ПРЕДМЕТНОЕ ОБУЧЕНИЕ
ОТКРЫТОЕ ПО В ШКОЛЕ

Иванов Олег Александрович

MAXIMA В ОБУЧЕНИИ МАТЕМАТИКЕ В ШКОЛЕ. УРОК 3. ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ В MAXIMA

Эта статья посвящена алгоритмам и способам их реализации в пакете Maxima.

Основными новыми операторами, которые будут введены на этом уроке, являются: оператор `block` и операторы цикла (в различных синтаксических формах).

При задании некоторой процедуры операторы можно заключать в скобки. К примеру, если `f(x):=(expr1,expr2,..., exprn)`, то значением функции `f(x)` является значение последнего из вычисленных выражений. Однако при проведении вычислений часто приходится использовать вспомогательные переменные, которые, естественно, имеют простые имена. При этом следует учитывать, что если переменная с именем *x* в одной процедуре получила некоторое значение, то это значение она сохранит и далее. Если в другой процедуре есть переменная с тем же именем, то есть опасность случайно использовать не то ее значение, которое хотелось бы. Для недопущения подобных коллизий во всех языках программирования есть возможность ввести так называемые *локальные переменные*, сохраняющие свои значения только в определенных пределах (*блоках*). Внутри оператора `block` как раз и можно ввести подобные переменные, поместив их имена в определенный список. К примеру, текст `block([a:1,b],b:x+a)` означает, во-первых, что переменные с именами *a* и *b* будут иметь значения только при выполнении одного-единственного оператора, во-вторых, что переменной *a* задано начальное значение 1. Если мы хотим, чтобы некоторое значение было доступно после исполнения оператора, то следует использовать оператор `return` (см. первый пример далее).

Циклы – неотъемлемая часть программирования. Напишем процедуру, вычисляющую сумму первых *n* нечетных чисел (см. рис. 1).

Здесь *i* есть переменная цикла или итератор, для которого указываются его начальное и конечное значения, а также шаг. Запись `for i:i0 thru i1 step d do (...)` означает, что вычисление операторов, находящихся в скобках после `do` (в теле процедуры) будут производиться для значений итератора *i*, равных *i*0, *i*0+*d*, и для всех последующих членов арифметической прогрессии с первым членом *i*0 и раз-

```
(%i1)  oddsum(n):=block([s:0],
       for i:1 thru 2*n-1 step 2 do s:s+i, return(s))$
```

Рис. 1

ностью d, не превосходящих значения $i1$. Если значение d явно не указано (в синтаксической конструкции опущено `step d`), то на каждом шаге цикла значение i увеличивается на 1.

Таким образом, в процедуре `oddsum` вычисление производится следующим образом. Начальное значение локальной переменной s равно 0.

Шаг 1: итератор i равен 1, следовательно $s = 0 + 1 = 1$.

Шаг 2: значение i увеличилось на 2, так что $i = 3$ и $s = 1 + 3 = 4$.

И так далее.

На последнем шаге значение i будет равно $2n - 1$.

Накопленная в локальной переменной s сумма есть значение данной процедуры, поскольку переменная s является аргументом оператора `return`.

Заметим, что подобное суммирование осуществляет уже известный нам оператор `sum`, поэтому в данном примере естественнее было написать так, как показано на рис. 2.

При помощи циклов можно осуществлять непосредственный перебор возможных вариантов (см. далее пример 3), а можно реализовывать конкретные алгоритмы.

Пример 1. Нахождение наибольшего общего делителя.

Исторически первым алгоритмом, видимо, является алгоритм Евклида поиска наибольшего общего делителя двух натуральных чисел. Его идея основана на том, что число d является одновременно делителем чисел a и b тогда и только тогда, когда оно является делителем чисел $a - b$ и b. Будем последовательно переходить от пары $(a; b)$, где $a > b$, к паре $(a - b; b)$. Если на некотором шаге получается, что первое число меньше второго, то меняем их местами. К примеру, для чисел 36 и 15 мы получим следующую последовательность пар

$$(36,15)\mapsto(21,15)\mapsto(6,15)\mapsto(15,6)\mapsto$$

$$\mapsto(9,6)\mapsto(3,6)\mapsto(6,3)\mapsto(3,3).$$

Значит, наибольший общий делитель пары (36, 15) равен наибольшему общему делителю пары (3, 3), равному 3. Сейчас мы реализуем этот алгоритм (см. рис. 3).

Смысл операторов `max` и `min` понятен, значением первого из них является наибольший из его аргументов, значением второго – наименьший. `#` – это знак неравенства, условие `a#b` означает, что $a \neq b$. Оператор `return` прекращает исполнение функции, внутри которого он находится. Поэтому на том шаге вычислений, на котором числа a и b стали равны друг другу, работа процедуры будет завершена.

Посмотрим на скорость работы оператора `gcd1`.

```
(%i2)  gcd1(2^18-1,2^4-1);
(%o2)  3
(%i3)  time(%o2);
(%o3)  [3.07]
```

```
(%i2)  oddsum1(n):=sum(2*i-1,i,1,n)$ oddsum1(7);
(%o3)  49
```

Рис. 2

```
(%i1)  gcd1(x,y):=block([a:max(x,y),b:min(x,y)],
       do if a#b then [a,b]:[max(a-b,b),min(a-b,b)]
       else return(a))$
```

Рис. 3

Как вы видите, для вычисления наибольшего общего делителя чисел 15 и 262143 процедуре `gcd1` потребовалось целых 3 секунды. Работу процедуры можно существенно ускорить, если переходить от пары (a, b) к паре (b, r), где r – это остаток от деления a на b, то есть $r = a \pmod b$. При этом в момент, когда очередной остаток r от деления большего числа на меньшее оказывается равным нулю, меньшее число в последней паре является наибольшим общим делителем исходной пары чисел. Реализуем эту идею (см. рис. 4).

Значением оператора `mod(a,b)` и является остаток от деления a на b.

Как вы видите, скорость работы значительно увеличилась.

Пример 2. Позиционная запись натуральных чисел.

Поставим задачу нахождения цифр заданного числа. Казалось бы, о чем здесь говорить, ведь цифрами, к примеру, числа 123 являются 1, 2 и 3. Однако задача не столь тривиальна, как кажется на первый взгляд. Попробуйте найти степень 2, десятичная запись которой начинается с 7. Конечно, можно просто последовательно вычислять числа 2^n и смотреть на экране, не появится ли впереди «семерка». Однако ясно, что подобный подход непродуктивен.

Наша цель – написать процедуру, значением которой является список цифр данного натурального числа, причем ясно, что будет разумно рассматривать сразу случай системы счисления с произвольным основанием (конечно, если основание больше 10, то «цифра» может оказаться многозначным числом), взяв его в качестве второго аргумента этой процедуры.

Имеем, $123 = 10 \cdot 12 + 3$, поэтому $3 = 123 \pmod{10}$, а $12 = (123 - 3)/10$. В общем случае, если p есть основание системы счисления и $n = a_{n-1} \cdot p^{n-1} + a_{n-2} \cdot p^{n-2} + \ldots + a_1 \cdot p + a_0$, то $a_0 = n \pmod p$. Теперь вместо числа n рассмотрим число $(n - a_0)/p$, его последняя цифра равна предпоследней цифре числа n. Напишем соответствующую процедуру (см. рис. 5).

Оператор `append(u,v)` производит слияние списков u и v, его результатом будет список $u_1, u_2, \ldots, v_1, v_2, \ldots$ В нашей процедуре цифра, найденная на текущем шаге цикла, добавляется справа к списку ранее найденных цифр (см. рис. 6).

```
(%i4)  gcd2(x,y):=block([a:max(x,y),b:min(x,y)],
       do if b#0 then [a,b]:[b,mod(a,b)]
       else return(a))$
(%i5)  gcd2(2^18-1,2^4-1);
(%o5)  3
(%i6)  time(%o5);
(%o6)  [0.01]
```

Рис. 4

```
(%i1)  intdig(n,p):=block([x:n,a:[mod(n,p)]],
       do (x:(x-first(a))/p, if x#0 then
       a:append([mod(x,p)],a) else return(a)))$
```

Рис. 5

Рассмотрим пример использования этой процедуры: найдем наименьшую степень двойки, десятичная запись которой начинается с цифры 7 (см. рис. 7).

Как вы видите, такая степень действительно существует!

Обратная операция заключается в восстановлении числа по списку его цифр. В процедуре `fromdig` используется следующая удобная для нас модификация оператора цикла.

Если *a* – это некоторый список, то в синтаксической конструкции

```
for x in a do (…)
```

переменная *x* будет принимать значения, равные последовательным элементам этого списка (см. рис. 8).

Идея этой процедуры проста. Если $a=[a_{n-1}, a_{n-2}, \ldots, a_0]$ есть список цифр числа *n* в системе счисления с основанием *p*, то $n=(\ldots(a_{n-1}\cdot p+a_{n-2})\cdot p+\ldots)\cdot p+a_0$.

Пример 3. Решение задачи о «счастливых билетах»

Давным-давно на автобусном (трамвайном, троллейбусном) билете стоял номер, состоявший из 6 цифр. Билет назывался «счастливым», если сумма первых трех цифр его номера оказывалась равной сумме его трех последних цифр. Задача состоит в том, чтобы вычислить количество всех счастливых билетов среди миллиона билетов (с номерами от 000000 до 999999).

Будем решать эту задачу перебором, однако – «разумным». Можно, конечно, просмотреть весь миллион номеров, установив для каждого из них, является ли соответствующий билет «счастливым». Однако оказывается, что достаточно перебрать не миллион, а всего лишь тысячу номеров. То, что в результате вычислений, представленных на рис. 9, мы получим число «счастливых билетов», – упражнение для читателя.

Кроме того, с задачей о «счастливых билетах» связана интересная математика. Объяснение того, что следующий оператор дает ответ к задаче, дано в [1: см. решение задачи 2.39], а также на рис. 10.

```
(%i2)  intdig(1357,10); intdig(13,2);
(%o2)  [1,3,5,7]
 (%o3)  [1,1,0,1]
```

Рис. 6

```
(%i7)  n:1$ x:2$ do(n:n+1,x:2*x,
       if first(intdig(x,10))=7 then return(n));
(%o9)  46
(%i10)  2^46;
(%o10)  70368744177664
```

Рис. 7

```
(%i4)  fromdig(a,p):=block([n:0], for x in a
       do n:n*p+x, return(n))$
(%i5)  fromdig([1,1,0,1],2);fromdig([1,3,5,7],10);
(%o5)  13
 (%o6)  1357
```

Рис. 8

Результатом работы оператора `coeff(p,x,n)` является коэффициент при x^n в многочлене p.

Пример 4. Вычисление чисел Фибоначчи

Напомним, что числа Фибоначчи определяются рекуррентным соотношением $F_n = F_{n-1} + F_{n-2}$ при $n \geq 2$, а $F_1 = F_0 = 1$. Попробуем написать процедуру для вычисления этих чисел. Казалось бы, никаких проблем здесь быть не может. Перед вами три процедуры (см. рис. 11).

Первая из них традиционна. Во второй процедуре почему-то аргумент находится в квадратных скобках, а не в круглых, при этом вместо использования условного оператора для определения первых двух чисел явно указаны их значения. В процедуре `fibon3` для вычисления n-го числа Фибоначчи просто сделано $n-1$ сложение. Как вы видите, каждая из этих процедур быстро и правильно нашла первые 11 чисел Фибоначчи.

Тем не менее, разница между этими процедурами огромна. Для того чтобы ее ощутить, попробуем найти число F_{100} (см. рис. 12).

```
(%i1)  t:makelist(0,i,0,27)$
       for i:0 thru 9 do for j:0 thru 9 do
       for k:0 thru 9 do t[i+j+k+1]:t[i+j+k+1]+1$
       sum(t[i]^2,i,1,28);
(%o3)  55252
```

Рис. 9

```
(%i4)  coeff(expand(sum(x^i,i,0,9)^6),x,27);
(%o4)  55252
```

Рис. 10

```
(%i1)  fibon1(n):=if n=0 then 1 elseif
       n=1 then 1 else fibon1(n-1)+fibon1(n-2)$
(%i2)  makelist(fibon1(i),i,0,10);
(%o2)  [1,1,2,3,5,8,13,21,34,55,89]
(%i3)  fibon2[0]:1$ fibon2[1]:1$
       fibon2[n]:= fibon2[n-1]+fibon2[n-2]$
(%i6)  makelist(fibon2[i],i,0,10);
(%o6)  [1,1,2,3,5,8,13,21,34,55,89]
(%i7)  fibon3(n):=block([a:1,b:1],if n=0 then
       return(a) elseif n=1 then return(b) else
       (thru n-1 do [a,b]:[b,a+b], return(b)))$
(%i8)  makelist(fibon3(i),i,0,10);
(%o8)  [1,1,2,3,5,8,13,21,34,55,89]
```

Рис. 11

Автору не удалось дождаться окончания работы оператора `fibon1(100)`, поэтому работа этого оператора была прервана.

```
(%i14)  fibon2[300]$
(%i15)  time(%o14);
(%o15)  [0.01]
```

Как видите, оператор `fibon2` вычислил трехсотое число Фибоначчи очень быстро. Оператор `fibon3` очень быстро нашел даже тысячное число Фибоначчи.

```
(%i16)  fibon3(1000)$
(%i17)  time(%o16);
(%o17)  [0.03]
```

При этом при попытке найти это число посредством оператора `fibon2` произошло переполнение (см. рис. 13).

В чем же дело, в чем разница между тремя операторами, которые, на первый взгляд, делают одно и то же?

Начнем исследование с анализа второго из этих операторов. Синтаксическая конструкция вида `a[n]:x` определяет значение n-го члена последовательности, поэтому `fibon2` есть имя последовательности, а не числовой функции. Для того, чтобы найти, к примеру, значение F_4, надо знать F_3 и F_2, при этом $F_2 = F_1 + F_0 = 1+1 = 2$, а $F_3 = F_2 + F_1 = 2+1 = 3$. Оператор `fibon2` так и поступит. Сначала будет найдено значение F_2, затем – значение F_3 и, наконец, F_4. Обратите внимание на то, что в процессе вычисления будут запоминаться уже найденные значения. Поэтому если найдено значение `fibon2(300)`, то для поиска, например, значения `fibon2(287)` вычисления производиться не будут, оно уже находится в памяти компьютера. С другой стороны, с этим и связана причина переполнения при поиске значения `fibon2(1000)`: программа должна вычислить и сохранить в памяти компьютера значения всех предыдущих чисел Фибоначчи!

Оператор `fibon1` вычисляет по-другому. Для него значение, к примеру, F_4 есть сумма $((F_1 + F_0) + F_1) + (F_1 + F_0)$, для вычисления которой компьютеру придется произвести не три, а четыре сложения. Казалось бы, невелика разница, однако она окажется огромной при вычислении чисел Фибоначчи с большими номерами (смотрите задание 3 к этому уроку). Конечно, $F_1 + F_0 = F_2$, однако при поиске F_4 это значение будет вычислено дважды!

Наконец, для вычисления значения F_n последний из рассмотренных операторов сделает просто $n-1$ сложение, затратив

```
(%i9)  fibon1(100);
Maxima encountered a Lisp error:
   Console interrupt.
  Automatically continuing.
  To reenable the Lisp debugger set *debugger-hook* to nil.
```

Рис. 12

```
(%i18)  fibon2[1000]$
Maxima encountered a Lisp error:
   Error in PROGN [or a callee]: Bind stack overflow.
  Automatically continuing.
  To reenable the Lisp debugger set *debugger-hook* to nil.
```

Рис. 13

на это немного времени и ресурсов компьютера.

Синтаксическая конструкция `thru k do (…)` означает, что вычисление тела цикла будет произведено k раз.

Таким образом, как вы видели, даже формально верная процедура может оказаться фактически бесполезной.

Задания для самостоятельной работы

1. Найдите и докажите формулу для наибольшего общего делителя чисел $2^n - 1$ и $2^k - 1$.

2. Существует ли, кроме 11, еще хотя бы одно простое число, записываемое одними единицами? (Проверку числа на простоту произведите при помощи оператора `primep`).

3. Найдите частоты появления каждой из цифр от 1 до 9 как первой цифры десятичной записи первых 100 степеней двойки.

4. Объясните, почему результатом вычисления первой из программ в примере 3 действительно является количество всех «счастливых билетов».

5. Сколько сложений будет произведено при вычислении числа Фибоначчи F_6 в процедуре `fibon1`? Найдите и докажите также общую формулу для числа сложений в процедуре `fibon1` при вычислении n-го числа Фибоначчи.

6. Ясно, что n-ую степень числа можно найти, сделав $n - 1$ умножение. Постарайтесь вычислить x^{1001} при помощи не 1000, а гораздо меньшего числа умножений. Опишите алгоритм, посредством которого степень числа может быть найдена за не очень большое число умножений, и напишите процедуру, реализующую этот алгоритм.

Ответы, указания и решения заданий для самостоятельной работы урока 2

1. Поскольку данное выражение равно a при $x = a$, равно b при $x = b$ и равно c при $x = c$, то оно тождественно равно x. Следовательно, справедливы следующие равенства:

$$\frac{c}{(c-a)(c-b)}+\frac{a}{(a-b)(a-c)}+\frac{b}{(b-a)(b-c)}=0,\quad \frac{c(a+b)}{(c-a)(c-b)}+\frac{a(b+c)}{(a-b)(a-c)}+\frac{b(a+c)}{(b-a)(b-c)}=0$$

и, $\frac{abc}{(c-a)(c-b)}+\frac{abc}{(a-b)(a-c)}+\frac{abc}{(b-a)(b-c)}=0$, впрочем, последнее тождество уже известно. Нетрудно понять, что аналогичным образом доказывается и тождество $\frac{c^2(x-a)(x-b)}{(c-a)(c-b)}+\frac{a^2(x-b)(x-c)}{(a-b)(a-c)}+\frac{b^2(x-a)(x-c)}{(b-a)(b-c)}=x^2$, которое тоже имеет три следствия. Другое дело, что непонятно, чему будет равно выражение $\frac{c^3(x-a)(x-b)}{(c-a)(c-b)}+\frac{a^3(x-b)(x-c)}{(a-b)(a-c)}+\frac{b^3(x-a)(x-c)}{(b-a)(b-c)}$, и, в частности, упрощается ли сумма $\frac{c^3}{(c-a)(c-b)}+\frac{a^3}{(a-b)(a-c)}+\frac{b^3}{(b-a)(b-c)}$. Воспользуйтесь Maxima, чтобы получить ответ, а затем доказать «на бумаге» предполагаемое равенство.

2. Достаточно доказать, что всякий ненулевой многочлен степени n имеет не более n корней. Проведите рассуждение по индукции, используя то, что если число a является корнем многочлена $p(x)$, то существует многочлен $q(x)$, являющийся частным от деления $p(x)$ на $x - a$, то есть $p(x) = (x - a)q(x)$.

3. Из следующего рис. 1, на котором изображены график $y = x^6 6^{-x}$ и горизонтальная прямая $y = 1$, очевидно, что данное уравнение имеет три решения.

Более того, достаточно ясно, как это доказать. Для этого достаточно исследовать рассматриваемую функцию, то есть найти промежутки ее возрастания и убывания.

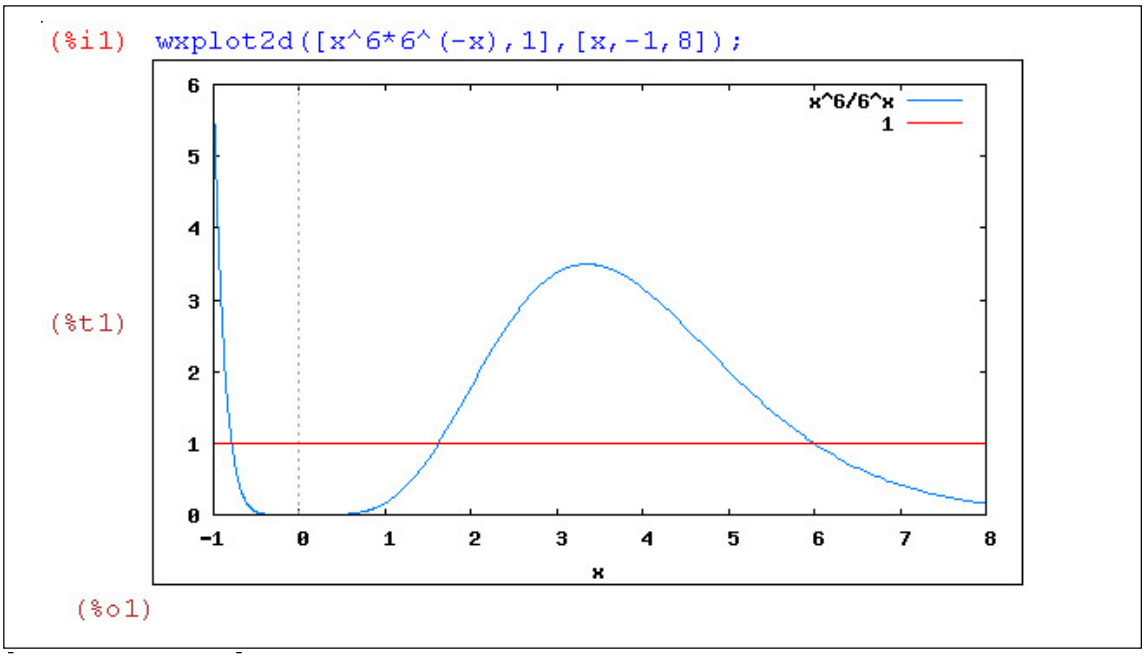

Рис. 1

7. а) Изобразим графически поведение последовательности $x_n = n^2/2^n$ (рис. 2).

Мы видим, что $x_1 < x_2 < x_3 < x_4 > x_5 > \ldots$ Первые три неравенства проверяются непосредственно. Осталось доказать, что $n^3/2^n > (n+1)^3/2^{n+1}$ при $n \geq 3$, или $(1+1/n)^3 < 2$, что верно, так как $(5/4)^3 = 125/64 < 2$.

Для решения пункта б) этой задачи проще изобразить вначале график соответствующей функции (рис. 3).

Мы видим, что наибольшее значение достигается, видимо, при n = 14. Однако,

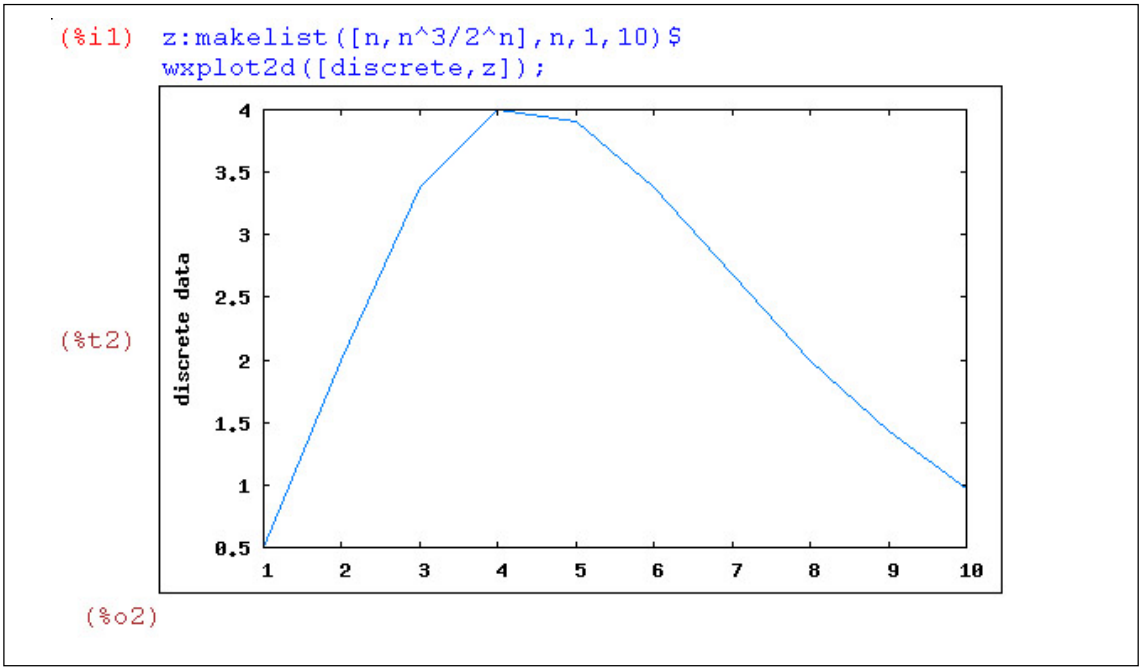

Рис. 2

Рис. 11

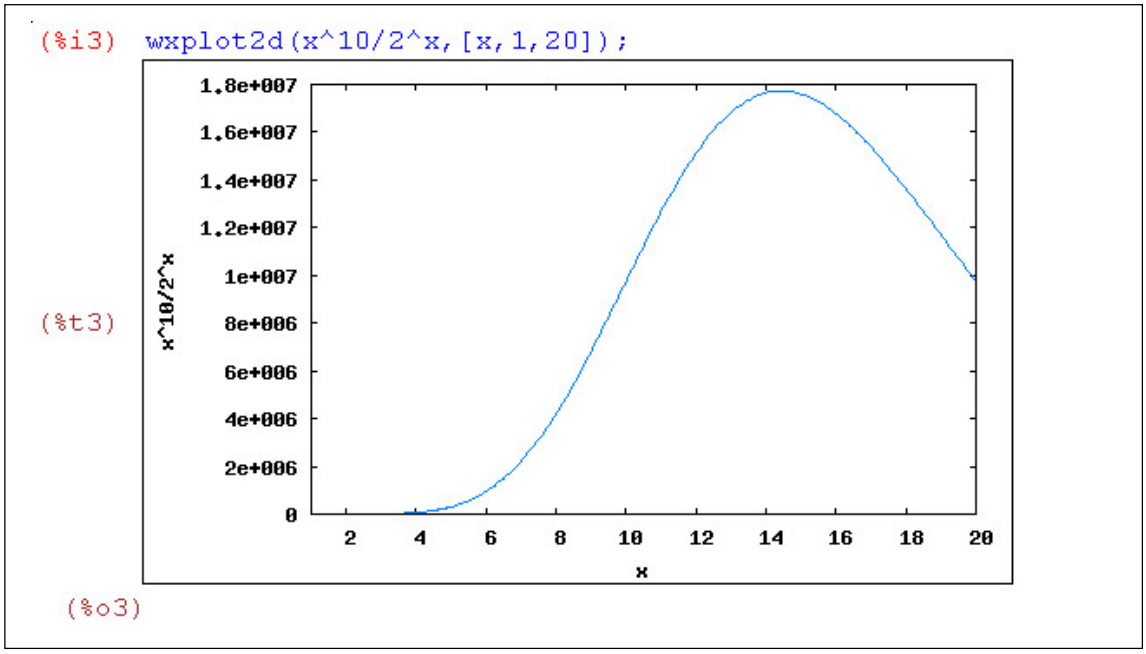

Рис. 3

чтобы быть в этом вполне уверенными, посмотрим на поведение последовательности x_n при n близких к 14 (рис. 4).

Сомнений в справедливости не остается. Докажите этот факт самостоятельно.

5. Попробуйте вначале нарисовать параболы $y=1-x^2$ и $x=1-y^2$ «на бумаге». В скольких точках они у вас пересеклись? После этого взгляните на рис. 5.

Таким образом, при a = 1 система имеет четыре решения. Если мы продолжим увеличивать значение a, то параболы станут более «крутыми», поэтому система по-

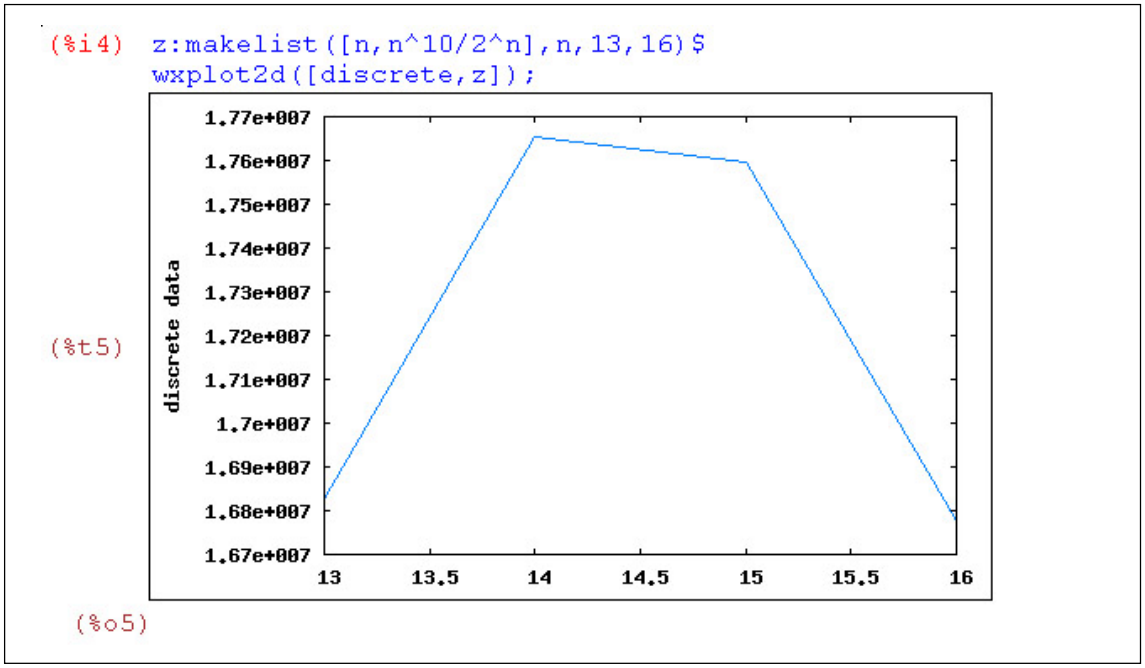

Рис. 4

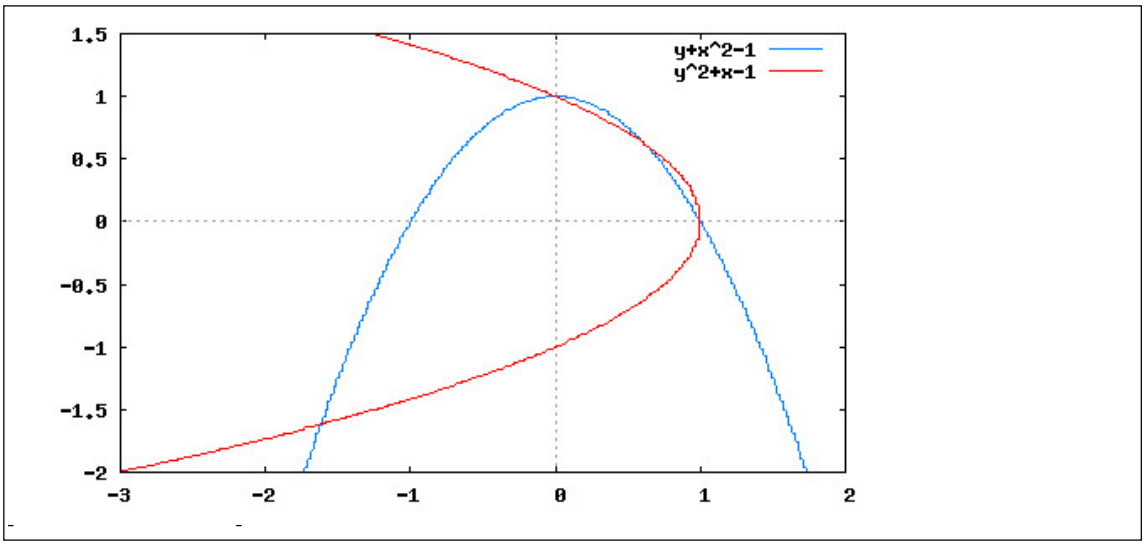

Рис. 5

прежнему будет иметь 4 решения. Однако как будет меняться число ее решений при уменьшении значения a? Вряд ли вам удастся «поймать» искомое значение параметра, глядя на графики. Можно поступить проще, попросив Maxima решить уравнение четвертой степени.

```
(%i3)  y:1-a*x^2$ solve(x-1+a*y^2,x);
```

$$(\%o4)\quad \left[x=-\frac{\sqrt{4a-3}-1}{2a},\ x=\frac{\sqrt{4a-3}+1}{2a},\ x=-\frac{\sqrt{4a+1}+1}{2a},\ x=\frac{\sqrt{4a+1}-1}{2a}\right]$$

Из приведенного ответа ясно видно, что это уравнение должно распасться на два квадратных. Найдите соответствующее разложение. Контрольный вопрос: сколько решений имеет данная система при $a = 3/4$?

6. Первый вопрос: а так ли уж важно, что первый член последовательности равен 2? Из приведенного первого решения задачи 4 очевидно, что это никакого значения не имеет. Именно, при любом значении $x_1 > 0$ последовательность x_n будет иметь своим пределом число $\sqrt{2}$. Второй вопрос: а откуда взялась рекуррентная формула, задающая заданную последовательность? Ответ: это результат применения метода касательных (метода Ньютона) поиска приближенного решения уравнения $x^2 = 2$. Теперь посмотрим на скорость сходимости к $\sqrt{2}$ последовательности с начальным данным, к примеру, $x_1 = 4$. Оказывается, что пятый член такой последовательности дает приближение только с 4 знаками после запятой. В чем же дело, почему настолько хороша именно последовательность с начальным членом, равным 2? Разгадка связана с понятием *цепной дроби* (см., к примеру, [1: гл. 9, упр. 9.14]).

Литература

1. *Иванов О.А.* Элементарная математика для школьников, студентов и преподавателей. М.: МЦНМО, 2009. 384 с.

Иванов Олег Александрович, профессор, доктор педагогических наук и кандидат физико-математических наук, профессор кафедры общей математики и информатики СПбГУ.