CIRCLE);
```



Общая задача интерполирования

Лабораторная работа №8.
Общая задача интерполирования.

Будем искать интерполяционную функцию в виде $F_N(x)=\sum_{n=0}^{N} a_n j_n(x)$, где $j_0(x), j_1(x),\ldots, j_N(x)$ – базовые интерполяционные функции, которые должны быть независимыми. Тогда из условия совпадения значений интерполяционной и интерполируемой функций в узловых точках получим систему из $N+1$ уравнения относительно $N+1$ неизвестного коэффициента a_n: $F_N(X_n)=Y_n$, $n=0,\ldots,N$.

```
> interpolate := proc(X, Y, f)
    local n, N, L, t, eqns, a, s;
    N := linalg[vectdim](X) - 1;
    s := 0;
    for n from 0 to N
      do
        s := s + a[n] * f(n, t)
      end do;
    L := unapply(s, t);
    eqns := {};
    for n from 1 to N + 1
      do
        eqns := eqns union {Y[n] = L(X[n])}
      end do;
    solve(eqns, {seq(a[n], n = 0 .. N)});
    assign(%);
    L
  end proc:
```

Протестируйте процедуру `interpolate` на следующих системах базовых интерполяционных функций:

степенные функции $j_n(x)=x^n$;

полиномы Чебышева $j_n(x)=cos(n\,arccos(x))$;

степенные функции $j_n(x)=\begin{cases} cos(n\,x), & n=0,2,4,\ldots; \\ sin(n\,x), & n=1,3,5,\ldots. \end{cases}$

В качестве интерполируемой функции возьмите табулированную функцию:

$$\begin{cases} X=[-2,-1,1,2], \\ Y=[-5,0,4,15]. \end{cases}$$

Сравните полученные результаты. Постройте графики получившихся интерполяционных функций.

8. Дифференцирование функций

Вычисление пределов функций

Для вычисления пределов функций используется функция `limit`:

```
> y := x -> (tan(x) - x) / (x - sin(x)):
  Limit(y(x), x = 0) = limit(y(x), x = 0);
```

$$\lim_{x \to 0} \frac{\tan(x) - x}{x - \sin(x)} = 2$$

```
> y := x -> (sin(x) / x)^(1 / (x^2)):
  Limit(y(x), x = 0) = limit(y(x), x = 0);
```

$$\lim_{x \to 0} \left(\frac{\sin(x)}{x} \right)^{\left(\frac{1}{x^2} \right)} = e^{\left(\frac{-1}{6} \right)}$$

```
> y := x -> cot(x):
  Limit(y(x), x = 0, right) =
  limit(y(x), x = 0, right);
```

$$\lim_{x \to 0+} \cot(x) = \infty$$

```
> Limit(y(x), x = 0, left) =
  limit(y(x), x = 0, left);
```

$$\lim_{x \to 0-} \cot(x) = -\infty$$

Исследование функций на непрерывность

Функция `iscont` позволяет исследовать функции на непрерывность:

```
> y := x -> x + 1 / x:
  iscont(y(x), x = 0 .. 1);
```

true

```
> iscont(y(x), x = 0 .. 1, closed);
```

false

Функция `discont` служит для поиска точек разрыва функций:

```
> discont(y(x), x);
```

$\{0\}$

```
> u := x -> GAMMA(x):
  discont(u(x), x);
```

$\{-\_NN1\sim\}$

Функция `singular` служит для поиска особых точек функций:

```
> singular(y(x), x);
```

$\{x = 0\}, \{x = \infty\}, \{x = -\infty\}$

```
> singular(u(x), x);
```

$\{x = -\_N1\sim + 1\}, \{x = \infty\}$

Вычисление производных функций

Для вычисления производных используется функция `diff`:

```
> y := x -> x + 1 / x:
  Diff(y(x), x) = diff(y(x), x);
```

$$\frac{d}{dx}\left(1+\frac{1}{x}\right)=1-\frac{1}{x^2}$$

```
  Diff(y(x), x$2) = diff(y(x), x$2);
```

$$\frac{d^2}{dx^2}\left(1+\frac{1}{x}\right)=\frac{2}{x^3}$$

```
> u := (x, y, z) -> sin(x) * sin(y) * sin(z):
  Diff(u(x, y, z), x$2, y) =
  diff(u(x, y, z), x$2, y);
```

$$\frac{\partial^3}{\partial y\,\partial x^2}(sin(x)sin(y)sin(z))=-sin(x)cos(y)sin(z)$$

Функция `diff` позволяет, например, построить график производной, но не позволяет вычислить значение производной в конкретной точке. Этот недостаток устраняется созданием дифференциального оператора:

```
> D(y);
  Diff(y, x) = D(y)(2);
```

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

$$\left(\frac{\partial}{\partial x}y\right)(2):=\frac{3}{4}$$

Создадим оператор Лапласа $D=\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}$ для описанной выше функции $u(x,y,z)=sin(x)sin(y)sin(z)$:

```
> Lu := (x, y, z) -> D[1$2](u)(x, y, z) +
  D[2$2](u)(x, y, z) + D[3$2](u)(x, y, z):
  Diff(u, x$2) + Diff(u, y$2) + Diff(u, z$2) =
  Lu(x, y, z);
```

$$\left(\frac{\partial^2}{\partial x^2}u\right)+\left(\frac{\partial^2}{\partial y^2}u\right)+\left(\frac{\partial^2}{\partial z^2}u\right)=-3\,sin(x)sin(y)sin(z)$$

Лабораторная работа №9.
Численное дифференцирование функций одной переменной с использованием интерполяционных формул.

Продифференцируйте интерполяционный полином Ньютона, построенный в *Лабораторной работе №7*. Найдите значения производных в узлах интерполирования. Исследуйте зависимость погрешности численного дифференцирования от величины $h=\max\limits_{0\le n\le N-1}(X_n-X_{n-1})$.

Лабораторная работа №10.

Численное дифференцирование функций одной переменной.

Пусть $X=[X_0,X_1,...,X_N]$ – вектор, задающий разбиение отрезка $[a,b]$:

$a=X_0<X_1<...<X_N=b$.

Пусть $Y=[Y_0,Y_1,...,Y_N]$ – вектор соответствующих значений заданной функции $y(x)$: $Y_n=y(X_n)$, $n=0,1,...,N$.

Для каждого $n=1,2,...,N-1$ постройте

а) параболу,

б) окружность,

проходящую через точки (X_{n-1},Y_{n-1}), (X_n,Y_n) и (X_{n+1},Y_{n+1}).

Вычислите значения производных построенных кривых в точках $X_1,X_2,...,X_{N-1}$. Сравните получившиеся значения с точными значениями производной функции $y(x)$ в точках $X_1,X_2,...,X_{N-1}$.

Опишем процедуру, вычисляющую производную функции, заданной параметрически:

```
> parametricdiff := proc(x, y, t)
    diff(y, t) / diff(x, t)
  end proc:
  x := t -> t * cos(t):
  y := t -> t * sin(t):
  dy := t -> parametricdiff(x(t), y(t), t):
  'dy / dx' = dy(t);
```

$$\frac{\partial}{\partial x}y=\frac{sin(t)+t\,cos(t)}{cos(t)-t\,sin(t)}$$

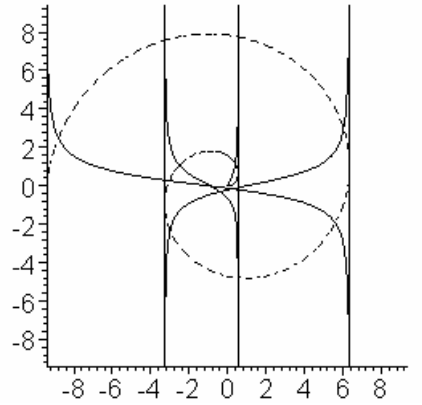
Как и в случае с функцией `diff`, невозможно вычислить значение таким образом описанной производной в конкретной точке. Поэтому и здесь правильнее будет создать дифференциальный оператор:

```
> parametricD := proc(x, y)
    D(y) / D(x)
  end proc:
  dy := parametricD(x, y)(t):
  Diff(y, x) = dy(t);
```

$$\frac{\partial}{\partial x}y=\frac{sin(t)+t\,cos(t)}{cos(t)-t\,sin(t)}$$

Построим теперь график функции и её производной:

```
> T := 3 * Pi:
  plot([[x, y, 0 .. T], [x, dy , 0 .. T]],
  -T .. T, -T .. T, color = black,
  linestyle = [DASHDOT, SOLID], axes = FRAME);
```



Лабораторная работа №11.

Дифференцирование неявно заданных функций.

Производная неявно заданной функции вычисляется с помощью функции `implicitdiff`.

Найдём производную $\frac{\partial y}{\partial x}$, если функция задана уравнением $x^2+4y^2=1$:

```
> F := x^2 + 4 * y^2 = 1:
  Diff(y, x) = implicitdiff(F, y, x);
```

$$\frac{\partial}{\partial x}y=-\frac{x}{4y}$$

Найдём производные $\frac{\partial z}{\partial x}$ и $\frac{\partial^2 z}{\partial x^2}$, если функция задана системой уравнений

$$\begin{cases} x^2+y^2=z, \\ x^2-xy+y^2=1: \end{cases}$$

```
> F1 := x^2 + y^2 = z:
  F2 := x^2 – x * y + y^2 = 1:
  Diff(z, x) =
  simplify(implicitdiff({F1, F2}, {y, z}, z, x));
  Diff(z, x$2) =
  factor(implicitdiff({F1, F2}, {y, z}, z, x$2));
```

$$\frac{\partial}{\partial x}z=\frac{2\left(x^2-y^2\right)}{x-2y}$$

$$\frac{\partial^2}{\partial x^2}z=\frac{2\left(5x^3-12yx^2+15y^2x-4y^3\right)}{(x-2y)^3}$$

Используя теорему о неявной функции, постройте дифференциальные операторы для рассмотренных выше неявных функций.

Замена переменных

Чтобы выполнить замену переменных в выражении, содержащем производные, используют функцию `dchange` из пакета `PDEtools`.

В уравнении $x^2\frac{d^2y}{dx^2}+x\frac{dy}{dx}+y=0$ выполним замену переменной $x=e^t$:

```
> with(PDEtools):
  eqn := x^2 * (D@@2)(y)(x) + x * D(y)(x) +
  y(x) = 0:
  simplify(dchange(x = exp(t), eqn));
```

$$\left(\frac{d^2}{dt^2}y(t)\right)+y(t)=0$$

Выразим оператор Лапласа $D=\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}$ через полярные координаты:

```
> Lu := (x, y) -> D[1$2](u)(x, y) +
  D[2$2](u)(x, y):
  tr := {x = r * cos(phi), y = r * sin(phi)}:
  simplify(dchange(tr, Lu(x, y), {r, phi}));
```

$$\frac{\left(\frac{d^2}{df^2}u(f,r)\right)+\left(\frac{d}{dr}u(f,r)\right)r+\left(\frac{d^2}{dr^2}u(f,r)\right)r^2}{r^2}$$

Поиск экстремумов функций

Функция `extrema` служит для поиска локальных условных и безусловных экстремумов функций:

```
> y := x -> x + 1 / x:
  extrema(y(x), {}, x);
```

$\{-2,2\}$

```
> z := (x, y) -> (x - 3)^2 + (y + 4)^2 - 12:
  extrema(z(x, y), {x + y = 1}, {x, y});
```

$\{-10\}$

К сожалению, функция `extrema` предназначена для решения задач на условный экстремум с ограничениями только типа равенств.

Для поиска глобальных безусловных минимумов и максимумов функций служат функции `minimize` и `maximize`:

```
> minimize(y(x), x = 0 .. infinity);
```

2

```
> minimize(z(x, y), location);
```

$-12,\{[\{x=3,y=-4\},-12]\}$

Лабораторная работа №12.
Поиск безусловных экстремумов функций одной переменной.

Опишем процедуру поиска локальных безусловных экстремумов функций одной переменной:

```
> ExtrPoints := proc(f)
    local x;
    for x in {solve(D(f)(x) = 0, x)}
      do
        if evalf(D[1$2](f)(x)) = 0
          then print(cat("x = ", x, ": ?"))
          elif evalf(D[1$2](f)(f)(x)) > 0
          then print(cat("x = ", x, ": Minimum"))
          else print(cat("x = ", x, ": Maximum"))
        end if
      end do
  end proc:
> y := x -> 12 * x^5 + 15 * x^4 – 40 * x^3:
  ExtrPoints(y);
```

$"x = -2 : Maximum"$

$"x = 0 : ?"$

$"x = 1 : Minimum"$

Доработайте процедуру `ExtrPoints` так, чтобы в результате её выполнения не оставалось точек, требующих дополнительного исследования.

Лабораторная работа №13.
Поиск безусловных экстремумов функций нескольких переменных.

Создайте процедуру поиска локальных безусловных экстремумов функций нескольких переменных. Достаточным условием минимума функции нескольких переменных является положительная определённость матрицы вторых производных этой функции, достаточное условие максимума – отрицательная определённость этой матрицы.

Лабораторная работа №14.
Поиск условных экстремумов функций нескольких переменных.

Приложение **1**
Функции, используемые при работе с выражениями

Таблица 1.1
Возможные значения параметра `form` функции преобразования выражений `convert(expr, form)`

Параметр	Описание
`float`	Преобразование в формат числа с плавающей точкой
`rational`	Преобразование в формат рационального числа
`fraction`	Преобразование в формат рациональной дроби
`decimal`	Преобразование числа в десятичный формат
`binary`	Преобразование числа в двоичный формат
`octal`	Преобразование числа в восьмеричный формат
`hex`	Преобразование числа в шестнадцатеричный формат
`arabic`	Переход от римских цифр к арабским
`roman`	Переход от арабских цифр к римским
`radians`	Переход от градусов к радианам
`degrees`	Переход от радиан к градусам
`vector`	Преобразование в векторный тип
`matrix`	Преобразование в матричный тип
`` `+` ``	Преобразование в сумму
`` `*` ``	Преобразование в произведение
`exp`	Переход от тригонометрических и гиперболических функций к экспоненциальным
`expln`	Переход от элементарных функций к экспоненциальным и тригонометрическим
`expsincos`	Переход от тригонометрических функций к синусу и косинусу, от гиперболических функций к экспоненциальным
`ln`	Переход от обратных тригонометрических и обратных гиперболических функций к логарифмическим
`sincos`	Переход от тригонометрических функций к синусу и косинусу, от гиперболических функций к гиперболическим синусу и косинусу
`tan`	Переход от тригонометрических и гиперболических функций к тангенсу
`trig`	Переход от экспоненциальных функций к тригонометрическим и гиперболическим
`GAMMA`	Переход от факториалов и биномиальных коэффициентов к гамма-функции
`factorial`	Переход от гамма-функции и биномиальных коэффициентов к факториалам
`binomial`	Переход от гамма-функции и факториалов к биномиальным коэффициентам
`signum`	Переход от модулей к функции знака
`abs`	Переход от функции знака и кусочно-гладких функций к модулям
`Heaviside`	Переход от кусочно-гладких функций, функции знака и модулей к функции Хевисайда
`piecewise`	Переход от функции Хевисайда, функции знака и модулей к кусочно-гладким функциям

Таблица 1.2
Возможные значения параметра `prop` функции `assume(expr, prop)`

Параметр	Описание
`numeric`	Число
`real`	Действительное число
`RealRange(x, y)`	Действительное число из отрезка $[a,b]$
`positive`	Положительное число
`negative`	Отрицательное число
`nonnegative`	Неотрицательное число
`nonpositive`	Неположительное число
`float`	Вещественное число с плавающей точкой
`rational`	Рациональное число
`irrational`	Иррациональное число
`fraction`	Рациональная дробь
`integer`	Целое число
`posint`	Положительное целое число
`negint`	Отрицательное целое число
`nonnegint`	Неотрицательное целое число
`nonposint`	Неположительное целое число
`prime`	Простое число
`even`	Чётное число
`odd`	Нечётное число
`complex`	Комплексное число
`nonreal`	Комплексное число, не лежащее на действительной оси
`imaginary`	Мнимое число
`GaussianInteger`	Комплексное число с целыми действительной и мнимой частями
`constant`	Константа
`realcons`	Действительная константа
`complexcons`	Комплексная константа
`scalar`	Скалярная величина
`vector`	Вектор
`matrix`	Матрица
`SquareMatrix`	Квадратная матрица
`triangular`	Треугольная матрица
`LowerTriangular`	Нижнетреугольная матрица
`UpperTriangular`	Верхнетреугольная матрица
`tridiagonal`	Трёхдиагональная матрица
`diagonal`	Диагональная матрица
`symmetric`	Симметрическая матрица, $A = A^T$
`antisymmetric`	Кососимметрическая матрица, $A = -A^T$
`idempotent`	Идемпотентная матрица, $A = A^2$
`singular`	Вырожденная матрица, $det(A) = 0$
`PositiveDefinite`	Положительно определённая матрица, $x^T Ax > 0 \ \ \forall x \neq q$

mapping	Функция
unary	Функция одного параметра
binary	Функция двух параметров
EvenMap	Чётная функция
OddMap	Нечётная функция
monotonic	Монотонная функция
StrictlyMonotonic	Строго монотонная функция
continuous	Непрерывная функция
differentiable	Дифференцируемая функция
Infinitely-Differentiable	Бесконечно дифференцируемая функция
PolynomialMap	Полином
LinearMap	Линейная функция

Таблица 1.3
Функции, используемые для оценивания выражений различного типа

Функция	Описание
eval(expr, eqns)	Символьное вычисление значения expr при заданных условиях eqns
evalf(expr)	Численное вычисление значения expr, вывод вычисленного значения в форме числа с плавающей точкой с мантиссой, порядок которой хранится в системной переменной Digits
evalf(expr, n)	Численное вычисление значения expr, вывод вычисленного значения в форме числа с плавающей точкой с мантиссой порядка n
evalhf(expr)	Численное вычисление значения expr, вывод вычисленного значения с точностью, присущей данной компьютерной системе
evalm(mexpr)	Вычисление значения матричного выражения mexpr
evalc(cexpr)	Вычисление значения комплексного выражения cexpr
evalb(bexpr)	Вычисление значения логического выражения bexpr
evalr(rexpr)	Оценка области значений интервального выражения rexpr
shake(expr, ampl)	Преобразование выражения expr в интервальное выражение (все константы const в expr заменяются на const * INTERVAL(1 – Float(10, -ampl) .. 1 + Float(10, -ampl))); оценка области значений полученного интервального выражения

Приложение **2**
Типы данных

Таблица 2.1
Типы данных

Тип	Описание
`numeric`	Число
`positive`	Положительное число
`negative`	Отрицательное число
`nonnegative`	Неотрицательное число
`nonpositive`	Неположительное число
`float`	Вещественное число с плавающей точкой
`rational`	Рациональное число
`fraction`	Рациональная дробь
`integer`	Целое число
`posint`	Положительное целое число
`negint`	Отрицательное целое число
`nonnegint`	Неотрицательное целое число
`nonposint`	Неположительное целое число
`prime`	Простое число
`even`	Чётное число
`odd`	Нечётное число
`complex`	Комплексное число
`nonreal`	Комплексное число, не лежащее на действительной оси
`imaginary`	Мнимое число
`constant`	Константа
`realcons`	Действительная константа
`complexcons`	Комплексная константа
`scalar`	Скалярная величина
`set`	Множество
`list`	Список
`vector`	Вектор
`matrix`	Матрица
`boolean`	Выражение логического типа
`function`	Функция
`symmfunc`	Симметрическая функция
`evenfunc`	Чётная функция
`oddfunc`	Нечётная функция
`trig`	Тригонометрическая функция
`arctrig`	Обратная тригонометрическая функция
`trigh`	Гиперболическая функция
`arctrigh`	Обратная гиперболическая функция
`polynom`	Полином
`linear`	Линейная функция
`quadratic`	Квадратичная функция
`cubic`	Кубическая функция
`quartic`	Функция четвёртого порядка

Приложение **3**
Математические функции

Таблица 3.1
Основные математические функции

Функция	Описание	Функция	Описание
`sin(x)`	Синус	`sinh(x)`	Синус гиперболический
`cos(x)`	Косинус	`cosh(x)`	Косинус гиперболический
`tan(x)`	Тангенс	`tanh(x)`	Тангенс гиперболический
`cot(x)`	Котангенс	`coth(x)`	Котангенс гиперболический
`sec(x)`	Секанс	`sech(x)`	Секанс гиперболический
`csc(x)`	Косеканс	`csch(x)`	Косеканс гиперболический
`arcsin(x)`	Арксинус	`arcsinh(x)`	Арксинус гиперболический
`arccos(x)`	Арккосинус	`arccosh(x)`	Арккосинус гиперболический
`arctan(x)`	Арктангенс	`arctanh(x)`	Арктангенс гиперболический
`arccot(x)`	Арккотангенс	`arccoth(x)`	Арккотангенс гиперболический
`arcsec(x)`	Арксеканс	`arcsech(x)`	Арксеканс гиперболический
`arccsc(x)`	Арккосеканс	`arccsch(x)`	Арккосеканс гиперболический

Функция	Описание
`exp(x)`	Экспоненциальная функция
`ln(x)`	Логарифм натуральный
`log[b](x)`	Логарифм по основанию `b`
`log10(x)`	Логарифм десятичный
`sqrt(x)`	Корень квадратный
`abs(x)`	Модуль
`signum(x)`	Знак
`round(x)`	Округлённое значение
`ceil(x)`	Наименьшее целое, большее или равное `x`
`floor(x)`	Наибольшее целое, меньшее или равное `x`
`trunc(x)`	Целая часть
`frac(x)`	Дробная часть
`max(x1, x2, ... , xN)`	Максимальное из чисел `x1`, `x2`, ... , `xN`
`min(x1, x2, ... , xN)`	Минимальное из чисел `x1`, `x2`, ... , `xN`

Таблица 3.2
Основные функции для работы с целыми числами

Функция	Описание
`factorial(n)`	Факториал
`iquo(n, m)`	Целочисленное частное при делении `n` на `m`
`irem(n, m)`	Остаток от деления `n` на `m`
`igcd(n1, n2, ... , nN)`	Наибольший общий делитель
`lcm(n1, n2, ... , nN)`	Наименьшее общее кратное
`ifactor(n)`	Разложение на простые множители
`isprime(n)`	Проверка, является ли число `n` простым

Таблица 3.3
Основные функции для работы с комплексными числами

Функция	Описание
`Re(z)`	Действительная часть
`Im(z)`	Мнимая часть
`abs(z)`	Модуль
`argument(z)`	Главное значение аргумента
`conjugate(z)`	Комплексно-сопряжённое число

Таблица 3.4
Некоторые специальные математические функции

Функция	Описание
`Dirac(x)`	Дельта-функция Дирака $d(x)=\begin{cases} d(0), & x=0; \\ 0, & x\neq 0 \end{cases}$
`Heaviside(x)`	Функция Хевисайда $h(x)=\begin{cases} 1, & x>0; \\ 0, & x<0 \end{cases}$
`GAMMA(x)`	Гамма-функция Эйлера $G(x)=\int_0^{\infty} e^{-t}t^{x-1}dt$
`Beta(x, y)`	Бета-функция Эйлера $B(x,y)=\int_0^{1} t^{x-1}(1-t)^{y-1}dt$
`Si(x)`	Интегральный синус $Si(x)=\int_0^{x}\frac{sin(t)}{t}dt$
`Ci(x)`	Интегральный косинус $Ci(x)=g+ln(x)+\int_0^{x}\frac{cos(t)-1}{t}dt$
`Shi(x)`	Интегральный гиперболический синус $Shi(x)=\int_0^{x}\frac{sinh(t)}{t}dt$
`Chi(x)`	Интегральный гиперболический косинус $Chi(x)=g+ln(x)+\int_0^{x}\frac{cosh(t)-1}{t}dt$
`Ei(n, x)`	Интегральная экспоненциальная функция $Ei(n,x)=\int_1^{\infty}\frac{e^{-xt}}{t^n}dt,\ n\geq 0,\ x>0$
`Ei(x)`	Интегральная экспоненциальная функция $Ei(x)=vp\int_{-\infty}^{x}\frac{e^t}{t}dt=g+ln(x)+\int_0^{x}\frac{e^t-1}{t}dt,\ x>0$
`Li(x)`	Интегральный логарифм $Li(x)=vp\int_0^{x}\frac{dt}{ln(t)}=Ei(ln(x)),\ x>1$

Приложение **4**

Процедуры и функции, используемые для работы с векторами и матрицами

Таблица 4.1
Функции из пакета `linalg`, используемые для работы с векторами

Функция	Описание
`vector(n, list)`	Создание вектора размерности n из списка `list`
`randvector(n)`	Создание случайного вектора размерности n
`vectdim(V)`	Размерность
`equal(U, V)`	Проверка совпадения векторов U и V
`norm(V, k)`	Норма $\|V\|_k = \sqrt[k]{\sum_{j=1}^{n} \|v_j\|^k}$, $k \ge 1$, $vect\,dim(V) = n$
`norm(V, infinity)`	Норма $\|V\|_\infty = \max_{1 \le j \le n} \|v_j\|$, $vect\,dim(V) = n$
`normalize(V)`	Нормированный вектор $normalize(V) = \frac{V}{\|V\|_2}$
`dotprod(U, V)`	Скалярное произведение, $vect\,dim(U) = vect\,dim(V)$
`crossprod(U, V)`	Векторное произведение, $vect\,dim(U) = vect\,dim(V) = 3$
`angle(U, V)`	Угол между векторами U и V, $vect\,dim(U) = vect\,dim(V)$

Таблица 4.2
Функции из пакета `linalg`, используемые для работы с матрицами

Функция	Описание
`matrix(m, n, list)`	Создание матрицы размерности m на n из списка `list`
`randmatrix(m, n)`	Создание случайной матрицы размерности m на n
`hilbert(n)`	Создание матрицы Гильберта размерности n на n
`rowdim(A)`	Число строк
`coldim(A)`	Число столбцов
`row(A, i)`	Строка с номером i
`col(A, j)`	Столбец с номером j
`minor(A, i, j)`	Минор, полученный вычёркиванием строки с номером i и столбца с номером j
`augment(A1, A2, ... , AN)`	Горизонтальное слияние матриц, $row\,dim(A_1) = row\,dim(A_2) = ... = row\,dim(A_N)$
`stack(A1, A2, ... , AN)`	Вертикальное слияние матриц, $col\,dim(A_1) = col\,dim(A_2) = ... = col\,dim(A_N)$
`equal(A, B)`	Проверка совпадения матриц A и B
`rank(A)`	Ранг матрицы A
`trace(A)`	След квадратной матрицы A, $trace(A) = \sum_{i=1}^{n} a_{ii}$, $row\,dim(A) = col\,dim(A) = n$
`det(A)`	Определитель квадратной матрицы A

`norm(A, 1)`	Норма $\|A\|_1 = \max\limits_{1\le j\le n}\sum\limits_{i=1}^{m}\|a_{ij}\|$, $\begin{cases} row\,dim(A)=m, \\ col\,dim(A)=n \end{cases}$
`norm(A, infinity)`	Норма $\|A\|_\infty = \max\limits_{1\le i\le m}\sum\limits_{j=1}^{n}\|a_{ij}\|$, $\begin{cases} row\,dim(A)=m, \\ col\,dim(A)=n \end{cases}$
`norm(A, frobenius)`	Норма $\|A\|_{frobenius} = \sqrt{\sum\limits_{i=1}^{m}\sum\limits_{j=1}^{n}\|a_{ij}\|^2}$, $\begin{cases} row\,dim(A)=m, \\ col\,dim(A)=n \end{cases}$
`cond(A, normname)`	Число обусловленности квадратной матрицы A $cond(A,norm\,name) = \|A\|_{norm\,name}\cdot\|A^{-1}\|_{norm\,name}$
`transpose(A)`	Транспонированная матрица
`multiply(A1, A2, ... , AN)`	Умножение матриц, $\begin{cases} row\,dim(A_k) = col\,dim(A_{k+1}), & k=1,...,N-1, \\ col\,dim(A_k) = row\,dim(A_{k+1}), & k=1,...,N-1 \end{cases}$
`multiply(A, V)`	Умножение матрицы A на вектор V, $col\,dim(A) = vect\,dim(V)$
`inverse(A)`	Обратная матрица
`adjoint(A)`	Сопряжённая матрица, $A\cdot adjoint(A) = det(A)\cdot E$
`LUdecomp(A, L = 'L', U = 'U')`	LU-разложение
`eigenvalues(A)`	Собственные значения
`jordan(A)`	Жорданова форма квадратной матрицы A

Таблица 4.3
Функции из пакета `linalg`, используемые для работы с векторными полями

Функция	Описание
`grad(expr, var)`	Градиент выражения `expr` по вектору переменных `var` $\nabla expr = \left[\frac{\partial\,expr}{\partial\,var_1}, \frac{\partial\,expr}{\partial\,var_2}, ..., \frac{\partial\,expr}{\partial\,var_n}\right]$, $vect\,dim(var) = n$
`potential(f, var, 'F')`	Проверка существования потенциала векторного поля `f`. Если потенциал существует, то он будет задан в `F`, при этом $\nabla F = f$
`diverge(f, var)`	Дивергенция векторного поля `f` $div(f) = dotprod(\nabla, f)$, $\nabla = \left[\frac{\partial}{\partial\,var_1}, \frac{\partial}{\partial\,var_2}, ..., \frac{\partial}{\partial\,var_n}\right]$, $vect\,dim(var) = vect\,dim(f) = n$
`curl(f, var)`	Ротор векторного поля `f` $curl(f) = crossprod(\nabla, f)$, $\nabla = \left[\frac{\partial}{\partial\,var_1}, \frac{\partial}{\partial\,var_2}, \frac{\partial}{\partial\,var_3}\right]$, $vect\,dim(var) = vect\,dim(f) = 3$

Приложение **5**
Константы

Таблица 5.1
Именованные константы

Константа	Описание
`Pi`	Число $p = 3.141592654$
`gamma`	Константа Эйлера $g = \lim_{n\to\infty}\left(\sum_{i=1}^{n}\frac{1}{i} - ln(n)\right) = 0.5772156649$
`Catalan`	Константа Каталана $\sum_{i=0}^{\infty}\frac{(-1)^i}{(2i+1)^2} = 0.9159655942$
`infinity`	Бесконечность ∞
`true`	Логическая константа «Истина»
`false`	Логическая константа «Ложь»
`FAIL`	Логическая константа «Неопределённость»

Приложение **6**
Операторы и операнды

Таблица 6.1
Операторы для работы с данными множественного типа

Оператор	Описание
`S union T`	Объединение двух множеств S и T
`S intersect T`	Пересечение двух множеств S и T
`S minus T`	Разность двух множеств S и T
`S subset T`	Проверка, является ли множество S подмножеством множества T
`s in S`	Проверка, является ли s элементом множества S

Таблица 6.2
Операторы в порядке их выполнения

Оператор	Тип
`||`	`binary`
`:-`	`binary`
`::`	`binary`
`%`	`nullary`
`&-operators`	`unary(prefix) / binary`
`.`	`unary(prefix / postfix) / binary`
`!`	`unary(postfix)`
`^ @@`	`binary`
`* &* / @ intersect`	`binary`
`+ - union minus`	`binary`
`mod`	`binary`
`subset`	`binary`
`..`	`binary`
`< <= > >= = <> in`	`binary`
`$`	`unary(prefix) / binary`
`not`	`unary(prefix)`
`and`	`binary`
`or`	`binary`
`xor`	`binary`
`implies`	`binary`
`->`	`binary`
`,`	`binary`
`assuming`	`binary`
`:=`	`binary`

Таблица 6.3
Возможные значения параметра `prop` функции `define(oper, prop)`

Параметр	Описание
`linear`	Оператор, линейный по первому аргументу
`multilinear`	Оператор, линейный по всем аргументам
`commutative`	Коммутативный оператор, $f(x,y)=f(y,x)$
`associative`	Ассоциативный оператор, $f(x,f(y,z))=f(f(x,y),z)=f(x,y,z)$
`identity`	Единичный оператор
`zero`	Нулевой оператор

Приложение 7
Средства графики

Таблица 7.1
Возможные значения параметра `options` процедуры `plot(..., options)`

Параметр	Возможные значения	Значение по умолчанию	Описание

`coords`	`cartesian`
	`polar`
	`...`
		`cartesian`	Система координат

`scaling`	`UNCONSTRAINED` `CONSTRAINED`	`UNCONSTRAINED`	Масштаб
`discont`	`false` `true`	`false`	Разбиение графика функции на промежутки непрерывности
`title`	`<Строка>`		Заголовок
`legend`	`<Строка>`		Легенда

`color`	`white`
	`black`
	`...`
		`red`	Цвет линий графика функции

`style`	`LINE`
	`POINT`
	`PATCH`
	`PATCHNOGRID`
		`PATCH`	Способ отображения графика функции

`linestyle`	`SOLID`
	`DOT`
	`DASH`
	`DASHDOT`
		`SOLID`	Тип линий

`thickness`	`<Целое число от 0 до 15>`	`0`	Толщина линий

`symbol`	`POINT`
	`CROSS`
	`CIRCLE`
	`BOX`
	`DIAMOND`
		`DIAMOND`	Тип символов для отображения точек

`symbolsize`	`<Целое положительное число>`	`10`	Размер символов для отображения точек

`axes`	`NORMAL`
	`FRAME`
	`BOXED`
	`NONE`
		`NORMAL`	Тип координатных осей

`labels`	`<Список из двух строк>`		Надписи для координатных осей
`xtickmarks`	`<Целое положительное число>`		Минимальное число отображаемых меток на оси абсцисс
`ytickmarks`	`<Целое положительное число>`		Минимальное число отображаемых меток на оси ординат

Таблица 7.2
Возможные значения параметра `options` процедуры `plot3d(..., options)`

Параметр	Возможные значения	Значение по умолчанию	Описание

`coords`	`rectangular`
	`cylindrical`
	`spherical`
	`...`
		`rectangular`	Система координат

`scaling`	`UNCONSTRAINED` `CONSTRAINED`	`UNCONSTRAINED`	Масштаб
`title`	`<Строка>`		Заголовок

`color`	`white`
	`black`
	`...`
			Цвет поверхности

`shading`	`XYZ`
	`XY`
	`Z`
	`ZGRAYSCALE`
	`ZHUE`
	`NONE`
		`XYZ`	Способ раскраски поверхности

`style`	`LINE`
	`POINT`
	`HIDDEN`
	`PATCH`
	`WIREFRAME`
	`CONTOUR`
	`PATCHNOGRID`
	`PATCHCONTOUR`
		`PATCH`	Способ отображения поверхности

`linestyle`	`SOLID`
	`DOT`
	`DASH`
	`DASHDOT`
		`SOLID`	Тип линий

`thickness`	`<Целое число от 0 до 15>`	`0`	Толщина линий

`symbol`	`POINT`
	`CROSS`
	`CIRCLE`
	`BOX`
	`DIAMOND`
		`POINT`	Тип символов для отображения точек

`symbolsize`	`<Целое положительное число>`	`10`	Размер символов для отображения точек

`axes`	`NORMAL`
	`FRAME`
	`BOXED`
	`NONE`
		`NONE`	Тип координатных осей

`labels`	`<Список из трёх строк>`		Надписи для координатных осей
`tickmarks`	`<Список из трёх целых положительных чисел>`		Минимальное число отображаемых меток на осях координат

Таблица 7.3
Функции из пакета `plots`, используемые для построения двумерных графиков

Функция	Описание
`display`	Построение графика для списка графических объектов
`animate`	Создание анимации графика функции
`polarplot`	Построение графика функции в полярной системе координат
`implicitplot`	Построение графика неявной функции
`complexplot`	Построение графика комплекснозначной функции
`listplot`	Построение графика сеточной функции
`pointplot`	Построение точечного графика
`polygonplot`	Построение многоугольника
`fieldplot`	Построение векторного поля
`gradplot`	Построение векторного поля градиента

Таблица 7.4
Функции из пакета `plots`, используемые для построения трёхмерных графиков

Функция	Описание
`display3d`	Построение графика для списка графических объектов
`animate3d`	Создание анимации графика функции
`spacecurve`	Построение пространственной кривой
`cylinderplot`	Построение графика функции в цилиндрической системе координат
`sphereplot`	Построение графика функции в сферической системе координат
`implicitplot3d`	Построение графика неявной функции
`complexplot3d`	Построение графика комплекснозначной функции
`listplot3d`	Построение графика сеточной функции
`pointplot3d`	Построение точечного графика
`polygonplot3d`	Построение многоугольника
`polyhedraplot`	Построение многогранника
`fieldplot3d`	Построение векторного поля
`gradplot3d`	Построение векторного поля градиента

Приложение **8**
Функции, используемые при работе с полиномами

Таблица 8.1
Функции, используемые для оценки степени и коэффициентов полинома

Функция	Описание
`degree(p, x)`	Степень полинома p по переменной x
`ldegree(p, x)`	Наименьшая степень полинома p по переменной x
`coeffs(p, x)`	Коэффициенты полинома p при различных степенях x
`coeff(p, x, n)`	Коэффициент полинома p при степени x^n
`lcoeff(p, x)`	Старший коэффициент полинома p по переменной x
`tcoeff(p, x)`	Младший коэффициент полинома p по переменной x