МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

Символьные вычисления в системах компьютерной математики

Учебно-методическое пособие для вузов

Составители: С. А. Ткачева,
П. В. Садчиков,
Л. В. Безручкина

Воронеж
2020

Утверждено научно-методическим советом математического факультета
2020 года протокол №

Рецензент: к.ф-м. н., доцент Савченко Г.Б.

Учебно-методическое пособие подготовлено на кафедре уравнений в частных производных и теории вероятностей математического факультета Воронежского государственного университета

Рекомендуется для студентов 4-5 курсов очной формы обучения математического факультета, обучающихся по направлениям 01.03.01 Математика, 02.03.01 Математика и компьютерные науки, 01.03.04 Прикладная математика и по специальности 01.05.01 Фундаментальные математика и механика

Система символьной математики Maxima

Maxima - еще одна программа для выполнения математических вычислений, символьных преобразований, а также построения разнообразных графиков. Сложные вычисления оформляются в виде отдельных процедур, которые затем могут быть использованы при решении других задач. Система Maxima распространяется под лицензией GPL и доступна как пользователям ОС Linux, так и пользователям MS Windows.

Система компьютерной математики Maxima -- настоящий ветеран среди программ этого класса. Она старше многих своих известных коммерческих собратьев по крайней мере на два десятка лет. Первоначально носившая имя Macsyma, она была создана в конце 1960-х годов в знаменитом Массачусетском технологическом институте и почти 20 лет (с 1982 по 2001) поддерживалась Биллом Шелтером (William Schelter), благодаря которому и приобрела свои замечательные качества и известность в научном мире. Подробности по истории системы, инсталляционный модуль (размером всего в 10 MB), документацию, исходный код и другую сопутствующую информацию можно найти на Web-узле пакета http://maxima.sourceforge.net. Выбор программы в меню **Пуск** vxmaxima позволит начать сеанс работы с этой программой.

При *старте программы* выводится некоторая информация о системе и "метка" (%i1).

wxMaxima 0.7.5 http://wxmaxima.sourceforge.net

Maxima 5.15.0 http://maxima.sourceforge.net

Using Lisp GNU Common Lisp (GCL) GCL 2.6.8 (aka GCL)

Distributed under the GNU Public License. See the file COPYING.

Dedicated to the memory of William Schelter.

The function bug_report() provides bug reporting information.

```
(%i1)
```

Это сообщение о том, что ядро Максимы только что запустилось (вместо нее, в зависимости от версии и конкретной сборки, (например: версия *Maxima* 5.31.2 (и выше)), может выводиться краткая информация о программе); и приглашение к вводу первой команды. Команда в Максиме — это любая комбинация математических выражений и встроенных функций, завершенная, в простейшем случае, точкой с запятой. После ввода команды и нажатия «Enter» (в более поздних версиях, например 5.31.2, ввод данных осуществляется с помощью комбинации клавиш «Shift»+«Enter») Maxima выведет результат и будет ожидать следующей команды:

```
(%i1)  (1/2+1/3+1/4)/(1/5+1/6+1/8);

(%o1)  130/59
```

Как видите, каждая ячейка имеет свою метку; эта метка — заключенное в скобки имя ячейки. Каждый ввод и вывод помечаются системой и затем могут быть использованы снова. Символ (%i1) используется для обозначения команд, введенных пользователем, а (%o1) - при выводе результатов вычислений. Ячейки ввода именуются как %i с номером (i от *input* — ввод), ячейки вывода — как %o с соответствующим номером (o от *output* —

вывод). Со знака `%` начинаются все встроенные служебные имена: чтобы, с одной стороны сделать их достаточно короткими и удобными в использовании, а с другой — избежать возможных накладок с пользовательскими именами, которые тоже часто удобно делать короткимиК примеру, внутренними именами **`%e`** и **`%pi`** обозначены общеизвестные математические постоянные; а через `%c` с номером обозначаются константы, используемые при интегрировании, для которых использование буквы «c» традиционно в математике. При вводе мы можем обращаться к любой из предыдущих ячеек по ее имени, подставляя его в любые выражения. Кроме того последняя ячейка вывода обозначается через `%`, а последняя ячейка ввода — через _. Это позволяет обращаться к последнему результату, не отвлекаясь на то, каков его номер.

```
(%i2) %+47/59;

(%o2) 3
```

Здесь `%+47/59` — то же самое, что `%o1+47/59`. Вывод результата вычисления не всегда нужен на экране; его можно заглушить, завершив команду символом `$` вместо `;`. Заглушенный результат при этом все равно вычисляется; как видите, в этом примере ячейки `%o1` и `%o2` доступны, хотя и не показаны (к ячейке `%o2` обращение идет через символ `%`, смысл которого расшифрован выше):

```
(%i1) sqrt(2)+3$
(%i2) 2*sqrt(2)+1$
(%i3) %-%o1;

(%o3) sqrt(2)-2
```

Каждую следующую команду не обязательно писать с новой строки; если ввести несколько команд в одну строчку, каждой из них все равно будет соответствовать свое имя ячейки. К примеру, здесь в строке после метки `%i1` введены ячейки от `%i1` до `%i4`; в ячейке `%i3` используются `%i1` и `%i2` (обозначенная как _ — предыдущий ввод):

```
(%i3) asin(1/2)$acos(1/2);%i1+_;%o1+%;

(%o4) %pi/3

(%o5) %pi/3+130/59

(%o6) %pi/3+260/59
```

В wxMaxima и TeXmacs последнюю или единственную команду в строке можно не снабжать завершающим символом `;` — это сработает так же, как если бы она была **завершена** `;` т. е. вывод заглушен не будет. Если вы выберете другой интерфейс, не забывайте ее добавлять.

Помимо использования имен ячеек, мы, естественно, можем и сами давать имена любым выражениям. По-другому можно сказать, что мы присваиваем значения переменным, с той разницей, что в виде значения такой переменной может выступать любое математическое выражение. Делается это с помощью двоеточия — знак равенства оставлен уравнениям, которые, учитывая общий математический контекст записи, проще и привычнее так читаются. Например:

```
(%i1)  equation:x^3-x=0$

(%i2)  solve(equation);
(%o2)  [x=-1,x=1,x=0]
```

В каком-то смысле двоеточие даже нагляднее в таком контексте, чем знак равенства: это можно понимать так, что мы задаем некое обозначение, а затем через двоеточие расшифровываем, что именно оно обозначает. После того, как выражение поименовано, мы в любой момент можем вызвать его по имени:

```
(%i3)  diff(equation,x);
(%o3)  3x^2-1=0

(%i4) solve([x^2+6*x+9], [x]);

(%o4) [x=-3]
```

Таким образом: *для инициализации процесса вычислений следует ввести команду, затем символ* `;` *(точка с запятой) и нажать клавишу* ***Enter (или Shift + Enter).*** *Если не требуется вывод полученной информации на экран, то вместо точки с запятой используется символ* ***$.*** *Обратиться к результату последней команды можно с помощью символа* `%`***.*** *Для повтора ранее введенной команды, скажем* `(%i2)`*, достаточно ввести два апострофа и затем метку требуемой команды, например,* `'(%i2) '`

Система Maxima обращает внимание на регистр введенных символов в именах встроенных констант и функций. Регистр букв важен при использовании переменных, например, Maxima считает x и X разными *переменными.*

Для стандартных математических констант используются следующие обозначения: `%e` для основания натуральных логарифмов, **%i** для мнимой единицы (квадратный корень из числа -1) и **%pi** для числа π.

Присваивание значения какой-либо переменной осуществляется с помощью знака : (двоеточие), а символ = (равно) используется при задании уравнений или подстановок.

```
(%i1) x:2;
(%o1)                                 2
(%i2) y:3;
(%o2)                                 3
(%i3) x + y;
(%o3)                                 5
```

Любое имя можно очистить от присвоенного ему выражения функцией `kill()`, и освободить занимаемую этим выражением память. Для этого нужно просто набрать `kill(name)`, где name — имя уничтожаемого выражения; причем это может быть как имя, назначенное вами, так и любая ячейка ввода или вывода. Точно так же можно очистить разом всю память и освободить все имена, введя `kill(all)`. В этом случае очистятся в том числе и все ячейки ввода-вывода, и их нумерация опять начнется с единицы. Когда же новый «сеанс» будет никак не связан с предыдущим, буду начинать нумерацию заново; это будет косвенным указанием сделать «`kill(all)`», если вы будете набирать примеры в Maxima, так как имена переменных и ячеек в таких «сеансах» могут повторяться. Для этого можно использовать основное меню. Кликаем ЛКМ по кнопке **Maxima** основного меню, переходим на строку **Очистить память** и еще раз кликнем ЛКМ. В этом меню можно также **Прервать** вычисления, **Перезапустить Maxima** и др. операции.

*Функция **kill** аннулирует присвоенные ранее значения переменных. Параметр `all` этой функции приводит к удалению значения всех переменных, включая метки `(%i)` и `(%o)`.*

```
(%i4) kill(x);
(%o4)                                done
(%i5) x + y;
(%o5)                               x + 3
(%i6) kill(all);
(%o0)                                done
(%i2) x + y;
(%o2)                               y + x
```

Для завершения работы с системой применяется функция **quit();**, а прерывание процесса вычислений осуществляется путем нажатия комбинации клавиш Ctrl- (после чего следует ввести :q для возврата в обычный режим работы), или команды в меню: **Maxima ->Прервать.** Можно использовать также команды: **Очистить память,** а также **Перезапустить Maxima.**

Работа с выражениями

При записи математических выражений могут использоваться четыре стандартные арифметрические операции (+, -, *, /) и операция возведения в степень (^, ^^ или **). Приоритет этих операций традиционен, для изменения порядка вычислений следует использовать круглые скобки. Кроме чисел выражения могут содержать результаты вычислений математических функций. Аргументы функций указываются в круглых скобках, например, запись sqrt(5) означает корень квадратный из числа 5. Если в результате расчета получается дробное выражение, то оно и выводится в виде обыкновенной дроби. Иррациональные числа, входящие в выражение, представляются в символьном виде.

Примеры работы программы с арифметическими операциями:

Сложение и умножение коммутирующие операции. Вычитание a- b представлено в Максима как сложение a+ (-b). Деление a/ b представлено в Максима как умножение, a * b ^ (-1).

```
(%i1) c + g + d + a + b + e + f;
(%o1)                g + f + e + d + c + b + a
(%i2) [op (%), args (%)];
(%o2)               [+, [g, f, e, d, c, b, a]]
(%i3) c * g * d * a * b * e * f;
(%o3)                        a b c d e f g
(%i4) [op (%), args (%)];
(%o4)               [*, [a, b, c, d, e, f, g]]
(%i5) apply ("+", [a, 8, x, 2, 9, x, x, a]);
(%o5)                       3 x + 2 a + 19
```

```
(%i6) apply ("*", [a, 8, x, 2, 9, x, x, a]);
                                 2  3
(%o6)                       144 a  x
```

op (операция), args(аргументы), apply(применить). Деление и возведение в степень не коммутирующие операции. Операции обозначаются как "/" и "^".

```
(%i1) [a / b, a ^ b];
                                a   b
(%o1)                          [-, a ]
                                b
(%i2) [map (op, %), map (args, %)];
(%o2)                [[/, ^], [[a, b], [a, b]]]
(%i3) [apply ("/", [a, b]), apply ("^", [a, b])];
                                a   b
(%o3)                          [-, a ]
                                b
```

```
(%i1) 17 + b - (1/2)*29 + 11^(2/4);
                                          5
(%o1)                   b + sqrt(11) + -
                                          2
(%i2) [17 + 29, 17 + 29.0, 17 + 29b0];
(%o2)                   [46, 46.0, 4.6b1]
```

Для раскрытия скобок используется функция **expand**. Команда **ev** позволит получить численное значение выражения. Ее первый аргумент есть вычисляемое выражение, а второй - опция **numer**. Напомним, что символ **%** означает результат предыдущего вычисления.

```
(%i3) 1/100+1/101;

(%o3) 201/10100

(%i4) (1+sqrt(2))^5;

(%o4) (sqrt(2)+1)^5

(%i5) expand(%i14);

(%o5) 29*sqrt(2)+41

(%i6) ev(%i15,numer);

(%o6) 82.01219330881975
```

Допускается более удобная форма функции **ev,** требующая указания только ее аргументов:

```
(%i7) 29*sqrt(2) + 41, numer;
(%o7)                       82.01219330881976
```

По умолчанию результат содержит 16 значащих цифр. Для вывода числа в экспоненциальной форме используется функция **bfloat**:

```
(%i8) bfloat(%i7);
(%o8)              ev(8.201219330881976b1,numer)
```

Запись mbn есть сокращенная форма выражения $m*10^n$. Количество значащих цифр в представлении числа определяется специальной переменной **FPPREC.** Увеличение ее значения приводит к возрастанию точности результата, например,

```
(%i9) fpprec;
(%o9)                      16

(%i10) fpprec:100;
(%o10)                      100

(%i11) ''(%i7);
(%o11)                      8.20121933088197607657604923686248525#
03077530516721861648463104778270702443495 4#
8350683851114422615155B1
```

Символ # в конце выводимой строки означает, что число не уместилось на одной строке и его оставшаяся часть переносится на следующую. В последнем примере мы использовали повторение ранее введенной команды "(%i7);

Пример.

Вычислим число pi с точностью 200 знаков после запятой:

```
(%i13) %pi, numer;
(%o13)              3.141592653589793

(%i14) fpprec:200;
(%o14)                      200

(%i15) bfloat(%pi);
(%o15)          3.14159265358979323846264338327950288 4#
19716939937510582097494459230781640628620899#
86280348253421170679821480865132823066470938#
44609550582231725359408128481117450284102701#
93852110555964462294895493038 19B0
```

Функции в Maxima

В Maxima доступны прямые и обратные тригонометрические функции: sin (синус), cos (косинус), tan (тангенс), cot (котангенс), asin (арксинус), acos (арккосинус), atan (арктангенс), acot (арккотангенс). Кроме них имеются две менее известные функции - секонс (sec x = 1/cos x) и косеконс (csc x = 1/sin x); log(x) – натуральный логарифм; abs(x) – модуль; sqrt(x) – корень квадратный; mod(x) – остаток от деления, min(x1, x2,…, xn) - минимальный элемент из списка; max(x1, x2,…, xn) - максимальный элемент из списка.

```
(%i22)      sin(%pi /6)^2 +cos(%pi /6)^2;
(%i22)      1

(%i23)      sec(%pi /3);
(%o23)                                  2

(%i24)      asin(1/2);
                                    % pi
(%o24)                              ---
                                    6
```

К сожалению, Maxima не может в символьном виде вычислить значения обратных тригонометрических функций в некоторых точках:

```
(%i25)     asin(sqrt(3)/2);
                                SQRT(3)
(%o25)                    ASIN(-------)
                                2
```

```
(%i26)      %, expand;
                          SQRT(3)
(%o26)                    ASIN(-------)
                          2
```

Для вычисления натурального логарифма используется функция `log`:

```
(%i27)      log(%e^2);
(%o27)                                2
(%i28)      5*log(a)+6*log(b);
(%o28)                       6*log(b)+5*log(a)
(%i29)      logcontract(%);

(%o29)      log(a^5*b^6)
```

Алгебраические преобразования

Особо важную роль играет способность системы Maxima производить разнообразные символьные преобразования алгебраических выражений. Следующий пример демонстрирует раскрытие скобок в них:

```
(%i1)      p1:x^2-1;

(%o1)                           x^2-1

(%i2)      p2:x-1;
(%o2)                                x - 1

(%i3)      expand(p1*p2);

(%o3)                            x^3-x^2-x+1
(%i4)      expand((p1+p2)^2);

(%o4)                       x^4+2*x^3-3*x^2-4*x+4
```

Функцию **divide** можно использовать для нахождения частного и остатка от деления одного многочлена на другой:

```
(%i5)      divide(p1*p2, p1);
(%o5)                                [x - 1, 0]
```

Функция **gcd** определяет наибольший общий делитель многочленов, а **factor** осуществляет разложение многочлена на множители:

```
(%i6)      gcd(x^3-1, x^2-1, x-1);
(%o6)         x - 1

(%i7)      factor(x^8-1);

(%o7)           (x-1)*(x+1)*(x^2+1)*(x^4+1)
```

Подстановка какого-либо выражения вместо переменной осуществляется при помощи операции =. Например, заменим все вхождения x в выражении на 5/z:

```
(%i8)      x^4+3*x^3-2*x, x=5/z;

(%o8)                         -10/z+375/z^3+625/z^4
```

Функция **ratsimp** выносит за скобки наибольший общий делитель:

```
(%i9)      ratsimp(%);
(%o9) -(10*z^3-375*z-625)/z^4
```

Используя функцию **assume** (to assume - допускать), можно при вычислениях учитывать дополнительные условия, задаваемые неравенствами:

```
(%i10) sqrt(x^2);
(%o10)                                ABS(x)
(%i11) assume(x<0);
(%o11)                                [x < 0]
(%i12) sqrt(x^2);
(%o12)                                 - x
```

Функция **forget** (to forget - забывать) снимает все ограничения, наложенные при помощи assume:

```
(%i13) forget(x<0);
(%o13)                                [x < 0]
(%i14) sqrt(x^2);
(%o14)                                ABS(x)
```

Maxima легко оперирует тригонометрическими выражениями. Так, функция **trigexpand** использует формулы преобразования сумм двух углов для представления введенного выражения в как можно более простом виде:

```
(%i15) sin(u+v)*cos(u)^3;


(%o15)              cos(u)^3*sin(v+u)

(%o14) (sin(v+4*u)+sin(v-2*u))/8+(3*sin(v+2*u)+3*sin(v))/8trigexpand(%);

(%o16)       cos(u)^3*(cos(u)*sin(v)+sin(u)*cos(v))
```

Функция **trigreduce** преобразует тригонометрическое выражение к сумме элементов, каждый из которых содержит только единственный sin или cos:

```
(%i17) (sin(v+4*u)+sin(v-2*u))/8+(3*sin(v+2*u)+3*sin(v))/8trigreduce(%);

(%o17) (sin(v+4*u)+sin(v-2*u))/8+(3*sin(v+2*u)+3*sin(v))/8
```

Функции **realpart** и **imagpart** возвращают действительную и мнимую часть комплексного выражения:

```
(%i18) z1:-3+%i*4;
(%o18)                          4 %i - 3

(%i19) z2:4-2*%i;
(%o19)                          4 - 2 %i

(%i20) z1*z2;
(%o20)           (4-2*%i)*(4*%i-3)

(%i21) expand(%);
(%o21)                                22 %i - 4

(%i22) realpart(%);
(%o22)                                     - 4

(%i23) imagpart(%);
(%o23)                                      22
```

Следующие примеры иллюстрируют присвоения в выражениях:

```
(%i1) p(x):= x^2+3*x-8;
                                    2
(%o1)                    p(x) := x  + 3 x - 8

(%i2) p(2*x);
                                    2
(%o2)                             4 x  + 6 x - 8
(%i3) factor(p(2*x-4));
                                   2
(%o3)                        2 (2 x  - 5 x - 2)
(%i4) f(x):=sin(x);
(%o4)                                    f(x) := SIN(x)
(%i5) g(x):=log(x);
(%o5)                                    g(x) := LOG(x)
(%i6) f(g(x));
(%o6)                                SIN(LOG(x))
(%i7) g(f(x));
(%o7)                                    LOG(SIN(x))
```

Оператор факториала. Переменные factlim, minfactorial, и factcomb управляют упрощением выражений, содержащих факториалы.

```
(%i1) factlim : 10;
(%o1)                                10
(%i2) [0!, (7/2)!, 4.77!, 8!, 20!];
                105 sqrt(%pi)
(%o2)    [1, -------------, 81.44668037931199, 40320, 20!]
```

В случае комплексного числа упрощают факториал после оценки операнда.

```
(%i1) [(%i + 1)!, %pi!, %e!, (cos(1) + sin(1))!];
(%o1)       [(%i + 1)!, %pi!, %e!, (sin(1) + cos(1))!]
(%i2) ev (%, numer, %enumer);
(%o2) [(%i + 1)!, 7.188082728976037, 4.260820476357,
                                                 1.227580202486819]
```

Восклицательный знак, стоящий после своего аргумента (т. е. постфиксный оператор), традиционно обозначает факториал. Двумя восклицательными знаками обозначен полуфакториал. Функции `abs(x)` и `signum(x)` возвращают модуль и знак числа. А функции `max(x1,...,xn)` и `min(x1,...,xn)` — соответственно максимальное и минимальное из заданных чисел. При этом факториал от натурального числа (и нуля) автоматически упрощается до натурального же числа:

```
(%i1) 0!;6!;28!;
(%o1) 1
(%o2) 720
(%o3) 304888344611713860501504000000
```

Точно так же и модуль определен для всех комплексных чисел. Минимум, максимум и знак определены только для действительных чисел, так как комплексные числа общего вида, как известно, между собой несравнимы.

```
(%i4)  abs(-x);
(%o4)  |x|

(%i5)  max(x,x+1,x-abs(a));
(%o5)  x+1
```

Справка о той или иной функции выводится по команде ***describe*** *(имя функции). При работе в графической оболочке wxMaxima, можно воспользоваться пунктом меню* ***help.*** *Процедура* ***example****(имя функции) демонстрирует примеры использования функции.*

Операции математического анализа

Maxima может вычислять производные и интегралы, раскладывать функции в ряды Тейлора, вычислять пределы и находить точные решения обыкновенных дифференциальных уравнений.

```
integrate               next            from            diff
in                      at              limit           sum
for                     and             elseif          then
else                    do              or              if
unless                  product         while           thru
step
```

Функция нахождения предела может принимать три различных варианта списка аргументов. Эта функция вполне соответственно ее действию: `limit`; и ее вызов выглядит как `limit(выражение, переменная, точка)`, то есть то, что в математической записи выглядит как $\lim_{x\to a} f(x)$, в контексте Maxima запишется как `limit(f(x), x, a)`:

```
(%i1)  limit((x^2-1)/(2*x^2-x-1),x,1);
(%o1)  2/3
(%i2)  limit(((1+m*x)^n-(1+n*x)^m)/x^2,x,0);
(%o2)  -m*(m-n)*n/2
(%i3)  limit(((x^n-a^n)-n*a^(n-1)*(x-a))/(x-a)^2,x,a);
(%o3)  n*(a^n*n-a^n)/(2*a^2)
```

Maxima может искать пределы не только в конечных точках, но и на бесконечности. Среди стандартных обозначений программы существуют универсальные названия для разных бесконечностей: плюс-бесконечность записывается через `inf` (от слова *infinity*, как нетрудно догадаться), минус-бесконечность — через `minf` (от *minus infinity*); для комплексных чисел бесконечность, как известно, одна, и она (комплексная бесконечность) обозначается полным словом `infinity`. При работе с пределами все три обозначения могут как использоваться при вводе, так и возникать в виде найденного значения предела; отдельно здесь надо отметить один момент касательно работы с интерфейсом к Maxima в редакторе TeXmacs: символы `inf` и `minf` при выводе здесь отображаются в своей традиционной математической нотации, то есть как ∞ и $-\infty$; символ вместо `inf` можно, кроме того, использовать еще и при вводе.

(%i1) $\lim\left(\frac{\sqrt{x}+\sqrt[3]{x}+\sqrt[4]{x}}{\sqrt{2x+1}}, x, \infty\right)$

(%o1) $\frac{1}{\sqrt{2}}$

(%i2) $\lim\left(\sqrt{x+\sqrt{x+\sqrt{x}}}-x, x, \infty\right)$

(%o2) $-\infty$

Второй вариант вызова функции limit() — это расширенная версия первого: limit(выражение, переменная, точка, направление), для поиска односторонних пределов. Для предела справа в качестве «направления» указывается plus, для предела слева — minus:

(%i1) $\lim\left(\frac{\mathrm{abs}(\sin(x))}{x}, x, 0, \mathrm{plus}\right)$

(%o1) 1

(%i2) $\lim\left(\frac{\mathrm{abs}(\sin(x))}{x}, x, 0, \mathrm{minus}\right)$

(%o2) -1

Кроме упомянутых выше бесконечностей, на выходе возможно появление и еще двух обозначений, на случай, если заданный предел не существует: **ind** (от слова *indefinite* — *неопределенный*) и **und** (от слова *undefined* — опять же *неопределенный*). В документации первое из этих обозначений описано как *indefinite but bounded* (*не определен, но ограничен*), что дает предположить, что функция, не имеющая предела, при этом ограничена либо в окрестности предельной точки, либо на всей прямой.

(%i1) $\lim\left(\sin\left(\frac{1}{x}\right), x, 0\right); \lim\left(\tan\left(\frac{1}{x}\right), x, 0\right)$

(%o1) ind
(%o2) und

```
(%i3) limit(a: 1/(1-e^(1/x)), x, 0)
(%o3) und
(%i4) limit(a, x, 0, plus); limit(a, x, 0, minus); limit(a, x, inf); limit(a, x, minf)
(%o4) 0
(%o5) 1
(%o6) -inf
(%o7) inf
(%i8) limit(a: atan(e^(1/x)), x, 0)
(%o8) und
(%i9) limit(a, x, 0, plus); limit(a, x, 0, minus); limit(a, x, inf); limit(a, x, minf)
(%o9) %pi/2
(%o10) 0
(%o11) %pi/4
(%o12) %pi/4
```

Первая функция имеет конечные односторонние пределы в нуле, а вторая ограничена вообще на всей оси; наличие любого из этих символов в качестве вывода дает нам понять, что искомого предела не существует.

Функция **`limit`**`()` в третьем варианте — `limit(выражение)` — предназначена уже не для поиска собственно пределов, а для упрощения выражений, содержащих символы `inf` и `minf`:

```
(%i1) limit(1/inf)
(%o1) 0
```

Выражения такого рода могут возникать, к примеру, при подстановках в формулы результатов вычисления каких-то других пределов или интегралов.

Для вычисления пределов используется функция limit: Для вычисления односторонних пределов используется дополнительный параметр, принимающий значение plus для вычисления предела справа и minus - слева.

Дифференцирование и интегрирование

Функция diff, diff(выражение, переменная)- возвращает производную от «выражения» по заданной переменной; с одним, diff(выражение) — полный дифференциал заданного выражения. Другими словами, запись diff(f, x) равнозначна математическому обозначению df/dx, а diff(f) — df. К примеру, diff(f, x, 3) означает d^3f/dx^3, а diff(f, x, 1, y, 2, z, 1) — $d^4f/dxdy^2dz$. Единственный флаг, имеющий прямое отношение к самой функции diff — это флаг derivabbrev, который влияет на отображение производных в ячейках вывода Maxima. По умолчанию он равен false, и производные обозначаются в виде дробей с буквой d; если

же его выставить в true, производные будут отображаться в сокращенном виде, с переменными дифференцирования записанными в виде индексов:

```
(%i1) diff(f(x)^2,x,4)
```

$$(\%o1)\quad 2f(x)\left(\frac{d^4}{dx^4}f(x)\right)+8\left(\frac{d}{dx}f(x)\right)\left(\frac{d^3}{dx^3}f(x)\right)+6\left(\frac{d^2}{dx^2}f(x)\right)^2$$

```
(%i2) derivabbrev: true$ '%i1
```

$$(\%o3)\quad 2f(x)\left(f(x)\right)_{xxxx}+8\left(f(x)\right)_x\left(f(x)\right)_{xxx}+6\left(\left(f(x)\right)_{xx}\right)^2$$

Кроме того, функция **diff** используется еще и для обозначения производных в дифференциальных уравнениях.

Основная функция интегрирования называется **integrate** и имеет два варианта вызова: для нахождения неопределенного и определенного интегралов. Первый выглядит как *integrate(выражение, переменная)*,

второй — как *integrate(выражение, переменная, нижний-предел, верхний-предел):*

```
(%i1) integrate(sqrt((exp(x)-1)/(exp(x)+1)),x)
```

$$(\%o1)\quad \log\left(2\sqrt{e^{2x}-1}+2e^x\right)+\arcsin\left(e^{-x}\right)$$

```
(%i2) integrate(cos(x)*cos(2*x)*cos(3*x),x)
```

$$(\%o2)\quad \frac{\sin(6x)}{24}+\frac{\sin(4x)}{16}+\frac{\sin(2x)}{8}+\frac{x}{4}$$

```
(%i3) integrate((x^(2*n-1))/(x^n+1),x)
```

$$(\%o3)\quad \frac{e^{n\log(x)}}{n}-\frac{\log\left(e^{n\log(x)}+1\right)}{n}$$

```
(%i4) integrate(x^2*sqrt(a^2-x^2),x,0,a)
Is a positive, negative, or zero?p
```

$$(\%o4)\quad \frac{\pi a^4}{16}$$

Обратите внимание еще на один момент в ячейках `%i4`–`%o4`. Когда в выражении используется какой-либо независимый символ, результат, вообще говоря, может зависеть от значения этого символа. Если при этом о возможных значениях символа ничего не известно, то Maxima задаст вам один или несколько вопросов об этом значении, и решение будет искать в зависимости от ваших ответов на них. Так, в этом примере значение определенного интеграла напрямую зависит от знака параметра a:

```
(%i5) integrate(x^2*sqrt(a^2-x^2),x,0,a)
Is a positive, negative, or zero?n
```

$$(\%o5)\quad -\frac{\pi a^4}{16}$$

Кроме обычных определенных интегралов Maxima умеет искать также и несобственные интегралы, то есть такие, у которых неограничена либо область

интегрирования, либо подынтегральная функция; и делается это все той же функцией `integrate`:

(%i1) integrate$\left(\frac{1}{1+e^{x}}, x, 1, \infty\right)$

(%o1) $\log(e+1)-1$

(%i2) integrate$\left(\frac{1}{\sqrt{x}}, x, 0, 1\right)$

(%o2) 2

В случае, если искомый интеграл не сходится, будет выдано сообщение об ошибке, говорящее о том, что интеграл расходящийся:

(%i3) integrate$\left(\frac{1}{x^2}, x, 0, 1\right)$

```
Integral is divergent
 -- an error.  Quitting.  To debug this try debugmode(true);
```

В случае, если интеграл не может быть найден, он либо целиком возвращается в несовершенном виде, либо упрощается частично и на выходе получается некоторая формула, включающая в несовершенном виде интеграл той части подынтегрального выражения, которую проинтегрировать не удалось:

(%i1) integrate$\left(\frac{1}{x^4-4x^3+2x^2-7x-4}, x\right)$

(%o1) $\frac{\log(x-4)}{73}-\frac{\int \frac{x^2+4x+18}{x^3+2x+1}\,dx}{73}$

Для нахождения производной используется функция ***diff****, первым аргументом которой является функция, вторым - переменная, по которой производится дифференцирование, и третьим (необязательным) - порядок производной:*

(%i1) `f:(x-2*sqrt(x))/x^2;`

(%o1) $\frac{x-2\sqrt{x}}{x^2}$

(%i2) `diff(f, x);`

(%o2) $\frac{1-\frac{1}{\sqrt{x}}}{x^2}-\frac{2(x-2\sqrt{x})}{x^3}$

```
(%i3)  expand(''%i2);
(%o3)  3/x^(5/2) - 1/x^2

(%i4)  g:x^6;
(%o4)  x^6

(%i5)  diff(g, x, 1);
(%o5)  6 x^5

(%i6)  diff(g, x, 4);
(%o6)  360 x^2
```

При вычислении кратных производных по нескольким переменным после указания функции перечисляются переменные дифференцирования с указанием соответствующих кратностей, например,

```
(%i7)  diff(x^6*y^3, x, 4, y, 2);
(%o7)  2160 x^2 y
```

Функция **integrate** позволяет вычислять интегралы. Для нахождения неопределенного интеграла после функции указывается единственный аргумент - переменная интегрирования:

```
(%i8)  f:x^2/(4*x^6+1);
(%o8)  x^2/(4 x^6 + 1)

(%i9)  integrate(f, x);
(%o9)  atan(2 x^3)/6
```

Для нахождения определенного интеграла следует указать дополнительные агрументы - пределы интегрирования:

```
(%i16)integrate(x^2, x, 0, 6);
(%o16)                                  72

(%i17)integrate(sin(x), x, 0, %pi);
(%o17)                                  2

(%i18)integrate(integrate(x*y, x, 1, 3), y, 0, 4);
```

```
(%o18)                                32
```

Maxima допускает задание и бесконечных пределов интегрирования. Для обозначения бесконечности используется переменная `INF` (`inf`):

```
(%i19)integrate(1/x^2, x, 1, inf);
(%o19)                            1

(%i20)integrate(1/(1+x^2), x, -inf, inf);
(%o20)                                  %pi

(%i21)integrate(1/x, x, 0, inf);

Integral is divergent  -- an error.
Quitting. To debug this try DEBUGMODE(TRUE);)
```

В последнем примере система сообщила о невозможности вычисления интеграла, т. к. он расходится (is divergent). Перечисленные выше операции математического анализа могут быть вычислены с помощью меню **Анализ** и далее соответствующие команды меню (Integrate), (Differentiate), (Find Limit).

Пример. Исследуем на непрерывность функцию arctg(1/(x-4)). Эта функция не определена в точке x = 4. Вычислим пределы справа и слева:

```
(%i5)  limit(atan(1/(x-4)), x, 4, plus);
(%o5)  %pi/2

(%i6)  limit(atan(1/(x-4)), x, 4, minus);
(%o6)  -%pi/2
```

Как видим, точка x = 4 является точкой разрыва I рода для данной функции, так как существуют пределы слева и справа, равные -PI/2 и PI/2 соответственно.

Задания

1. Вычислите первую производную функции $tg^2(x^4 - 2)$.
2. Найдите предел при x -> 0 функции (3x - sin x)/tg 2x.
3. Найдите одну из первообразных функции $\cos^2 x$.

Нахождение суммы ряда. Разложение в ряд Тейлора

Для вычисления конечных и бесконечных сумм следует записать сумму в символьном виде, после чего упростить полученное выражение:

```
(%i1)  sum(1/n^2,n,1,inf);
```

$$(\%o1)\quad \sum_{n=1}^{\infty} \frac{1}{n^2}$$

```
(%i2)  %,simpsum;
(%o2)  %pi^2/6
```

Maxima может находить разложение функций в ряд Тейлора.

Пример. Разложить в ряд Тейлора функцию g(x)=ln x в точке x=1. Получим многочлен Тейлора четвертого порядка в точке x=1:

```
(%i3)  g:log(x);
(%o3)  log(x)
```

```
(%i4)  taylor(g,x,1,4);
(%o4)  x-1-(x-1)^2/2+(x-1)^3/3-(x-1)^4/4+...
```

Формат задания: **taylor** *(expr, [x_1, x_2, ...], a, n)* (**taylor**(функция, переменные, точка *x=a, до какой степени параметра разложения вычисляется ряд*).
taylor *(expr, [x_1, x_2, ...], [a_1, a_2, ...], [n_1, n_2, ...])* ,
taylor *(expr, [x_1, a_1, n_1], [x_2, a_2, n_2], ...)* – разложение в ряд Тейлора функции нескольких переменных в окрестностях точек *x1=a1, x2=a2,.., до порядка n1,n2,…*
Примеры.

```
(%i5) taylor (sqrt (x + 1), x, 0, 5);
                      2    3      4      5
                     x    x    5 x    7 x
(%o5)/T/      1 + - - -- + -- - ---- + ---- + . . .
                     2    8    16   128    256
```

```
(%i6) taylor (sin (y + x), x, 0, 3, y, 0, 3);
                3                 2
               y                 y
(%o6)/T/ y - -- + . . . + (1 - -- + . . .) x
               6                 2

                    3                       2
                 y   y            2      1   y            3
          + (- - + -- + . . .) x  + (- - + -- + . . .) x  + . . .
                 2   12                   6   12
(%i7) taylor (sin (y + x), [x, y], 0, 3);
                     3        2      2      3
                    x  + 3 y x  + 3 y  x + y
(%o8)/T/   y + x - ------------------------ + . . .
                              6
```

Решение уравнений

Уравнения и системы уравнений решаются в Maxima функцией `solve`. Но прежде чем рассмотреть ее подробнее, рассмотрим списки, или векторы в Maxima; поскольку именно в виде списков solve возвращает корни, да и принимает параметры в случае решения системы уравнений, а не одного уравнения.

Синтаксис списков в Maxima весьма прост; это перечисление элементов в квадратных скобках: `[элемент1, элемент2, …, элементN]`. Особенность — не в синтаксисе. Основное достоинство списков в том, что их элементами могут быть совершенно любые выражения: символы, арифметические выражения, вызовы функций, присвоения, уравнения, другие списки. Функция `solve` в своем простейшем варианте, для решения одиночного уравнения, в качестве аргументов никаких списков не принимает (а принимает либо уравнение и символ, относительно которого его надо решать, либо только уравнение, если символ в нем всего один). А вот в качестве результата она уже и в таком варианте возвращает список, состоящий из всех корней заданного уравнения:

```
(%i1) eq: (x+1)/(x^2+1)=x^2/(x+2)$solve(eq);
```

(%o2) $$\left[x=-\frac{\sqrt{5}-1}{2},\ x=\frac{\sqrt{5}+1}{2},\ x=-\frac{\sqrt{7}\,\%i+1}{2},\ x=\frac{\sqrt{7}\,\%i-1}{2}\right]$$

Как видите, функция `solve` находит все комплексные корни уравнения, а не только действительные.

К элементу списка можно обратиться с помощью тех же квадратных скобок, указав в них номер элемента после имени списка. Напомню, что равенство, переданное в качестве дополнительного параметра функции ev, означает подстановку переменной в вычисляемое выражение. Вот так мы можем осуществить проверку решения, подставив корень из выданного списка в исходное уравнение:

```
(%i3) eq, %[1]
```

(%o3) $$\frac{1-\frac{\sqrt{5}-1}{2}}{\frac{\left(\sqrt{5}-1\right)^2}{4}+1}=\frac{\left(\sqrt{5}-1\right)^2}{4\left(2-\frac{\sqrt{5}-1}{2}\right)}$$

```
(%i4) ratsimp(%)
```

(%o4) $$\frac{\sqrt{5}-3}{\sqrt{5}-5}=\frac{\sqrt{5}-3}{\sqrt{5}-5}$$

Точно таким же образом можно обратиться и к любому другому элементу списка:

```
(%i5) eq, %o2[2], ratsimp; eq, %o2[3], ratsimp; eq, %o2[4], ratsimp
```

(%o5) $$\frac{\sqrt{5}+3}{\sqrt{5}+5}=\frac{\sqrt{5}+3}{\sqrt{5}+5}$$

(%o6) $-1=-1$

(%o7) $-1=-1$

Вообще говоря, в качестве первого аргумента функции `solve` можно задавать не только уравнение, а вообще любое выражение. Возможность такой записи позволяет, к примеру, легко найти критические точки любой непрерывной функции (а заодно и вычислить значения функции в этих точках):

(%i1) $f:\frac{1+x-x^2}{1+x+x^2}$$

(%i2) solve(diff(f,x))

(%o2) $[x=-2,x=0]$

(%i3) f,%[1];f,%th(2)[2]

(%o3) $-\frac{5}{3}$

(%o4) 1

В этом примере есть еще два важных момента. Первый — функция `%th()`. Она, как видно из контекста, вызывается как `%th(n)` и возвращает *n*-ю с конца ячейку вывода. Перейдем теперь к решению систем уравнений. Для этого существует такой вариант записи: `solve([уравнение1, уравнение2, …], [переменная1, переменная2, …])`; либо сокращенный, аналогично варианту для одиночного уравнения: если количество уравнений и количество неизвестных равны, список неизвестных можно не писать: `solve([уравнение1, уравнение2 …])` (не забудьте квадратные скобки, иначе Maxima примет его за вариант с одним уравнением).

(%i1) solve($[x^2+y^2=2,x+y=1]$)

(%o1) $\left[\left[y=-\frac{\sqrt{3}-1}{2},x=\frac{\sqrt{3}+1}{2}\right],\left[y=\frac{\sqrt{3}+1}{2},x=-\frac{\sqrt{3}-1}{2}\right]\right]$

Здесь возвращается список из нескольких списков, каждый из которых соответствует одному решению системы. В качестве подстановок можно использовать как такие списки целиком (например, в данном контексте, `%o1[1]`), так и отдельные их элементы (например, `%o1[1][1]`).

В случае, когда уравнений меньше, чем неизвестных, `solve` поступит точно так же, как и в случае одного уравнения с несколькими символами: все неуказанные будет воспринимать как параметры:

(%i1) solve($[x^2+y^2=a^2,x+y=2a+1],[x,y]$)

(%o1) $\left[\left[x=-\frac{\sqrt{-2a^2-4a-1}-2a-1}{2},y=\frac{\sqrt{-2a^2-4a-1}+2a+1}{2}\right],\left[x=\frac{\sqrt{-2a^2-4a-1}+2a+1}{2},y=-\frac{\sqrt{-2a^2-4a-1}-2a-1}{2}\right]\right]$

Если `solve` не находит точных решений, она может, как и `integrate`, вернуть уравнение или систему уравнений в некотором упрощенном виде, а может и самостоятельно попытаться решить систему численно:

(%i1) $eqs: [4x^2 - y^2 = 12, xy - x = 2]$$

(%i2) solve(eqs)

(%o2) [[y = 2, x = 2], [y = -0.15356757100197, x = -1.733751846381093], [y = 3.60800322187\
0287 i + 0.076783785237878, x = -0.5202594388652 i - 0.13312403573587], [y = 0.07678378523\
7878 - 3.608003221870287 i, x = 0.5202594388652 i - 0.13312403573587]]

В таком случае, если вам все же нужны точные значения корней (в аналитической записи), либо если они не найдены даже в числах, можно попробовать решить уравнения по очереди, выражая одно неизвестное через другое:

(%i3) solve(eqs[2], y)

(%o3) $\left[y = \frac{x+2}{x}\right]$

(%i4) eqs[1], %

(%o4) $4x^2 - \frac{(x+2)^2}{x^2} = 12$

(%i5) solve(%)

Maxima может решать уравнения и системы алгебраических уравнений с помощью функции ***solve****. Равная нулю правая часть уравнения может быть опущена*:

```
(%i1) solve (x^2=1, x);
(%o1)                                [x = - 1, x = 1]

(%i2) solve(x^2-1,x);
(%o2)                                [x = - 1, x = 1]

(%i3) solve(log(x+3)=1, x);
(%o3)                                  [x=%e-3]
```

При решении тригонометрических уравнений выдается только одно из бесконечного множества возможных решений:

```
(%i4) solve(sin(x)-1, x);
SOLVE is using arc-trig functions to get
a solution. Some solutions will be lost.
(%o4)                          [x=%pi/2]
```

В следующем примере функция solve используется для решения системы из трех уравнений с тремя неизвестными:

```
(%i5) s:[x+y+z=3, x+2*y-z=2, x+y*z+z*x=3];
(%o5)              [z + y + x = 3, - z + 2 y + x = 2,
                y z + x z + x = 3]
(%i6) solve(s, [x,y,z]);
(%o6)                 [[x = 1, y = 1, z = 1],
                  [x = 7, y = - 3, z = - 1]]
```

Если уравнение не имеет решений на множестве действительных чисел, то Maxima ищет решения среди комплексных чисел:

```
(%i7) solve(x^2+1,x);
(%o7)                              [x = - %i, x = %i]
```

Задание. Решите уравнение *ln(tgx)=0.*

Нахождение приближенных значений корней алгебраических уравнений

В Maxima можно находить приближенные значения корней алгебраических уравнений используя команды меню Maxima или встроенные функции **allroots** *(expr),* **realroots** *(expr, bound),* **find_root** *(expr, x, a, b)*

Пример 1. Найти все корни уравнения: *(1 + 2*x)^3 = 13.5*(1 + x^5);*

```
(%i1) eqn: (1 + 2*x)^3 = 13.5*(1 + x^5);
                              3          5
(%o1)                (2 x + 1)  = 13.5 (x  + 1)
(%i2) soln: allroots (eqn);
(%o2) [x = .8296749902129361, x = - 1.015755543828121,

x = .9659625152196369 %i - .4069597231924075,

x = - .9659625152196369 %i - .4069597231924075, x = 1.0]
(%i3) for e in soln
        do (e2: subst (e, eqn), disp (expand (lhs(e2) - rhs(e2))));
                       - 3.5527136788005E-15

                      - 5.32907051820075E-15

          4.44089209850063E-15 %i - 4.88498130835069E-15

         - 4.44089209850063E-15 %i - 4.88498130835069E-15

                        3.5527136788005E-15

(%o3)                           done
(%i4) polyfactor: true$
(%i5) allroots (eqn);
(%o5) - 13.5 (x - 1.0) (x - .8296749902129361)

                              2
 (x + 1.015755543828121) (x  + .8139194463848151 x

 + 1.098699797110288)
```

Пример 2. Находим только вещественные корни уравнения: *-1 - x + x^5=0*

```
(%i1) realroots (-1 - x + x^5, 5e-6);
                                 612003
(%o1)                       [x = ------]
                                 524288
(%i2) ev (%[1], float);
(%o2)                    x = 1.167303085327148
(%i3) ev (-1 - x + x^5, %);
(%o3)                   - 7.396496210176905E-6
(%i1) realroots (expand ((1 - x)^5 * (2 - x)^3 * (3 - x)), 1e-20);
(%o1)                    [x = 1, x = 2, x = 3]
(%i2) multiplicities;
(%o2)                          [5, 3, 1]
```

Пример 3. Найти значение всех корней уравнения: $\sin x - x = 0$ на отрезке $[0.1, \pi]$

```
(%i1) f(x) := sin(x) - x/2;
                                          x
(%o1)                  f(x) := sin(x) - -
                                          2
(%i2) find_root (sin(x) - x/2, x, 0.1, %pi);
(%o2)                   1.895494267033981
(%i3) find_root (sin(x) = x/2, x, 0.1, %pi);
(%o3)                   1.895494267033981
(%i4) find_root (f(x), x, 0.1, %pi);
(%o4)                   1.895494267033981
(%i5) find_root (f, 0.1, %pi);
(%o5)                   1.895494267033981
(%i6) find_root (exp(x) = y, x, 0, 100);
                              x
(%o6)           find_root(%e  = y, x, 0.0, 100.0)
(%i7) find_root (exp(x) = y, x, 0, 100), y = 10;
(%o7)                   2.302585092994046
(%i8) log (10.0);
(%o8)                   2.302585092994046
```

Рассмотрим более подробно методы построения приближенных решений.

1. Метод половинного деления

Рассмотрим следующую задачу: дано нелинейное уравнение $f(x)=0$, необходимо найти корень уравнения, принадлежащий интервалу $[a,b]$, с заданной точностью ε.

Реализация метода половинного деления в виде функций **Maxima** представлена в следующем примере. Функция *bisect*, в которую передается выражение f, определяющее уравнение, которое необходимо решить:

```
(%i1) bisect(f,sp,eps):=block([a,b],a:sp[1],b:sp[2],p:0,
      while abs(b-a)>eps do (
        p:p+1, c:(a+b)/2,
        fa:float(subst(a,x,f)), fc:float(subst(c,x,f)),
        if fa·fc<0 then b:c else a:c
      ),
      float(c));
```

(%o1) $\text{bisect}(f,sp,eps):=\text{block}\left([a,b],a:sp_1,b:sp_2,p:0,\text{while } |b-a|>eps \text{ do } \left(p:p+1,c:\frac{a+b}{2},fa:\text{float}(\text{subst}(a,x,f)),fc:\text{float}(\text{subst}(c,x,f)),\text{if } fa\,fc<0 \text{ then } b:c \text{ else } a:c\right),\text{float}(c)\right)$

Последовательность команд, организующая обращение к *bisect* и результаты вычислений

```
(%i7) f:exp(-x)-x$
      a:-1$ b:2$ eps:0.000001$ xrez:bisect(f,[-2,2],eps)$
      print("Решение  ",xrez,"Невязка  ",subst(xrez,x,f))$
```

Решение 0.567143440246582 *Невязка* $-2.348157265297246\,10^{-7}$

В представленном примере решается уравнение $e^{-x}-x=0$. Поиск корня осуществляется на отрезке [-2,2] с точностью до 0,0000001.

2. Метод простых итераций

В ряде случаев весьма удобным приемом уточнения корня уравнения является метод последовательных приближений (метод итераций).

Пусть с точностью ε необходимо найти корень уравнения $f(x)=0$, принадлежащий интервалу $[a,b]$. Функция $f(x)$ и ее первая производная непрерывны на этом отрезке.

Для применения этого метода исходное уравнение $f(x)=0$ должно быть приведено к виду $x=\varphi(x)$.

В качестве начального приближения может быть выбрана любая точка интервала $[a,b]$.

Для того, чтобы последовательность $x_1, x_2, \ldots, x_n$ приближалась к искомому корню, необходимо, чтобы выполнялось условие сходимости $|\varphi'(x)|<1$.

Пример реализации метода итераций представлен ниже:

```
(%i24) f:exp(-x)-x$
       beta:0.1$ x1:1$ x0:0$ eps:0.000001$ p:0$
       while abs(x1-x0)>eps do
                  (x0:x1, p:p+1, x1:float(x0+beta*(subst(x0,x,f))))$
       print("Число итераций ",p," ","Решение ",float(x1),
       "    Невязка ",float(abs(x1-x0)))$

Число итераций 67   Решение 0.5671484832781433   Невязка 9.650298036234517 10^-7
```

3. Метод Ньютона (метод касательных)

Рассмотренные ранее методы решения нелинейных уравнений являются методами прямого поиска. В них для нахождения корня используется нахождение значения функции в различных точках интервала $[a,b]$.

Рассмотрим нелинейное уравнение $f(x)=0$, Для которого необходимо найти корень на интервале $[a,b]$ с точностью ε.

Метод Ньютона основан на замене исходной функции $f(x)$, на каждом шаге поиска касательной, проведенной к этой функции. Пересечение касательной с осью X дает приближение корня.

Выберем начальную точку $x_0=b$ (конец интервала). Находим значение функции в этой точке и проводим к ней касательную, пересечение которой с осью X дает первое приближение корня x_1:

$x_1=x_0-h_0$, где

$$h_0=\frac{f(x_0)}{tg(\alpha)}=\frac{f(x_0)}{f'(x_0)}.$$

Поэтому $x_1=x_0-\dfrac{f(x_0)}{f'(x_0)}$. В результате, итерационный процесс схождения к корню реализуется рекуррентной формулой $x_{n+1}=x_x-\dfrac{f(x_n)}{f'(x_n)}$.

Процесс поиска продолжается до тех пор, пока не выполнится условие: $|x_{n+1}-x_n|\le\varepsilon$, откуда $\left|\dfrac{f(x_n)}{f'(x_n)}\right|\le\varepsilon$.

Пример реализации метода Ньютона в **Maxima** представлен ниже:

```
(%i1)  newton(f,x0,eps):=block[[df,xn,xn0,r,p],
       xn0:x0, df:diff(f,x),
       p:0, r:1,
       while abs(r)>eps do (
         p:p+1, xn:xn0-float(subst(xn0,x,f)/subst(xn0,x,df)),
         print("x0,x1 ",xn0,xn),r:xn-xn0, xn0:xn
       ),
       [xn,p]]$
```

Последовательность команд для обращения к функции *newton* и результаты вычислений представлены в следующем примере

```
(%i5)  f:exp(-x)-x$
       eps:0.000001$ xrez:newton(f,1,eps)$
       print("Решение   ",xrez[1],"Число итераций ",xrez[2],
         "Невязка ",subst(xrez[1],x,f))$
       x0,x1  1 0.5378828427399902
       x0,x1  0.5378828427399902 0.5669869914054133
       x0,x1  0.5669869914054133 0.567143285989123
       x0,x1  0.567143285989123 0.5671432904097838
       Решение   0.5671432904097838 Число итераций  4 Невязка  0.0
```

Решение дифференциальных уравнений

Помимо «просто» уравнений, Maxima позволяет также решать и обыкновенные дифференциальные уравнения первого и второго порядка. Функций, непосредственно занимающихся решением таких уравнений, существует две. Первая из них занимается поиском частных решений линейных дифференциальных уравнений и систем таких уравнений; это `desolve`, от слов *differential equation solve*. Функция принимает два аргумента, первый из которых — уравнение либо список уравнений, а второй — соответственно одна переменная или список переменных. Если не заданы значения функций и/или их производных в нуле, то в найденном решении они просто отображаются в виде `f(0)`.

Задать эти значения позволяет функция `atvalue(выражение, переменная = точка, значение);` *то есть, в данном случае* `atvalue(f(x), x=0, значение)` *или* `atvalue('diff(f(x)), x=0, значение)`. *Производные в уравнениях и системах, решаемых с помощью этой функции, должны быть записаны непременно в виде* `'diff(f(x), x)`, *а не просто* `'diff(f, x)`, *а сами функции, соответственно, тоже в виде* `f(x)`, *а не* `f` *— нужно продемонстрировать зависимость функции от ее аргумента.*

```
(%i1)  ['diff(f(x),x) = 'diff(g(x),x) + sin(x),
        'diff(g(x),x,2) = 'diff(f(x),x) - cos(x)]
```

$$\left[\frac{d}{dx}f(x)=\frac{d}{dx}g(x)+\sin x,\frac{d^2}{dx^2}g(x)=\frac{d}{dx}f(x)-\cos x\right]$$

```
(%i2)  atvalue('diff(g(x),x),x=0,a)$ atvalue(f(x),x=0,1)$
(%i4)  desolve(%o1,[f(x),g(x)])
```

$$[f(x)=a\,e^x-a+1,g(x)=\cos x+a\,e^x-a+g(0)-1]$$

И конечно же, точно так же как для обычных уравнений и систем, здесь мы тоже можем проверить решение с помощью подстановки, но только надо еще дополнительно задать принудительное вычисление производных, так как в уравнениях они фигурируют в несовершенной форме:

```
(%i5)  %o1,%,diff
```

$$[a\,e^x=a\,e^x,a\,e^x-\cos x=a\,e^x-\cos x]$$

Вторая функция из этой группы называется `ode2` и предназначена она для решения обыкновенных дифференциальных уравнений первого и второго порядка; ее название происходит от фразы *ordinary differential equations of 1st or 2nd order*. Пишется она так: `ode2(уравнение, зависимая-переменная, независимая-переменная).` Здесь уже независимая переменная указывается в списке параметров функции явно, и потому обозначения вида `y(x)` не нужны: и функция, и переменная обозначаются просто одиночными буквами. Также в отличие от предыдущей функции, `ode2` ищет не частное, а общее решение. Произвольная константа в решении уравнения первого порядка обозначена через `%c`; в решении уравнения второго порядка таких констант, естественно, две, и обозначаются они как `%k1` и `%k2`.

(%i1) $x^2\,'\mathrm{diff}(y,x)+3\,y\,x=\dfrac{\sin(x)}{x}$

(%o1) $x^2\left(\dfrac{d}{d\,x}\,y\right)+3\,x\,y=\dfrac{\sin x}{x}$

(%i2) $\mathrm{ode2}(\%,y,x)$

(%o2) $y=\dfrac{\%\mathrm{c}-\cos x}{x^3}$

(%i3) $'\mathrm{diff}(y,x,2)+y\,'\mathrm{diff}(y,x)^3=0$

(%o3) $\dfrac{d^2}{d\,x^2}\,y+y\left(\dfrac{d}{d\,x}\,y\right)^3=0$

(%i4) $\mathrm{ode2}(\%,y,x)$

(%o4) $\dfrac{y^3+6\,\%\mathrm{k1}\,y}{6}=x+\%\mathrm{k2}$

В дополнение к функции `ode2` существуют три функции для поиска частных решений на основе полученных общих. Иначе говоря, эти функции, получая конкретные условия относительно значения функции-решения в заданной точке, находят исходя из этих значений соответствующие им величины интегральных констант. Одна из этих функций предназначена для обработки решения дифференциального уравнения первого порядка. Она называется `ic1` (*i* от *initial value* — *начальное значение*; *c* от *constant* — *константа*; *1* от *1st order* — *первого порядка*) и принимает три аргумента: первый — само решение, в том виде, в котором его находит функция `ode2`; второй — значение независимой переменной (*x*-координаты), третий — значение функции (зависимой переменной, *y*) при этом значении *x* и возвращает частное решение, проходящее через точку с заданными координатами (*x*, *y*):

(%i5) $\mathrm{ic1}(\%o2,x=\pi,y=0)$

(%o5) $y=-\dfrac{\cos x+1}{x^3}$

Две функции работают с решениями уравнений второго порядка. Так как в общем решении уравнения второго порядка фигурируют две независимые константы, то эти функции задают уже по два условия для поиска частного решения. Первая функция выглядит как `ic2 (общее решение, x, функция в точке x, производная в точке x).` Расшифровка названия аналогична предыдущей функции. Действует тоже аналогично ей, а в качестве второго условия задает значение производной в той же заданной точке:

(%i8) bc2(%o4, $x=0, y=1, x=1, y=3$)

(%o8) $\frac{y^3-10y}{6}=x-\frac{3}{2}$

Нахождение общих решений дифференциальных уравнений первого и второго порядка с помощью меню Maxima выполняется посредством задания последовательности кнопок панели Maxima: Уравнения→Solve ODE, частные решения находим далее Initial Value Problem (1), Initial Value Problem (2) Boundary Value Problem.

Задания для самостоятельной работы

Вариант 1

1. Решить дифференциальные уравнения:

A) $xy' - y = y^3$; B) $y' + \frac{y}{x} = -xy^2$ C) $y'' - y' - 2y = 0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $\left(1+e^x\right)y \cdot y' = e^x$; $y=1$ при $x=0$;

B) $y'' + 4y' = 12x^2 - 2x + 2$; $y=0$, $y'=0$

при $x=0$, построить графики решений.

3. Найти значение всех корней уравнения:

A) $x^5 - 1 = 0$;

B) найти приближенно значение $\cos x$, в окрестности точки $x=1$, с точностью до 0.001.

4. Разложить в ряд Тейлора следующие функции

1. $\sin(x+\frac{\pi}{4})$. 2. e^{x^2}.

Вариант 2

1. Решить дифференциальные уравнения:

A) $y' \operatorname{tg} x = y$; B) $x\left(x+2y\right)dx + \left(x^2 - y^2\right)dy = 0$ C) $y'' - 9y = 2 - x$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $xy' + y - e^x = 0$; $y=b$ при $x=a$;

B) $y'' + 4y = 2\cos 2x$; $y=0$, $y'=4$

при $x=0$, построить графики решений.

3. Найти приближеннное значение корней уравнения:

A)(-4.5 + 6x)^3 = 0.7(9 + x^5);

B) найти приближенно значение $\sin x = 0$, в окрестности точки $x=1$.

4. Разложить в ряд Тейлора следующие функции

1. $\sin^2 x$ 2. $\ln(2+x)$

Вариант 3

1. Решить дифференциальные уравнения:

A) $(1+y)dx-(1-x)dy=0$; B) $y'=\frac{y}{x}-1$ C) $y''+y'=\frac{1}{2}$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $\left(xy^2+x\right)dx+\left(x^2y-y\right)dy=0$; $y=1$ при $x=0$;

B) $y''-9y'=2-x;\quad y=0,\ y'=1$

при $x=0$, построить графики решений.

3. Найти приближеннное значение корней уравнения:

A) (5.6 - 2x)^3 = -7.7(-1 + x^5)

B) $2\sin x-x=0$.

4. Разложить в ряд Тейлора следующие функции

1. $\cos x^2$. 2. $\ln(1+x-2x^2)$.

Вариант 4

1. Решить дифференциальные уравнения:

A) $\left(1+y^2\right)dx+\left(1+x^2\right)dy=0$; B) $y'=-\frac{x+y}{x}$ C) $y''+3y=0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $x(x-1)y'+y=x^2(2x-1);y=4$ при $x=2$;

B) $y''-2y'+3y=0;\quad y=1,\ y'=3$

при $x=0$, построить графики решений.

3. Найти приближеннное значение действительных корней уравнения:

$(1-x)^5(2-x)^3(3-x)=0$.

4. Разложить в ряд Тейлора следующие функции

1. $\cos(x+y)$. 2. $\frac{x}{4+x^2}$.

Вариант 5

1. Решить дифференциальные уравнения:

A) $\left(1+e^x\right)yy'=e^x$; B) $(x-y)ydx-x^2dy=0$ C) $y''-4y'+10y=0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $y'+y\cos x=\cos x;\ y=1$ при $x=0$;

B) $y''-2y'+2y=0;\quad y=1,\ y'=3$

при $x=0$, построить графики решений.

3. Найти приближеннное значение корней уравнения:

A) (11.4 + x)^3 = 1.9(5 + x^5)$

B) $\sin x-x/2=0$.

4. Разложить в ряд Тейлора следующие функции

1. $\frac{1}{\sqrt{4-x^2}}$. 2. $\ln\frac{1+x}{1-x}$.

Вариант 6

1. Решить дифференциальные уравнения:

А) $2xyy' = x^2 + y^2$; В) $(x+y)dx + xdy = 0$; С) $y'' + 2y' - 2y = 0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

А) $xy' - y + x^2 = 0;\ \ y = 0$ при $x = 1;$

В) $y'' - 4y' + 3y = 0;\ \ y = 6,\ y' = 10$

при $x = 0$, построить графики решений.

3. Найти приближеннное значение действительных корней уравнения:

$x^5 - x - 1.$

4. Разложить в ряд Тейлора следующие функции

1. $\cos 2x.$ 2. $\dfrac{x}{9 + x^2}.$

Вариант 7

1. Решить дифференциальные уравнения:

А) $x(x+2y)dx + (x^2 - y^2)dy = 0$; В) $y' = \dfrac{x-y}{x-2y}$ С) $y'' + 16y = 0.$

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

А) $y' - y\,\mathrm{tg}\,x = \dfrac{1}{\cos x};\ \ y = 0$ при $x = 0;$

В) $y'' + 4y' = 12x^2 - 2x + 2;\ \ y = 0,\ y' = 0$

при $x = 0$, построить графики решений.

3. Найти приближеннное значение корней уравнения:

А) (18 + 2x)^3 = 5.3(1 + x^5);

В) $\sin x - x = 0$ на отрезке $[0, \pi]$

4. Разложить в ряд Тейлора следующие функции

1. $xe^{-6x}.$ 2. $e^{(x+y)}.$

Вариант 8

1. Решить дифференциальные уравнения:

А) $y' + 2y = e^{-x}$; В) $y' - \dfrac{y}{x} = x$ С) $y'' - 9y' - 10y = 0.$

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

А) $y' - \dfrac{y}{1-x^2} - 1 - x = 0;\ \ y = 0$ при $x = 0;$

В) $y'' + 4y = 2\cos 2x;\ \ y = 0,\ y' = 4$

при $x = 0$, построить графики решений.

3. Найти приближеннное значение корней системы уравнений:

$24x^2 - y^2 = 10,\quad xy - x = 2$

4. Разложить в ряд Тейлора следующие функции

1. xe^{-2x}. 2. e^{-x}.

Вариант 9

1. Решить дифференциальные уравнения:

A) $xy' - 2y = x^3 \cos x$; B) $y' + \frac{2y}{x} = x^3$; C) $y'' + 4y' - 7y = 0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $\left(x^2 - 3y^2\right)dx + 2xydy = 0$; $y = 1$ при $x = 2$;

B) $y'' - 9y' = 2 - x$; $y = 0$, $y' = 1$

при $x = 0$, построить графики решений.

3. Найти приближеннное значение корней уравнения:

A) $x^6 + 1 = 0$;

B) $\cos x - x = 0$ на отрезке $[-\pi, \pi]$

4. Разложить в ряд Тейлора следующие функции

1. $\frac{2x-3}{(x-1)^2}$. 2. $\frac{3x-5}{x^2 - 4x + 3}$.

Вариант 10.

1. Решить дифференциальные уравнения:

A) $y' x \ln x - y = 3x^3 \cos x$; B) $y' + \frac{y}{x} = -xy^2$ C) $y'' + 20y' + 19y = 0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $y' \sin x = y \ln y$; $y = 1$ при $x = \frac{\pi}{2}$;

B) $y'' - 2y' + 3y = 0$; $y = 1$, $y' = 3$

при $x = 0$, построить графики решений.

3. Найти значение всех корней уравнения:

A) $x^3 - 1 = 0$;

B) $\cos x - x/2 = 0$ на отрезке $[-\pi, \pi]$

4. Разложить в ряд Тейлора следующие функции

1. $\cos^2 x$. 2. $\sin(x + y)$.

Вариант 11

1. Решить дифференциальные уравнения:

A) $\left(1+e^x\right)yy^{/}=e^x$; B) $\left(x-y\right)ydx-x^2dy=0$ C) $y^{//}-6y^{/}+9y=0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям, построить графики решений:

A) $y^{/}+y\cos 2x=\cos 2x;\ y=2$ при $x=0$;

B) $y^{//}-2y^{/}+2y=0;\quad y=1,\ y^{/}=3$

при $x=0$.

3. Найти приближенное значение корней уравнения:

A) (11.4 + x)^4 = 1.9(5 - x^2)$

B) $\sin 2x-x/3=0$.

4. Разложить в ряд Тейлора следующие функции

1. $\frac{1}{\sqrt{9-x^2}}$. 2. $\ln\frac{1}{1-x}$.

Вариант 12.

1. Решить дифференциальные уравнения:

A) $xyy^{/}=x^2+y^2$; B) $\left(x-y\right)dx+xdy=0$; C) $y^{//}+y^{/}-y=0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $xy^{/}-y+x^2=0;\quad y=0$ при $x=1{,}5$;

B) $y^{//}-4y^{/}+3y=0;\quad y=3,\ y^{/}=9$

при $x=0$, построить графики решений.

3. Найти приближенное значение действительных корней уравнения:

x^5-x+7.

4. Разложить в ряд Тейлора следующие функции

1. $\cos 10x$. 2. $\frac{x}{16+x^2}$.

Вариант 13.

1. Решить дифференциальные уравнения:

A) $x\left(x+2y\right)dx+\left(x^2-y^2\right)dy=0$; B) $y^{/}=\frac{x-y}{x-2y}$ C) $y^{//}-16y=0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $y^{/}-y\,\mathrm{tg}\,x=\frac{1}{\cos x};\ y=1$ при $x=0$;

B) $y^{//}+4y^{/}=12x^2-2x+2;\ y=0,\ y^{/}=1$

при $x=0$, построить графики решений.

3. Найти приближеннное значение корней уравнения:

A) (10 - 2x)^3 = 5.3(1 - x^5);

B) $\sin 2x - 2x = 0$ на отрезке [$-0, \pi$]

4. Разложить в ряд Тейлора следующие функции

1. xe^{-12x}. 2. $e^{(x-y)}$.

Вариант 14

1. Решить дифференциальные уравнения:

A) $y' + y = e^{-x}$; B) $y' + \frac{y}{x} = x$ C) $y'' - 9y' + 10y = 0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $y' - \frac{y}{1-x^2} - 1 - x = 0$; $y = 0$ при $x = 1$;

B) $y'' + 4y = 2\cos 2x$; $y = 0$, $y' = 4.5$

при $x = 0$, построить графики решений.

3. Найти приближенное значение корней системы уравнений:

$24x^2 - y^2 = 10, \quad xy - x = 4$

4. Разложить в ряд Тейлора следующие функции

1. xe^{-14x}. 2. $e^{-\cos x}$.

Вариант 15

1. Решить дифференциальные уравнения:

A) $xy' - 2y = x^2 \cos x$; B) $y' + \frac{2y}{x} = x^3$; C) $y'' + 4y' - 7y = 0$.

2. Найти частные решения уравнений, удовлетворяющие указанным начальным условиям:

A) $\left(x^2 - 3y^2\right)dx + 2xydy = 0$; $y = 0$ при $x = 2$;

B) $y'' - 9y' = 2 - x$; $y = 0$, $y' = 1$

при $x = 0$, построить графики решений.

3. Найти приближеннное значение корней уравнения:

A) $x^{12} + 1 = 0$;

B) $\cos 3x - x = 0$ на отрезке [$-\pi, \pi$]

4. Разложить в ряд Тейлора следующие функции

1. $\frac{3x-5}{(x-1)^2}$. 2. $\frac{3x-5}{x^2 - 4x + 3}$.

Численные методы решения дифференциальных уравнений

1. Решение задачи Коши для обыкновенного дифференциального уравнения первого порядка

Рассмотрим постановку задачи Коши для системы обыкновенных дифференциальных уравнений (ОДУ) вида

$$\frac{dy}{dx} = f(x, y)\ , \tag{1.1}$$

где: y-искомая вектор-функция; x – независимая переменная;
$y(x) = \big(y_1(x), \dots, y_m(x)\big)$; $f(x) = f_1, \dots, f_m$, m – порядок системы;
$y_1(x), \dots, y_m(x)$ – координаты; $x \geq 0$; $y(0) = y^0$.

Систему (1.1) можно переписать в развернутом виде

$$\frac{dy_i}{dx} = f_i(x, y_1, \dots, y_m), \tag{1.2}$$

где: i=1,…, m; $y_i(0) = y_i^0$.

Если i=1, то мы получаем обыкновенное дифференциальное уравнение первого порядка:

$$\frac{dy}{dx} = f(x, y)\ , \tag{1.3}$$

При этом решение задачи Коши для уравнения (1.3) заключается в нахождении интегральной кривой, проходящей через заданную точку и удовлетворяющую заданному начальному условию $y(a) = y_a$. Задача состоит в том, чтобы найти искомую функцию y, удовлетворяющую (21.3) и заданным начальным условиям.

Построение численных алгоритмов решения уравнения (1.1) опирается на дискретизацию задачи. Введем в области расчета $x \in [a, b]$ дискретный набор точек $x_i = a + hi, i = 0,1, \dots, N, h = (b - a)/N$, в которых будем вычислять приближенное решение. Точки x_i называются узлами интегрирования или шагом сетки. Сеточной областью (сеткой) называется совокупность всех узлов.

Для характеристики точности численного метода определяется погрешность приближенного решения по формуле:

$$\partial = \max_i |y_i - y(x_i)|, \tag{1.4}$$

где: $y(x_i)$ - значение точного решения в узле сетки.

Существует два класса методов для решения задачи (1.1).

1) семейство одношаговых методов (Рунге-Кутта);
2) семейство многошаговых (m-шаговых) методов.

Численный метод называется *явным*, если вычисление решения в следующей точке y_{i+1} осуществляется по явной формуле. Метод называется *одношаговым*, если вычисление решения в следующей точке y_{i+1} производится с использованием только одного предыдущего значения y_i.

В дальнейшем будем рассматривать численные методы решения задачи Коши на примере уравнения первого порядка:

$$\begin{cases} \frac{dy}{dx} = a(x, y), a \leq x \leq b \\ y(a) = y_a \end{cases} \tag{1.5}$$

Метод Эйлера

Простейшим численным методом решения задачи Коши (1.5) для обыкновенного дифференциального уравнения является метод Эйлера, который еще называют методом ломаных Эйлера.По оси x введем равномерную сетку с шагом h>0 , т.е. рассмотрим систему точек $x_i = \{x_i = 0x, i = 0,1,2,...\}$ Обозначим через y(x)точное решение задачи (2.5) , а через $y_i = y(x_i)$ - приближенные значения функций y в заданной системе точек.

Заменяя в уравнении (1.5) производную в окрестности каждого i -го узла сетки разностным отношением, приходим к уравнению:

$$\frac{y_{i+1}-y_i}{h}=f(x_i,y_i),\quad i=0,1,2,...,N-1,\quad y_0=y_0 \qquad (1.6)$$

Алгебраические соотношения между компонентами сеточной функции, которыми заменяются исходные дифференциальные уравнения в окрестности каждого узла сетки, называются разностными уравнениями. Поэтому уравнение (1.6) — разностное уравнение. В окончательной форме значения y_{i+1} можно определить по явной формуле

$$y_{i+1}=y+hf(x_i,y_i) \qquad (1.7)$$

Геометрическая интерпретация метода Эйлера: интегральная кривая y(x) на отрезке [a ;b] приближается к ломаной, наклон которой определяется наклоном интегральной кривой уравнения в точке $[x_i;y_i]$.

Метод Эйлера относится к явным одношаговым методам. Вследствие систематического накопления ошибок метод используется редко или используется только для оценки вида интегральной кривой. Метод Эйлера называют методом Рунге-Кутта первого порядка точности.

ПРИМЕР 1. Решение задачи Коши методом Эйлера. Применяя метод Эйлера, найти решение задачи Коши: $\begin{cases} y'=y-x \\ y(0)=1.5 \end{cases}$ в трех последовательных точках $x_1=0.2$, $x_2=0.2$, $x_3=0.6$. Найти точное решение задачи и найти величину абсолютной погрешности в указанных точках.

Решение. Возьмем шаг h = 0.2 . Используя расчетную формулу Эйлера, найдем приближенное решение задачи Коши:

$y_1=y_0+0.2(y_0-x_0)=1.5+0.2+1.5=1.8$

$y_2=y_1+0.2(y_1-x_1)=1.8+0.2(1.8-0.2)=2.12$

$y_3=y_2+0.2(y_2-x_2)=2.12+0.2(2.12-0.4)=2.464$

Таким образом, получили численное решение задачи:

x_i	0	0.2	0.4	0.6
y_i	1.5	1.8	2.12	2.464

Графиком приближенного решения является ломаная, последователь- но соединяющая точки (x_i,y_i). В этой задаче легко находится точное решение, например, методом вариации постоянной: $y(t)=0,5e^t+t+1$. Вычислим значения точного решения в указанных точках.

t_i	0	0.2	0.4	0.6
$y(t_i)$	1.5	1.811	2.146	2.511

Абсолютную погрешность вычислим так: $r_i=|y(t_i)-y_i|$.

Тогда $r_1=0.011, r_2=0.026, r_3=0.047$. Таким образом, максимальная величина погрешности равна R = 0.05.

Рассмотрим решение этой задачи в Maxima. Зададим концы отрезка, на котором будем искать решение, и шаг:

```
(%i1) a:0$b:0.6$ h:0.2$
```

Найдем количество точек разбиения отрезка с шагом h :

```
(%i4) n:1+floor((b-a)/h)$
```

Сформируем два пустых одномерных массива размера n□1 для хранения значения координат точек [x , y] искомого решения:

```
(%i5) x1:make_array(flonum, n+1)$
      y1:make_array(flonum, n+1)$
```

Зададим начальное условие:

```
(%i7) x1[0]:a$ x1[1]:a+h$y1[0]:1.5$
```

Заполним массив x 1 значениями, начиная с 0.2 до 1 с шагом h . Для этого используем цикл с параметром.

```
(%i10) for i:2 thru n step 1 do (x1[i]:x1[i -1]+h)$
```

Используя расчетную формулу Эйлера () главы 2, заполним массив y1 :

```
(%i11) for i:1 thru n step 1 do
       (y1[i]:float(y1[i-1]+h*(y1[i-1]-x1[i-1])))$
```

Выведем полученное решение на экран:

```
(%i12) for i:0 thru n step 1 do
       (display(x1[i]), display(y1[i]));
```

$x1_0=0$

$y1_0=1.5$

$x1_1=0.2$

$y1_1=1.8$

$x1_2=0.4$

$y1_2=2.12$

$x1_3=0.6$

$y1_3=2.464$

$x1_4=0.8$

$y1_4=2.836800000000001$

$x1_5=1.0$

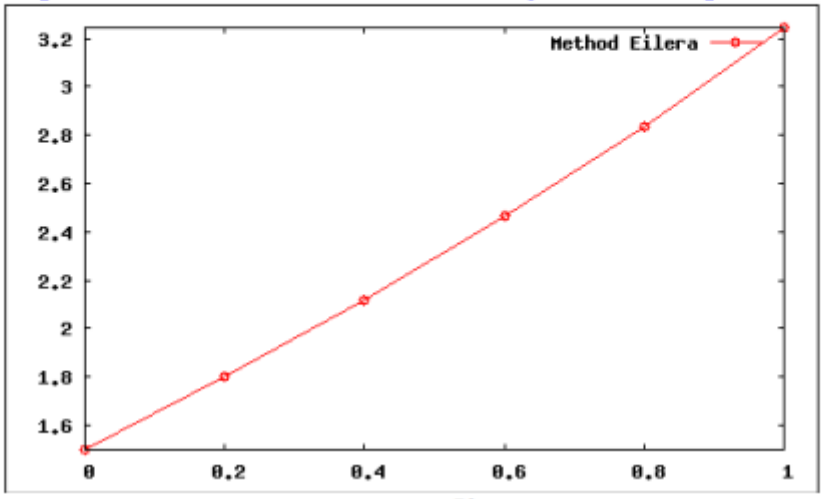
$y1_5=3.244160000000001$

Выполним построение ломаной Эйлера (найденного решения задачи) средствами пакета г:

```
(%i13) load(draw)$

(%i14) wxdraw2d(point_type=circle,point_size=1,
       points_joined=true,
       key="Method Eilera", color=red, points(x1, y1))$
```

(%t14)



Для нахождения точного решения задачи Коши воспользуемся встроенной командой desolve. Дифференциальное уравнение запомним под именем eq1:

```
(%i15) eq1:'diff(y(t),t)=y(t)-t;
```

$$(\%o15)\quad \frac{d}{dt}y(t)=y(t)-t$$

Задаем начальное условие:

```
(%i16) atvalue(y(t), t=0, 1.5);
(%o16) 1.5
```

Находим точное решение задачи Коши :

```
(%i17) desolve(eq1, y(t));
rat: replaced -1.5 by -3/2 = -1.5
```

$$(\%o17)\quad y(t)=\frac{\%e^{t}}{2}+t+1$$

Вычислим значения функции в точках отрезка [0,1] с шагом h=0.2 .

```
(%i19) for i:0 thru n do (z1[i]:%e^x1[i]/2+x1[i]+1,
       display(z1[i]));
```

$$z1_0=\frac{3}{2}$$

$$z1_1=1.810701379080085$$

$$z1_2=2.145912348820635$$

$$z1_3=2.511059400195255$$

$$z1_4=2.912770464246234$$

$$z1_5=3.359140914229522$$

```
(%o19) done
```

Найдем величину абсолютной погрешности:

```
(%i20) for i:1 thru n do
       (display(abs(z1[i]-y1[i])));
|0.010701379080085|=0.010701379080085
|0.025912348820635|=0.025912348820635
|0.047059400195254|=0.047059400195254
|0.075970464246233|=0.075970464246233
|0.11498091422952|=0.11498091422952
(%o20) done
```

Метод Эйлера-Коши

Отличительная особенность метода Эйлера-Коши от метода Эйлера заключается в том, что значение правой части уравнения вычисляется не только в точках сетки (шаг h), но и также в середине отрезков (шаг $\frac{h}{2}$) (в промежуточных точках).

Предположим, что приближенное значение y_i решения задачи в точке $x=x_i$ уже известно, y_{i+1} вычисляются по следующим формулам:

$$y_{i+1}=y_i++\Delta_y,\ \Delta y_i=\Delta y_{i1}+\Delta y_{i2},$$

$$\Delta y_{i1}=\frac{h}{2}f\left(x_i,y_i\right),\ \Delta y_{i2}=\frac{h}{2}f\left(x_i+h,y_i+hf\left(x_i,y_i\right)\right)$$

Отсюда

$$y_{i+1} = y_i + h\frac{f(x_i, y_i) + f(x_{i+1}, y_i + hf(x_i, y_i))}{2}. \quad (1.8)$$

Геометрическая интерпретация метода Эйлера-Коши: определяется направление интегральной кривой в исходной точке (x_i, y_i) и во вспомогательной точке $\left(x_{i+1}, y^0_{i+1}\right)$, $y^0_{i+1} = y_{i+1} + hf(x_i, y_i)$, а в качестве окончательного выбирается среднее из этих направлений. Метод Эйлера-Коши называют методом Рунге-Кутта второго порядка точности.

ПРИМЕР 2. Применяя метод Эйлера-Коши, найти решение задачи Коши из предыдущего примера.

Решение. Возьмем шаг $h = 0.2$. Используя расчетную формулу Эйлера-Коши (1.8), найдем приближенное решение задачи Коши:

$$y_1^0 = y_0 + hf(x_0, y_0) = 1.5 + 0.2(1.5 - 0) = 1.8$$

$$y_1 = y_0 + h\frac{y_0 - x_0 + f\left(x_1, y_1^0\right)}{2} = 1.5 + 0.2\frac{1.5 - 1.8 - 0,2}{2}$$

$$y_1 = 1.5 + 0.2\frac{1.5 + 1.6}{2} = 1.5 + 0.31 = 1.81$$

$$y_2^0 = y_1 + h(y_1 - x_1) = 1.8 + 0.2(1.81 - 0.2) = 2.132$$

$$y_2 = y_1 + h\frac{y_1 - x_1 + f\left(x_2, y_2^0\right)}{2}$$

$$y_2 = 1.81 + 0.2\frac{(1.81 + 0.2 + 2.132 - 0.4)}{2} = 1.81 + 0.3342 = 2.1442$$

$$y_3^0 = y_2 + h(y_2 - x_2) = 2.1442 + 0.2(2.1442 + 0.4) = 2.49304$$

$$y_3 = y_2 + h\frac{y_2 - x_2 + f\left(x_3, y_3^0\right)}{2}$$

$$y_2 = 2.1442 + 0.2\frac{2.1442 - 0.4 + 249304 - 0.6}{2} = 2.1442 + 0.363724 = 2.507924$$

Таким образом, получили численное решение задачи Коши:

x_i	0	0.2	0.4	0.6
y_i	1.5	1.81	2.1442	2.507924

Графиком приближенного решения является ломаная, последовательно соединяющая точки (*xi* , *yi*) . Как видим, решения задачи Коши, полученные методом Эйлера и методом Эйлера-Коши очень близки.

Реализация Метода Эйлера-Коши в Maxima

Для решения этой задачи методом Эйлера сформируем еще три пустых массива x2, y2 и вспомогательный массив z.

```
(%i3)    a:0$b:0.6$h:0.1$
(%i4)    n:1+floor((b-a)/h)$
(%i7)    x2:make_array(flonum,n+1)$
         z:make_array(flonum,n+1)$
         y2:make_array(flonum,n+1)$
```

Зададим начальное условие:

```
(%i10)   x2[0]:a$x2[1]:a+h$y2[0]:1.5$
```

Заполним массив x 2 значениями, начиная с 0.2 до 1 с шагомh .
Для этого используем цикл с параметром.

```
(%i11)   for i:2 thru n step 1 do (x2[i]:x2[i-1]+h)$
```

Теперь воспользуемся расчетной формулой Эйлера-Коши и найдем решение:

```
(%i12)   for i:1 thru n step 1 do (z[i]:float(y2[i-1]+h*(y2[i-1]-x2[i-1])),
         y2[i]:float(y2[i-1]+h*(y2[i-1]-x2[i-1]+z[i]-x2[i])/2))$
```

Выведем найденное решение на экран:

```
(%i13)   for i:0 thru n step 1 do (display(x2[i]),display(y2[i]))$
```

$x2_0=0$

$y2_0=1.5$

$x2_1=0.1$

$y2_1=1.6525$

$x2_2=0.2$

$y2_2=1.8105125$

$x2_3=0.3$

$y2_3=1.9746163125$

$x2_4=0.4$

$y2_4=2.1454510253125$

$x2_5=0.5$

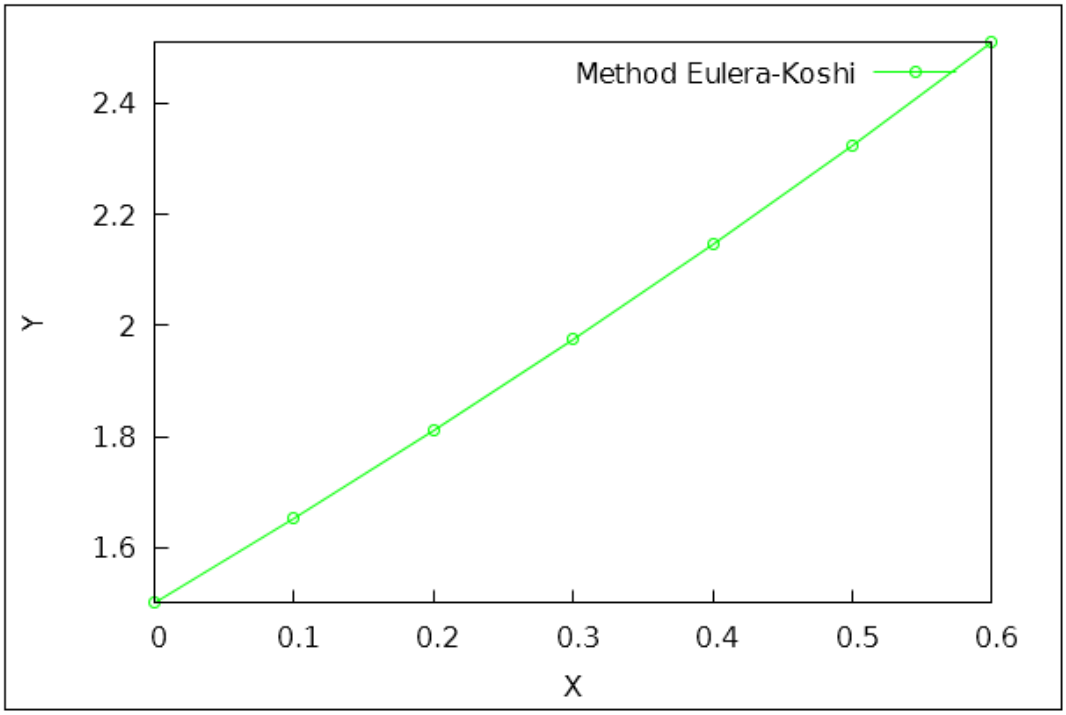
$y2_5=2.323723382970313$

$x2_6=0.6$

$y2_6=2.510214338182196$

Выполним построение найденного решения задачи (4.1) средствами пакета draw:

```
(%i15)  wxdraw2d(point_type=circle,point_size=1,points_joined=true,
        key="Method Eulera-Koshi",color=green,points(x2,y2))$
```



(%t15)

Как видим, метод Эйлера-Коши дает более точный результат, чем метод Эйлера. Максимальная погрешность вычислений составляет 0.7%.

В качестве примера рассмотрим еще решение следующей задачи:

$y' = x^3 - y$, $y(0) = -1$, $[a,b] = [0,1]$

```
(%i3)    a:1.0$b:2.0$h:0.2$
(%i4)    n:1+floor((b-a)/h)$
(%i7)    x2:make_array(flonum,n+1)$
         z:make_array(flonum,n+1)$
         y2:make_array(flonum,n+1)$
(%i10)   x2[0]:a$x2[1]:a+h$y2[0]:-1.0$
(%i11)   for i:2 thru n step 1 do (x2[i]:x2[i-1]+h)$
(%i12)   for i:1 thru n step 1 do (z[i]:float(y2[i-1]+h*(-y2[i-1]+(x2[i-1])^3)),
         y2[i]:float(y2[i-1]+h*(-y2[i-1]-(x2[i-1])^3+z[i]-x2[i])/2))$
(%i13)   for i:0 thru n step 1 do (display(x2[i]),display(y2[i]))$
```

$x2_0 = 1.0$

$y2_0 = -1.0$

$x2_1 = 1.2$

$y2_1 = -1.18$

$x2_2 = 1.4$

$y2_2 = -1.43464$

$x2_3 = 1.6$

$y2_3 = -1.7854672$

$x2_4 = 1.8$

$y2_4 = -2.257437856$

$x2_5 = 2.0$

$y2_5 = -2.87884909888$

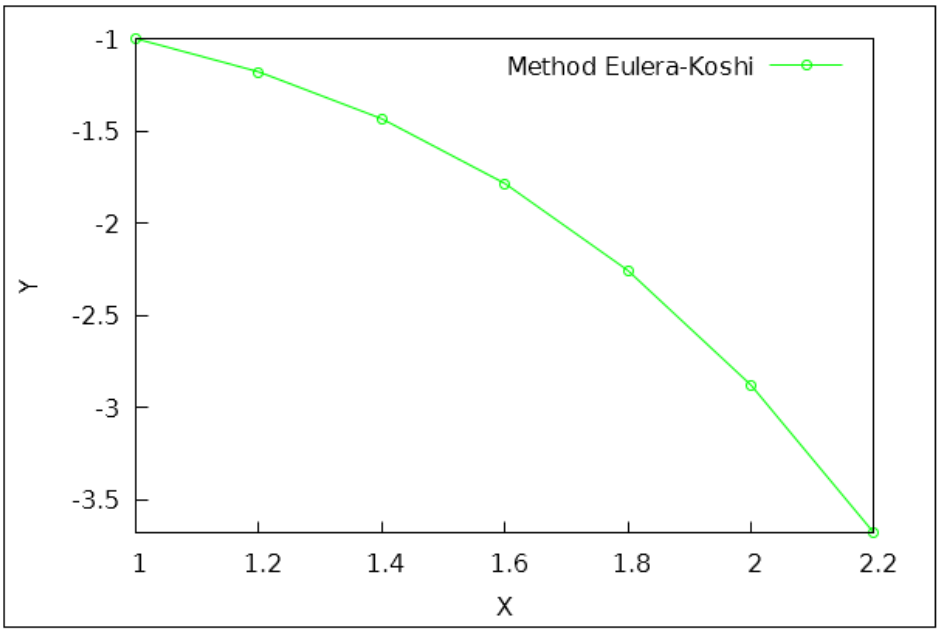
$x2_6 = 2.2$

$y2_6 = -3.6812721169024$

```
(%i14)   wxdraw2d(point_type=circle,point_size=1,points_joined=true,
         key="Method Eulera-Koshi",color=green,points(x2,y2))$
```

0 errors, 0 warnings

(%t14)



Метод Рунге-Кутта

В вычислительной практике наиболее часто используется метод Рунге-Кутта четвертого порядка (классический метод Рунге-Кутта),поскольку позволяет наиболее точно находить решения обыкновенного дифференциального уравнения.

В этом методе значения y_{i+1} находятся по следующим формулам:

$$y_{i+1} = y_i + \Delta y_i; \Delta y_i = h(\mathrm{k}_1 + 2k_2 + 2k3 + k_4)/6, \text{ где } i = 0,1, \ldots$$

$$k_1 = f(\mathrm{x}_i; \mathrm{y}_i); \mathrm{k}_2 = f(\mathrm{x}_i + \frac{h}{2}; \mathrm{y}_i + \frac{h}{2}.\mathrm{k}_1);$$

$$k_3 = f(\mathrm{x}_i + \frac{h}{2}; \mathrm{y}_i + \frac{h}{2}.\mathrm{k}_2); \mathrm{k}_4 = f(\mathrm{x}_i + h; \mathrm{y}_i + h.\mathrm{k}_3); \quad (1.9)$$

ПРИМЕР 3. Решение задачи Коши методом Рунге-Кутта 4 порядка. Применяя метод Рунге-Кутта, найти решение задачи Коши: $\begin{cases} y' = y - x \\ y(0) = 1.5 \end{cases}$ в трех последовательных точках $x_1 = 0.2; x_2 = 0.4; x_3 = 0.6 \cdot$

Решение. Возьмем шаг h=0.2 . Используя расчетные формулыРунге-Кутта (2.9), найдем приближенное решение задачи Коши:

$k_1 = f(x_0, y_0) = y_0 - x_0 = 1.5 - 0 = 1.5$

$k_2 = f(x_0 + \frac{h}{2}, y_0 + \frac{h}{2}k_1 = f(0 + 0.2/2, 1.5 + 0.2/2 \cdot 1.5 = f(0.1, 1.65) = 1.65 - 0.1 = 1.55$

$k_3 = f(0.1, 1.5 + 0.1 \cdot 1.55) = f(0.1, 1.655) = 1.655 - 0.1 = 1.555$

$k_4 = f(0.2, 1.5 + 0.2 \cdot 1.555 = f(0.2, 1.811) = 1.811 - 0.2 = 1.611$

$y_1 = y_0 + h(k_1 + 2k_2 + 2k_3 + k_4)/6 = 1.5 + 0.2(1.5 + 2 \cdot 1.55 + 2 \cdot 1.555 + 1.611 + 1.611)/6 = 1.8107$

$k_1 = f(x_1, y_1) = y_1 - x_1 = 1.8107 - 0.2 = 1.6107$

$k_2 = f(x_1 + \frac{h}{2}, y_1 + \frac{h}{2}k_1) = f(0.2 + 0.1, 1.8107 + 0.1 \cdot 1.6107) = f(0.3, 1.97177) = 1.67177$

$k_3 = f(0.3, 1.8107 + 0.1 \cdot 1.67177) = f(0.2, 1.977877) = 1.977877 - 0.3 = 1.677877$

$k_4 = f(0.4, 1.8107 + 0.2 \cdot 1.677877) = f(0.4, 2.1462754) = 1.7462754$

$y_2 = y_1 + h(k_1 + 2k_2 + 2k_3 + k_4)/6$

$y_2 = 1.8107 + 0.2(1.6107 + 2 \cdot 1.67177 + 2 \cdot 1.677877 + 1.7462754)/6 = 2.14590898$

Аналогично находим

$y_3 = 2.511053228172$

Таким образом, получили численное решение задачи Коши:

x_i	0	0.2	0.4	0.6
y_i	1.5	1.8107	2.14590898	2.511053228172

Графиком приближенного решения является ломаная, последовательно соединяющая точки (x_i , y_i) .

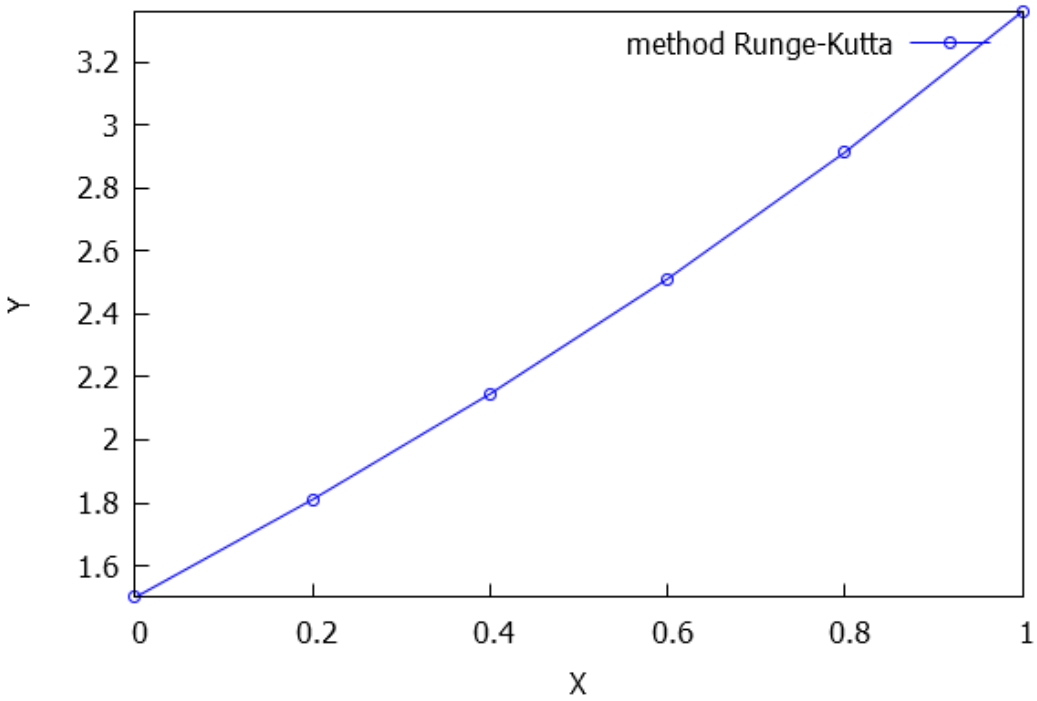
Решение задачи с помощью системы Maxima.

В системе Maxima для нахождения численного решения задачи Коши методом Рунге-Кутта(четвертого пордяка точности) есть встроенная функция rk. Для того, чтобы она стала активной, требуется подключить пакет dynamics с помощью команды:

```
(%i1) load("dynamics")$
(%i2) sol: rk(y-x,y,1.5,[x,0,1,0.2]);
```

(sol)
[[0.0,1.5],[0.2,1.8107],[0.4,2.14590898],[0.6000000000000001,2.511053228172],[0.8,2.91276041288928],[1.0,3.359125568302967]](sol)(sol)[[0.0,1.5],[0.2,1.8107],[0.4,2.14590898],[0.6000000000000001,2.511053228172],[0.8,2.91276041288928],[1.0,3.359125568302967]]

```
(%i3) wxdraw2d (point_type =circle,
      point_size=1,points_joined = true,
      key="method Runge-Kutta",
      color=blue,points(sol))$
```



Задания для самостоятельной работы

Решить задачу Коши для дифференциального уравнения $y' = f(x,y)$ на отрезке $[a,b]$, при заданном начальном условии $y(a) = c$ с шагом *h*: Методом Эйлера, методом Эйлера-Коши, методом Рунге-Кутта 4 порядка, найти величину погрешности в каждом случае, построить графики решений и сравнить приближенные решения с точным решением. Какой из методов дает более точное приближение?

1. $y' = 5x + 2\cos(y + 2.6)$, $y(0) = 1.5$, $[0,1]$;
2. $y' = \frac{xy}{x^2 + y^2}$, $y(0) = 1$, $[0,1]$;

3. $y' = 3e^x + 2y$, $y(0.3) = 1.415$, $[0.3; 0.6]$;

4. $y' = x^2 + y^2$, $y(0) = 0.27$, $[0,1]$;

5. $y' = x^2 - xy + y^2$, $y(0) = 0.1$, $[0,1]$;

6. $y' = x + \sin\frac{y}{3}$, $y(0) = 1$, $[0,2]$;

7. $y' = 5 - 2\sin(y + x)^2$, $y(0) = 1.5$, $[0,1]$;

8. $y' = \frac{2y - x}{y}$, $y(1) = 2$, $[1,2]$;

9. $y' = 2x + \cos y$, $y(0) = 0$, $[0, 0.1]$;

10. $y' = x^3 - y$, $y(1) = -1$, $[1,2]$;

11. $y' = 2x^2 + xy + 3y^2$, $y(0) = 1$, $[0,1]$;

12. $y' = 2xy + x^2$, $y(0) = 0$, $[0, 0.5]$;

13. $y' = 7 + 2\sin(y - x)$, $y(0) = 1$, $[0,1]$;

14. $y' = x - \sin\frac{2y}{3}$, $y(0) = 1$, $[0,2]$;

15. $y' = 2x - y$, $y(0) = -1$, $[0, 0.5]$;

2. Решение краевых задач для линейных дифференциальных уравнений второго порядка методом конечных разностей

Линейное дифференциальное уравнение второго порядка имеет вид:

$$y'' + p(x)y' + q(x)y = f(x) \qquad (2.1)$$

где p(x), q(x)и f(x)—некоторые непрерывные на $[a,b]$функции. Краевая задача для линейного дифференциального уравнения состоит в нахождении его решения $y = y(x)$, удовлетворяющего двухточечным линейным краевым условиям

$$\begin{cases} \alpha_1 y(a) + \alpha_2 y'(a) = A \\ \beta_1 y(b) + \beta_2 y'(b) = B \end{cases} \qquad (2.2)$$

Где

α_1, α_2, β_1, β_2, A, B -постоянные и

$|\alpha_1| + |\alpha_2| \neq 0$, $|\beta_1| + |\beta_2| \neq 0$.

При решении этой задачи методом конечных разностей отрезок $[a,b]$разбивают на равные части с шагом h, где $h = \frac{a-b}{n}$. Точки разбиения имеют абсциссы

$$x_k = x_1 + (k-1)h, \quad k = 1,2,\ldots,n+1, \quad x_1 = a \quad x_{n+1} = b$$

Значения в точках деления x_k искомой функции и её производных $y' = y'(x), \quad y'' = y''(x)$ обозначим соответственно через $y_k = y(x_k), \quad y'_k = y'(x_k), \quad y''_k = y''(x_k)$. Заменяя производные конечно-разностными отношениями для внутренних точек x_k отрезка[a , b], приближенно будем иметь

$$y'_k = \frac{y_{k+1} - y_k}{h}$$
$$y''_k = \frac{y_{k+2} - 2y_{k+1} + y_k}{h^2} \qquad (2.3)$$

Для конечных точек отрезка $x_1 = a \quad x_{n+1} = b$ полагаем

$$y'_1 = \frac{y_2 - y_1}{h} \quad \text{и} \quad y'_{n+1} = \frac{y_{n+1} - y_n}{h} \qquad (2.4)$$

Используя формулы (2.3), дифференциальное уравнение (2.1) при

$$x = x_k (\ k=2,3\ ,\ \ldots\ ,\ n)$$

приближенно можно заменить системой линейных уравнений

$$\frac{y_{k+2} - 2y_{k+1} + y_k}{h^2} + p(x_k)\frac{y_{k+1} - y_k}{h} + q(x_k)y_k = f(x_k); \quad k = 1,2,...,n-1$$

В силу формул (2.4) краевые условия (2.2) дополнительно дают ещё два уравнения

$$\alpha_1 y_1 + \alpha_2 \frac{y_2 - y_1}{h} = A$$
$$\beta_1 y_{n+1} + \beta_2 \frac{y_{n+1} - y_n}{h} = B$$

Таким образом получаем систему $n+1$ линейных уравнений с $n+1$ неизвестными $y_1, y_2, \ldots, y_n, y_{n+1}$, представляющими собой значения искомой функции $y=y(x)$,

$$\begin{cases} \dfrac{y_{k+2} - 2y_{k+1} + y_k}{h^2} + p(x_k)\dfrac{y_{k+1} - y_k}{h} + q(x_k)y_k = f(x_k) \\ \alpha_1 y_1 + \alpha_2 \dfrac{y_2 - y_1}{h} = A \\ \beta_1 y_{n+1} + \beta_2 \dfrac{y_{n+1} - y_n}{h} = B \end{cases}$$

Обозначим

$p(x_k)=p_k$, $q(x_k)=q_k$, $f(x_k)=f_k$.

Выполняя алгебраические преобразования над уравнениями, можно привести систему к следующему виду:

$$\begin{cases} (h^2 q_k - hp_k + 1)y_k + (hp_k - 2)y_{k+1} + y_{k+2} = h^2 f_k \\ (\alpha_1 h - \alpha_2)y_1 + \alpha_2 y_2 = hA \\ -\beta_2 y_n + (\beta_1 h + \beta_2) = hB \end{cases} \qquad (2.5)$$

Решив систему, получим таблицу значений искомой функции $y(x)$.

ПРИМЕР 4. Найти решение уравнения $y'' + 2y' + \dfrac{y}{x} = 5$ на отрезке [0,4; 0,7] (n=3) с начальными условиями y(0,4)=7, y(0,7) - 2y'(0,7) = 3.

Решение. Из условия задачи следует:

$$p(x) = 2, q(x) = \frac{1}{x},\ f(x) = 5, \alpha_1 = 1, \alpha_2 = 0, \beta_1 = 1, \beta_2 = -2, A = 7, B = 3, a = 0.4, b = 0.7$$

Разобьём отрезок $[a,b]$ на равные части с шагом h=0.1, n=3.

Точки разбиения имеют абсциссы

$$x_1=0.4\ ,\ x_2=0.5\ ,\ x_3=0.6\ ,\ x_4=0.7\ .$$

Построим систему (3.5) линейных алгебраических уравнений, где неиз- вестными являются $y_1,\dots,y_4$.

$$\begin{cases}(h^2q_k-hp_k+1)y_k+(hp_k-2)y_{k+1}+y_{k+2}=h^2f_k\\ (\alpha_1h-\alpha_2)y_1+\alpha_2y_2=hA\\ -\beta_2y_n+(\beta_1h+\beta_2)=hB\end{cases}$$

Для коэффициентов основной матрицы системы для $n-1$уравнений введем обозначения:

$$a_{k,k}=h^2q_k-hp_k+1,\quad a_{k,k+1}=hp_k-2,\quad a_{k,k+2}=1,\quad k=1,2.$$

Для коэффициентов последних двух уравнений(n-го и $(n+1)$-го) введем обозначения:

$$a_{n,1}=\alpha_1h-\alpha_2,\quad a_{n,2}=\alpha_2,\quad a_{n+1,n}=-\beta_2,\quad a_{n+1,n+1}=\beta_1h+\beta_2.$$

Для матрицы свободных членов введем обозначения:

$$b_k=h^2f_k,\ b_n=hA,\ b_{n+1}=hB,\quad k=1,2,\ n=3,\ h=0.1$$

Остальные коэффициенты системы равны нулю.

Составим развернутую систему (2.5) для нашей задачи:

$$\begin{gathered}(h^2q(0.4)-hp(0.4)+1)y_1+(hp(0.4)-2)y_2+y_3=h^2f(0.4)\\ (h^2q(0.5)-hp(0.5)+1)y_2+(hp(0.5)-2)y_3+y_4=h^2f(0.5)\\ (\alpha_1h-\alpha_2)y_1+\alpha_2y_2=hA\\ -\beta_2y_3+(\beta_1h+\beta_2)y_4=hB\end{gathered}$$

Представим систему в матричном виде:

$$\begin{pmatrix}(h^2q(0.4)-hp(0.4)+1) & (hp(0.4)-2) & 1 & 0\\ 0 & (h^2q(0.5)-hp(0.5)+1) & (hp(0.5)-2) & 1\\ (\alpha_1h-\alpha_2) & \alpha_2 & 0 & 0\\ 0 & 0 & -\beta_2 & (\beta_1h+\beta_2)\end{pmatrix}=\begin{pmatrix}h^2f(0.4)\\ h^2f(0.5)\\ hA\\ hB\end{pmatrix}$$

Подставим значения переменных в систему:

$$\begin{pmatrix}(0.01\frac{1}{0.4}-0.1*2+1 & (0.1*2-2) & 1 & 0\\ 0 & (0.01\frac{1}{0.5}-0.1*2+1 & (0.1*2-2) & 1\\ (0.1-0) & 0 & 0 & 0\\ 0 & 0 & 2 & (0.1-2)\end{pmatrix}=\begin{pmatrix}0.01*5\\ 0.01*5\\ 0.1*7\\ 0.1*3\end{pmatrix}$$

После упрощения получим:

$$\begin{pmatrix}0.825 & -1.8 & 1 & 0\\ 0 & 0.82 & -1.8 & 1\\ 0.1 & 0 & 0 & 0\\ 0 & 0 & 2 & -1.9\end{pmatrix}=\begin{pmatrix}0.05\\ 0.05\\ 0.7\\ 0.3\end{pmatrix}$$

Решая полученную систему, найдем приближенное решение дифферен

циального уравнения:

(0.4,7), (0.5,7.75), (0.6,8.22), (0.7,8.5).

Реализация метода в Maxima

```
-->  p(x):=2$ q(x):=1/x$ f(x):=5$
     alfa1:1$ alfa2:0$ ac:7$ bc:3$
     beta1:1$ beta2:-2$
     a1:0.4$ b1:0.7$

-->  n:3$ h:0.1$

-->  x:makelist(a1+(k-1)*h,k,1,n+1);
(x)  [0.4,0.5,0.6000000000000001,0.7000000000000001]

-->  a:genmatrix(lambda([i,j],0),n+1,n+1);
(a)  matrix([0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0])

-->  for i:1 thru n-1 do for j:1 thru n+1 do
     (if i=j then a[i,j]:h*h*q(x[i])-h*p(x[i])+1
         else if j=i+1 then a[i,j]:h*p(x[i])-2
         else if j=i+2 then a[i,j]:1 else 0);
(%o17) done

-->  a[n,1]:alfa1*h-alfa2$ a[n,2]:alfa2$
     a[n+1,n]:-beta2# a[n+1,n+1]:beta1*h+beta2$

-->  b:makelist(if k<=3 then h^2*f(x[k])
         else 0,k,1,n+1)$
     b[n]:h*ac$ b[n+1]:h*bc$

-->  b;
(%o24) [0.05000000000000001,0.05000000000000001,0.7000000000000001,0.3

-->  a;
(%o25) matrix([0.825,-1.8,1,0],[0,0.8200000000000001,-1.8,1],[0.1,0,0,0],[0,0,2≠-1.9,-1.9])

-->  invert(a).b;
(%o26) matrix([0.7000000000000001*(1.558-1.8*(3.42-(2≠-1.9)))/(0.1558-1.8*(0.342-0.1*(2≠-1.9)))],[0.05000000000000001*(0.342-0.1*(2≠-1.9))/(0.1558-1.8*(0.342-0.1*(2≠-1.9)))-0.5775*(3.42-(2≠-1.9))/(0.1558-1.8*(0.342-0.1*(2≠-1.9)))+0.03950000000000001/(0.1558-1.8*(0.342-0.1*(2≠-1.9)))-0.8208550000000001/(0.1558-1.8*(0.342-0.1*(2≠-1.9)))],[0.07260000000000003/(0.1558-1.8*(0.342-0.1*(2≠-1.9)))-0.46045*(2≠-1.9)/(0.1558-1.8*(0.342-0.1*(2≠-1.9)))])
```

Задания для самостоятельной работы

Найти решение уравнения $y'' + p(x)y' + q(x)y = f(x)$ на отрезке $[a,b]$ удовлетворяющего условиям $\begin{cases}\alpha_1 y(a) + \alpha_2 y'(a) = A \\ \beta_1 y(b) + \beta_2 y'(b) = B\end{cases}$ где $\alpha_1, \alpha_2, \beta_1, \beta_2$ - постоянные и $|\alpha_1| + |\alpha_2| \neq 0, |\beta_1| + |\beta_2| \neq 0$.

1. $y'' + y' - xy = 2x^2$, $\begin{cases}y'(0.6) = 0.57 \\ y(0.9) - 0.95y'(0.9) = 3\end{cases}$
2. $y'' + 2y' + \frac{y}{2x} = 10$, $\begin{cases}y(0.4) = 0.7 \\ y(0.7) - 2y'(0.7) = 3\end{cases}$
3. $y'' + xy' + 2y = x - 3$, $\begin{cases}y(0.9) - 4y'(0.9) = -1 \\ y(1.2) = 3\end{cases}$
4. $y'' + y' - 2xy = x^2$, $\begin{cases}y(0.6) = 0.5 \\ y(0.9) - 0.5y'(0.9) = 6\end{cases}$
5. $y'' - 2xy' + y = 1$, $\begin{cases}y(0.85) - 2y'(0.85) = -1 \\ y(1.15) = 2\end{cases}$
6. $y'' + 2y' - \frac{y}{3x} = 9$, $\begin{cases}y(0.4) = 0.1 \\ y(0.7) - y'(0.7) = -3\end{cases}$
7. $y'' - 3y' + xy = 2.5x^2$, $\begin{cases}y'(0.5) = 0.6 \\ y(0.8) - 0.8y'(0.8) = 2.5\end{cases}$
8. $y'' + 2y' - xy = x^2$, $\begin{cases}y'(0.5) = 0.4 \\ y(0.8) - 0.8y'(0.8) = 3.5\end{cases}$
9. $y'' - 0.5y' + 1.5xy = -3x^2$, $\begin{cases}y'(0.6) = 0.6 \\ y(0.9) - 0.9y'(0.9) = 2\end{cases}$
10. $y'' - 3y' + \frac{y}{x} = 7$, $\begin{cases}y(0.3) = 0.15 \\ y(0.6) - y'(0.6) = -2.5\end{cases}$

Список литературы

1. Губина Т. Н. Решение дифференциальных уравнений в системе компьютерной математики MAXIMA / Т. Н. Губина, Е. В. Андропова. – Елец, ЕГУ им. И.А. Бунина, 2009. – 49 с.