Программирование в Mathcad

1. Немного истории

Средства программирования появились в 6-й версии Mathcad, а некоторые их дополнения и изменения в 7-й (операторы `break`, `continue` и `return`), 12-й (локальная функция — *см. разд. 8*) и в 13-й версиях (средства отладки — *см. разд. 7*).

Несмотря на то, что пакет Mathcad возник как прямая альтернатива программированию, при работе в этой среде всегда ощущалась потребность в программировании для расширения и совершенствования базового набора математических инструментов — операторов и функций. Более-менее опытные пользователи решали эту проблему тремя путями.

Путь 1. В самых первых версиях Mathcad были две функции (`if` и `until`), позволявшие через различные хитрости и трюки реализовывать две основные *алгоритмические конструкции — выбор* (`if`) и *повторение* (`until`). Хитрить же приходилось из-за неспособности функций `if` и `until` иметь в качестве аргументов несколько операторов. Поэтому для реализации даже несложного алгоритма нужно было подключать механизм *вложенных* функций и операторов, что нередко превращало *программу* в настоящую *криптограмму*, в которой даже сам автор разбирался с трудом.

Вот как, например, выглядит поиск корня уравнения (нуля функции) методом половинного деления (рис. 1) с использованием функций `if` и `until`. Читатель может сравнить "программу" на рис. 1 с программой (без кавычек) на рис. 2.6, реализующей практически тот же алгоритм: отрезок неопределенности [`a`, `b`] делится пополам, и смотрится, где расположен корень; соответствующая половинка снова делится пополам, и так до тех пор (`until` — рис. 1 или `while` — рис. 2.6), пока не будет достигнута нужная точность расчета.

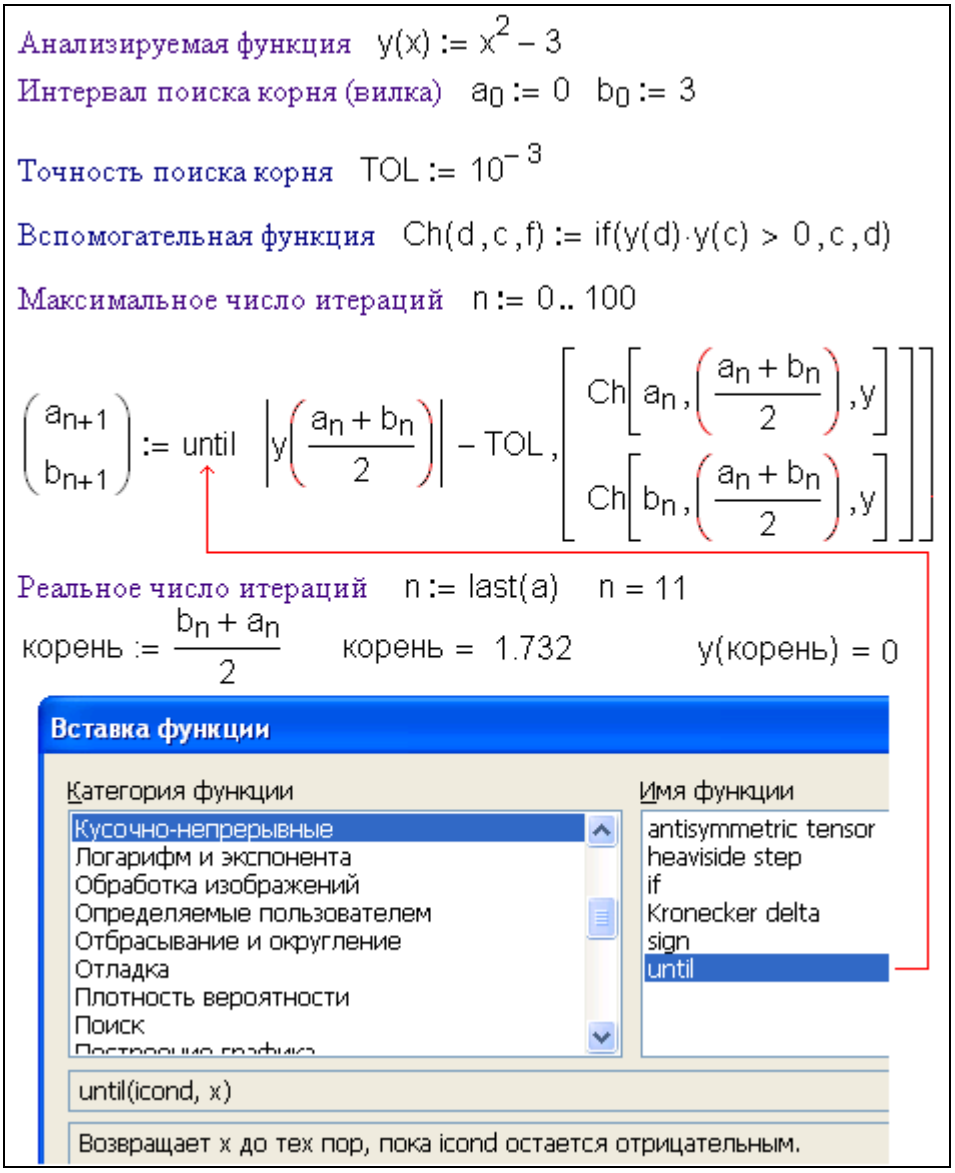

Рис. 1. Поиск корня алгебраического уравнения методом половинного деления с использованием функций `if` и `until`

Кроме "зашифрованности" алгоритма (чего стóят аргументы функции `until` на рис. 1) "беспрограммный" поиск корня имеет и другой недостаток: он приводит к нерациональному использованию ресурсов компьютера — к генерации векторов `a` и

b[1], у которых нас интересуют только последние (last) элементы, между которыми зажат искомый корень (переменная корень).

Операторы if и until позволяют *менять естественный порядок* выполнения операторов в Mathcad-документе сверху вниз и слева направо на более сложный. Кроме того, есть еще два признака программирования: *локальные переменные*, а также и *функции*, объединение операторов в *операторные блоки*. Но об этом речь пойдет чуть ниже.

> ***Примечание***
> Функция until в 2000-й версии перешла в разряд недокументированной, в 11-й ее совсем изъяли из Mathcad, а в 12-й она вернулась в инструментарий программирования (см. п. 13 в *"Предисловии к четвертому изданию книги…"*, где перечислены новинки Mathcad 12).

Путь 2. Версии Mathcad, начиная с 4.0, — это полноценные *Windows-приложения*. Поэтому при решении конкретной задачи в среде Mathcad можно в статике (через файлы на диске или через буфер обмена) или в динамике (технологии DDE и OLE — *см. разд. 9*) перенести данные (скаляр, вектор или матрицу) в среду, например FORTRAN, и, используя богатый набор средств вычислительной математики этого языка программирования, решить задачу (этап задачи). Начиная с 5-й версии Mathcad, пользователям была предоставлена возможность программирования на языке C и объявления в среде Mathcad новых встроенных функций (операторов). Код этих функций нужно откомпилировать каким-либо 32-разрядным транслятором и прикрепить к среде Mathcad через механизм DLL, опираясь на соглашение UserEFI. Но этот путь с самого начала был в чем-то тупиковым. Во-первых, Mathcad создавался как инструмент решения широкого класса задач теми, кто не хотел или не умел возиться с классическими языками программирования, и мы это уже не раз подчеркивали. При обращении же к языку C получалось так, что от чего ушли, к тому и пришли. Во-вторых, тот, кто все-таки переключался из среды Mathcad в среду языка C, как правило, там и оставался, решая всю задачу целиком. В-третьих (вернее, во-вторых с половиной), если кто-то и мог решить свою задачу на языке C, то он обычно не пользовался услугами Mathcad по неким "моральным соображениям", считая это ниже своего достоинства[2]. Но главным недостатком в технологии использования языка C для расширения возможностей Mathcad является невозможность включения в C-программу богатого математического инструментария Mathcad.

Путь 3. Последние версии Mathcad оборудованы элементами интерфейса Controls, поддерживаемыми программами на языках JScript или VBScript. Этим можно воспользоваться не только для форматирования самих Controls, но и для написания программ, особенно тем пользователям Mathcad, кто силен в языке BASIC. Так на рис. 2

---

[1] С другой стороны эти векторы хранят историю решения задачи, что полезно при отладке *(см. разд. 7)*.

[2] Бытовало даже такое мнение, что математические пакеты — это средство решения задач на компьютере *ленивыми*. "Ленивые" же резонно на это возражали, что все блага цивилизации — это их (ленивых) изобретения.

показано, как можно в цикле определить сумму чисел S от a до b, вставив в Mathcad-документ Controls — текстовое поле с возможностью ввода двух переменных a и b (по умолчанию количество вводов **Number of Inputs** равно нулю) и одним (умолчание) выводом (**Number of Outputs**) для переменной S.

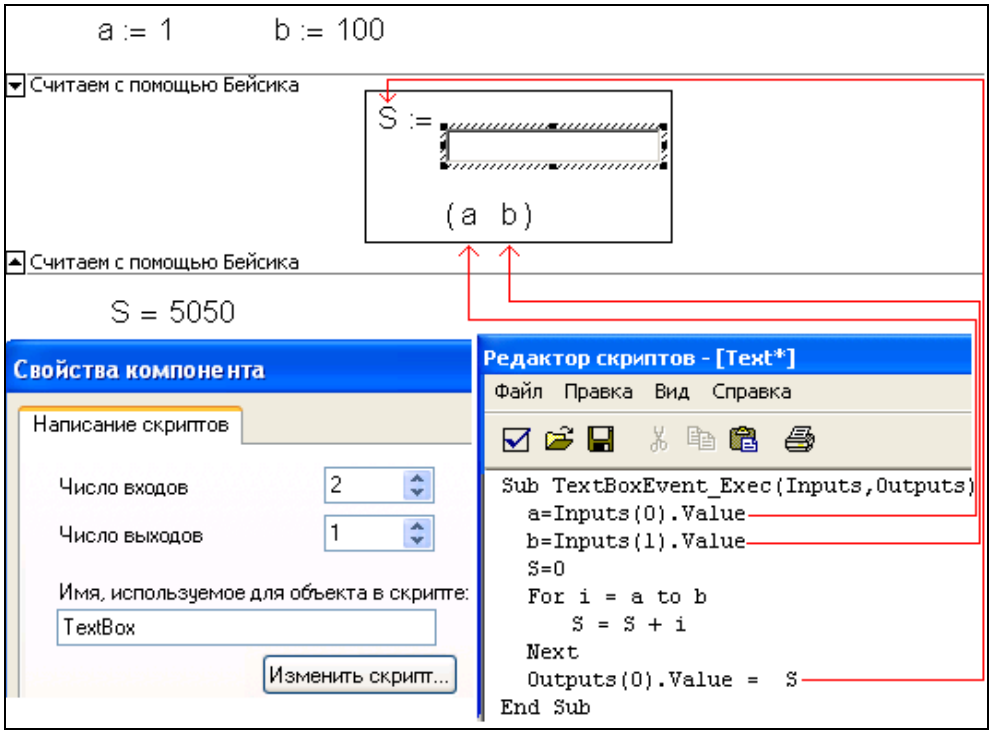

Рис. 2. Программа, введенная в Mathcad через текстовое поле

Как понимает читатель, через текстовое поле, показанное на рис. 2, как через "игольное ушко" можно "протащить" в Mathcad-документ довольно сложные BASIC-программы и решить поставленную задачу. Недостатки здесь известные. Во-первых, и это уже отмечалось, скрипты нежелательны в Mathcad-документах, т. к. через них можно "протащить" не только "невинные" программы (см. рис. 2), но и вирусы — "чуму" компьютеров.

Во-вторых, программы, введенные в Mathcad-документы через Controls, по-прежнему отрезаны от мощнейших инструментов Mathcad — в них нельзя вставлять функции и операторы решения уравнений и систем, линейной алгебры, оптимизации и всего того, что составляет суть данного математического пакета. Программирование в Controls, как правило, ведется только для улучшения интерфейса.

2. Панель программирования

С первого взгляда на Mathcad возникает вопрос: почему в него не был встроен какой-либо известный и широко используемый язык программирования, а разработан новый, ни на что не похожий?

> ***Примечание***
> Язык программирования Mathcad, развиваясь в 7-й версии, стал все больше приобретать черты языка C. Здесь нет ничего удивительного, т. к. сам Mathcad создается на этом языке.

Электронные таблицы Excel, текстовый редактор Word и система управления базами данных Access (Microsoft Office) используют встроенный язык BASIC[3], что кажется естественным даже для самых непримиримых противников этого языка. Но после детального знакомства с языком Mathcad удивление сменяется пониманием и даже удовлетворением. Становится очевидным, и это уже подчеркивалось, что в рамки традиционных языков с их программами в текстовом формате невозможно втиснуть богатый набор инструментария Mathcad, который реализован не только и не столько в виде функций, сколько в общепринятом в математике виде (см., например, оператор производной на рис. 3.6, вставленный в Mathcad-программу[4]).

В Mathcad, по сути, не встроен язык программирования, а просто снято вышеупомянутое *ограничение* на использование составных операторов в теле алгоритмических управляющих конструкций выбора и повторения. Кроме того, как было уже отмечено, введено понятие *локальной переменной* и *функции*, добавлен цикл с параметром `for`, операторы досрочного выхода из цикла `break` и `continue`, а также оператор досрочного выхода из программы `return`.

Теперь о технике Mathcad-программирования.

Алгоритмические конструкции в среде Mathcad вводятся не традиционным набором через клавиатуру ключевых слов `if`, `then`, `else`, `while` и т. д., а нажатием одной из кнопок панели инструментов **Программирование**.

---

[3] Точнее VBA — Visual Basic for Applications.

[4] Исключение составляют только функции, работающие в паре с ключевым словом `Given`. Их нельзя вставить в программу.

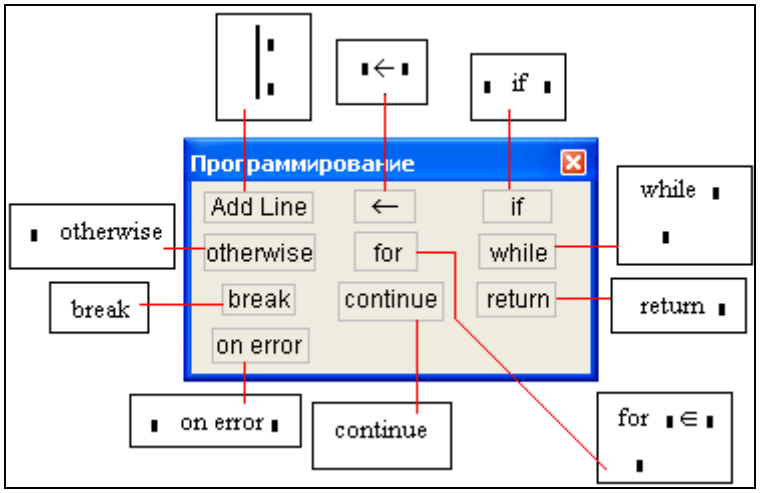

Рис. 3. Панель **Программирование** Mathcad

Щелчок по одной из десяти кнопок, показанных на рис. 3, создает на дисплее заготовку соответствующей программной конструкции.

Опишем их.

Кнопка **Add Line** добавляет строку в программу, в тело цикла, в ветвь альтернативы и т. д. Этим действием снимается вышеупомянутое ограничение на число операторов во вложенных конструкциях языка (рис. 4).

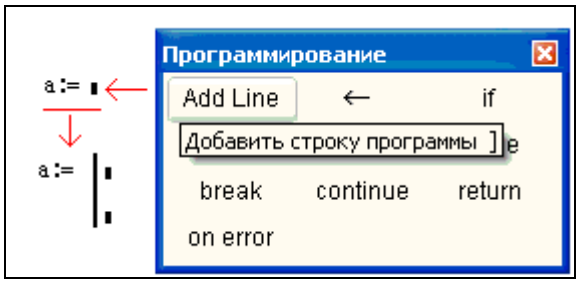

Рис. 4. Вставка строки программы

Новая строка программы возникает либо выше, либо ниже текущей, где находится курсор, в зависимости от вида курсора: ⌟ — строка появится внизу, ⌞ — вверху. Положение курсора меняется после нажатия клавиши <Insert>.

Вертикальная линия объединяет отдельные операторы в *операторный блок* с одним входом и одним выходом, который выполняется как единый оператор (один из трех атрибутов структурного программирования). Какое-то подобие операторного блока пользователь Mathcad часто выделяет и в беспрограммном документе, реализуя, например, метод последовательных приближений или зажимая операторы ключевым

словом `Given` и функцией `Find` (`MinErr`, `Minimize`, `Maximize`, `Odesolve` и `Pdesolve`).

Кнопка **Add Line** в чем-то подобна кнопке создания вектора, объединяющего в единый блок скалярные величины. Здесь сходство не только внешнее, но и функциональное (см. рис. 23, где вместо кнопки **Add Line** использована кнопка вставки массива). Этот прием доказывает, что без кнопки **Add Line** в принципе можно обойтись.

Кнопка ← — это оператор присвоения значения *локальной переменной*. На языке Pascal мы пишем `A:=B+C`, на языке BASIC — `A=B+C`, а на языке Mathcad — `A←B+C`. Почему? Сначала опять же приходится недоумевать, но потом понимаешь, что без знака ← программа превратилась бы в нечто невразумительное, режущее глаз программиста:

- Pascal:

  ```
  A:=A:=B+C
  ```

- BASIC:

  ```
  A=A=B+C
  ```

> ***Примечание***
>
> Первое выражение (Pascal) содержит явную синтаксическую ошибку, второе — нет, т. к. на языке BASIC символ = означает не только присвоение, но и булеву операцию "эквивалентно". В среде языка C программист написал бы
>
> ```
> A = A == B + C
> ```
>
> а на языке Pascal:
>
> ```
> A := A = B + C.
> ```

В Mathcad-выражении:

```
A:=A←B+C
```

все более-менее ясно: локальной переменной `A` (она в середине между символами `:=` и ←) присваивается значение суммы двух переменных `B` и `C`, значение которых уже задано выше в Mathcad-документе (глобальные переменные). Затем эта сумма передается глобальной переменной `A` (она слева от знака `:=`). Вернее, полуглобальной — глобальной она станет, если за ней будет стоять символ ≡.

Благодаря локальным переменным можно создавать объемные Mathcad-документы, поручая разработку отдельных функций и операторов разным программистам и не заботясь о разделении переменных: в разных программах переменные могут совпадать по имени, но при этом они не будут "перебегать друг другу дорогу" (технология программирования "сверху вниз"). С локальными переменными мы, кстати, сталкивались и ранее: примеры индексы `i`, `j` и др. в операторах суммы или произведения.

С другой стороны, можно отметить и некую "вредность", а не полезность института локальных переменных в Mathcad.

Локальность переменной подразумевает ее невидимость вне программы, что с одной стороны не смешивает отдельные программы, а с другой — очень затрудняет процесс отладки программ. Было бы лучше, если бы все переменные программы были

видимы и вне программ, но при желании разработчика становились локальными через механизм стилей переменных Mathcad. Возникает такое чувство, что разработчики языка программирования Mathcad не знали о существовании такого мощного инструмента как стили переменных и не задействовали его. Итак, кнопка оператора ввода локальной переменной лишняя. Тут можно было спокойно обойтись кнопкой **:=** и при необходимости сменой стиля переменной.

Нажав кнопку **while**, мы получим на экране заготовку цикла с предпроверкой — слово `while` с двумя пустыми квадратиками (местозаполнителями) — рис. 5.

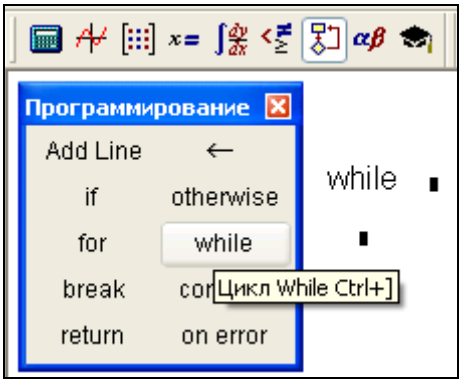

Рис. 5. Заготовка цикла `while`

В первый квадратик (правее `while`) нужно будет записать булево выражение, управляющее циклом, или просто константу, не равную нулю, чтобы получить *бесконечный цикл* (как мы это сделали на рис. 3.2, например), а во второй (ниже `while`) — тело цикла, операторы которого станут выполняться, пока булево выражение возвращает истинное значение (в среде Mathcad — это числовое значение, отличное от нуля). Если в теле цикла более одного оператора (а это основное отличие оператора `while` от вышеупомянутой функции `until` — см. рис. 1), то нужно воспользоваться кнопкой **Add Line** (см. рис. 4).

Цикл, т. е. возможность выполнения какого-либо фрагмента программного кода и, в частности, Mathcad-документа, — это "сердцевина" программирования, т. е. то, из-за чего в основном и приходится прибегать к программированию при решении поставленной задачи. На рис. 6—8 показано решение подобной задачи методом последовательного приближения, подвод ее к циклу `while` и к автоматизации этого вычислительного процесса.

Примечание

Метод последовательного приближения — очень распространенный метод решения инженерных задач — задается какое-то разумное значение искомой переменной, далее это значение расчетом по формулам уточняется. Перечисленные действия (цикл, итерация) повторяются до тех пор, пока (while) два очередных приближения будут выдавать близкие значения искомой переменной. Если такой сходимости не наблюдается, то

этот метод также используют (пытаются использовать), называя его при этом по-другому — методом "научного тыка".

На рис. 6 в нижней его половине переменной μ глобальным присваиванием дается некое значение, которое "закидывается" вверх и участвует в расчете нового значения — переменной μ1. Это значение становится новым приближением (его нужно скопировать — перенести в оператор μ ≡ ■), и так повторяется несколько раз (у нас на рис. 6 три раза), пока не будет достигнута точность расчета (менее 0.1% в нашем случае[5]). На рис. 6.9 и 6.10 читатель может увидеть подобные приближения при решении краевой задачи об эпидемии методом стрельбы (это тоже последовательные приближения: "перелет — недолет — попал").

[5] Решается задача, связанная с процессом обработки природной воды некими реагентами для того, чтобы не откладывалась накипь при ее нагреве в котле. Но мы сейчас обсуждаем не суть задачи, а ее реализацию в Mathcad — метод последовательных приближений.

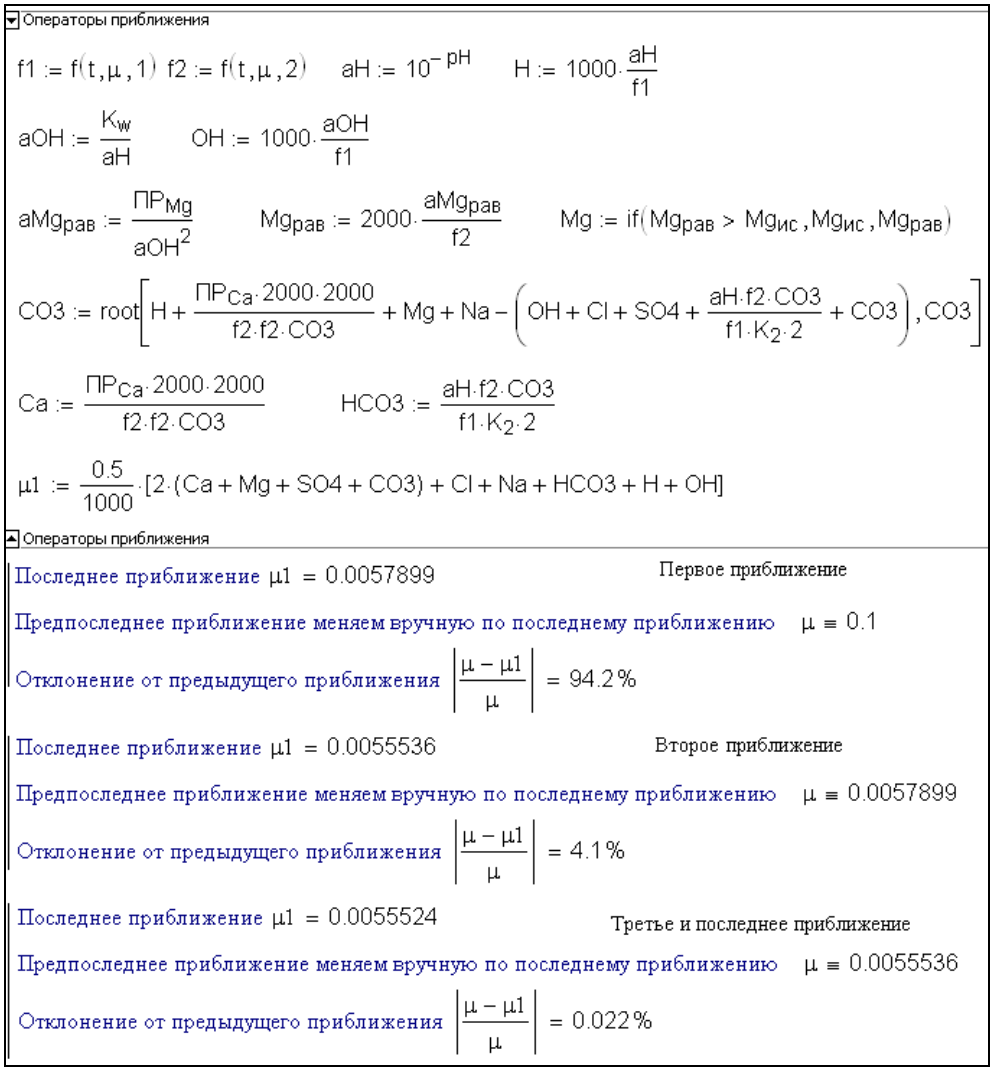

Рис. 6. Метод последовательных приближений — начальный вариант Mathcad-документа

На рис. 7 операторы, участвующие в приближениях, объединены в операторный блок с вводом в расчет *локальных* переменных, все из которых выводятся через векторную запись для контроля за их значениями, что очень важно при *отладке* программ *(см. разд. 7)*.

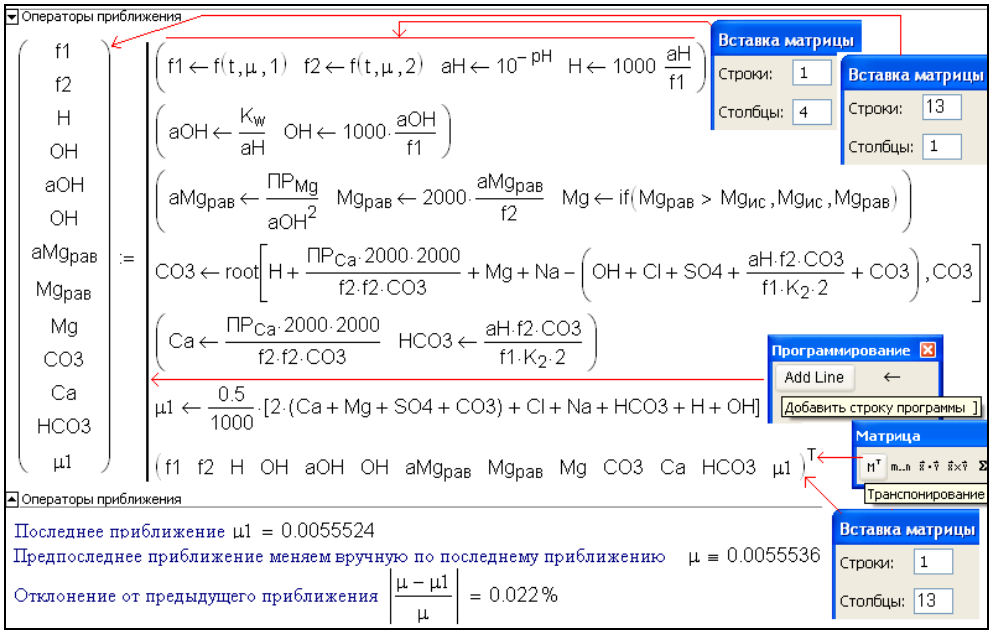

Рис. 7. Метод последовательных приближений — операторы в Mathcad-программе

На рис. 8 показан последний шаг в автоматизации метода последовательных приближений — ввод в расчет оператора `while`, который в паре с операторами `return` и `if` реализует *цикл с выходом из середины*. Это *универсальный цикл* программирования: если оператор `return` переместить в самый конец программы, то получится второй тип цикла — *цикл с постпроверкой* (цикл `repeat...until`, если вспомнить язык Pascal), которого нет среди встроенных инструментов (кнопок) программирования Mathcad (см. рис. 3). Правее оператора `while` на рис. 8 читатель не видит никакого оператора: там записана единица, обеспечивающая бесконечный цикл, цвет которой белый *(см. разд. 1.2.3)*.

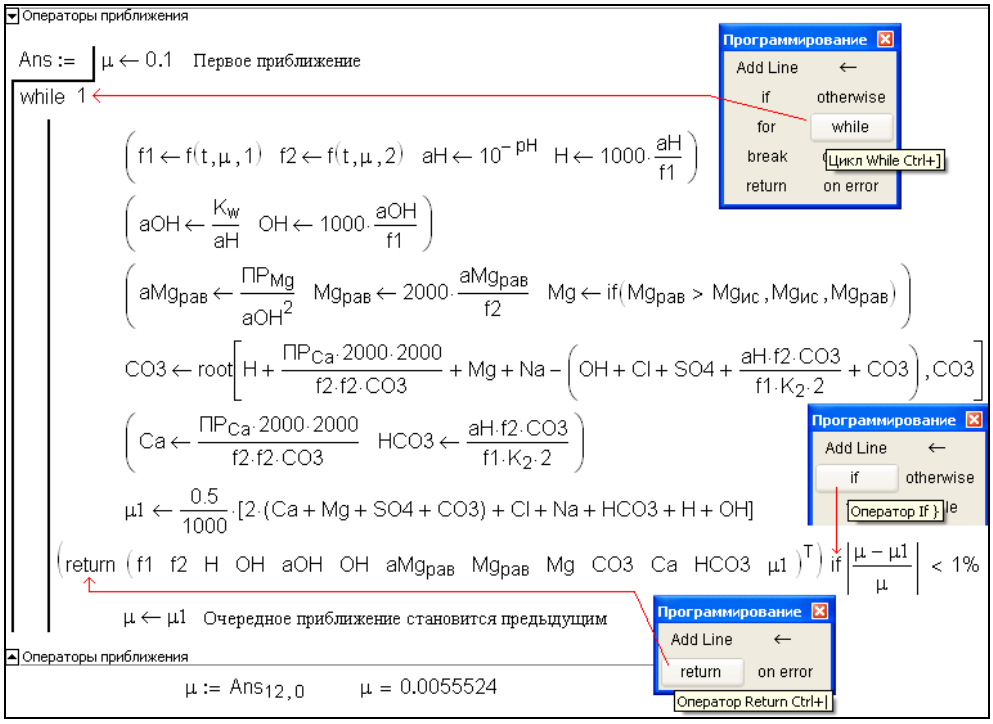

Рис. 8. Метод последовательных приближений — цикл с выходом из середины

Кнопка `if`, которую мы уже задействовали в цикле на рис. 8, позволяет вводить в программу альтернативу с одной ветвью[6]. Так, Pascal-конструкция:

```
if A > B then C := D
```

в среде Mathcad будет выглядеть несколько по-арабски (по-еврейски — записана справа налево):

```
C ← D if A > B
```

Но если ветвь альтернативы — составной оператор, то все почти встанет на свои места, вернее, будет записано уже по-китайски (сверху вниз).

❒ Pascal:

```
if A>B then begin E:=F; F:=G end;
```

Примечание

Ключевые слова `begin` и `end` в среде языка Pascal отмечают начало и конец программных блоков (см. описание кнопки **Add Line**).

[6] Кстати, без программирования альтернативу с одной ветвью реализовать нельзя: функция if всегда должна иметь три аргумента.

❒ Mathcad:

```
if A>B
    E ← F
    F ← G
```

Кнопка **otherwise** превращает неполную альтернативу в полную.

❒ Pascal:

```
if A > B then C := D else E := F;
```

❒ Mathcad:

```
C ← D if A > B
E ← F otherwise
```

Но если в ветвях полной альтернативы по одному оператору, то можно воспользоваться не оператором (кнопкой) `if`, а функцией `if`:

```
C ← if(A > B, D, F)
```

или

```
if(A > B, C←D, E←F)
```

На рис. 9 показан Mathcad-документ с расчетом плотности воздуха по закону идеального газа в зависимости от температуры и давления, значение которых вводятся через текстовые поля с выбором единиц измерения через переключатели, объединенные в группу под названием "радиокнопки"[7]. В случае температуры этот переключатель возвращает числа от 1 до 4 и работает в паре с *функциями* `if`, вложенными друг в друга. В случае же давления (тут возвращаются целые числа от 1 до 5) работают операторы `if` в паре с оператором `otherwise` в конце списка выбора. Второй способ записи альтернативы со множеством ветвей более удобен для анализа (поиска возможных ошибок, например) и расширения, но и первый (с функциями, а не операторами `if`) также имеет право на существование. Дело в том, что некая предпочтительность записи альтернативы через функцию `if` (выбор температурных шкал на рис. 9), а не через оператор `if` вызвана тем, что до недавнего времени считалось, что если в Mathcad-документе удалось избежать использования операторов из панели **Программирование**, то это хорошо. Такой документ можно было открыть и в дешевых версиях Mathcad, не имеющих средства программирования. Еще одно преимущество функции `if` перед оператором `if` — компактность записи в текстовом, а не в графическом ("операторном") формате. Читатель вправе сам выбирать, что вставлять в свои разработки — оператор `if` или функцию `if` (альтернатива с альтернативой, так сказать).

---

[7] Во времена, когда у нас глушили "Голос Америки", "Немецкую волну" и прочие "вражеские" радиостанции, был такой хитрый прием. Если нажать у радиоприемника сразу две кнопки KB I и KB II (короткие волны первого и второго диапазонов), то можно было перейти к диапазону, где "глушилки" не работали. С виртуальными радиокнопками на экране дисплея такой фокус не пройдет.

Плотность воздуха в зависимости от температуры и давления

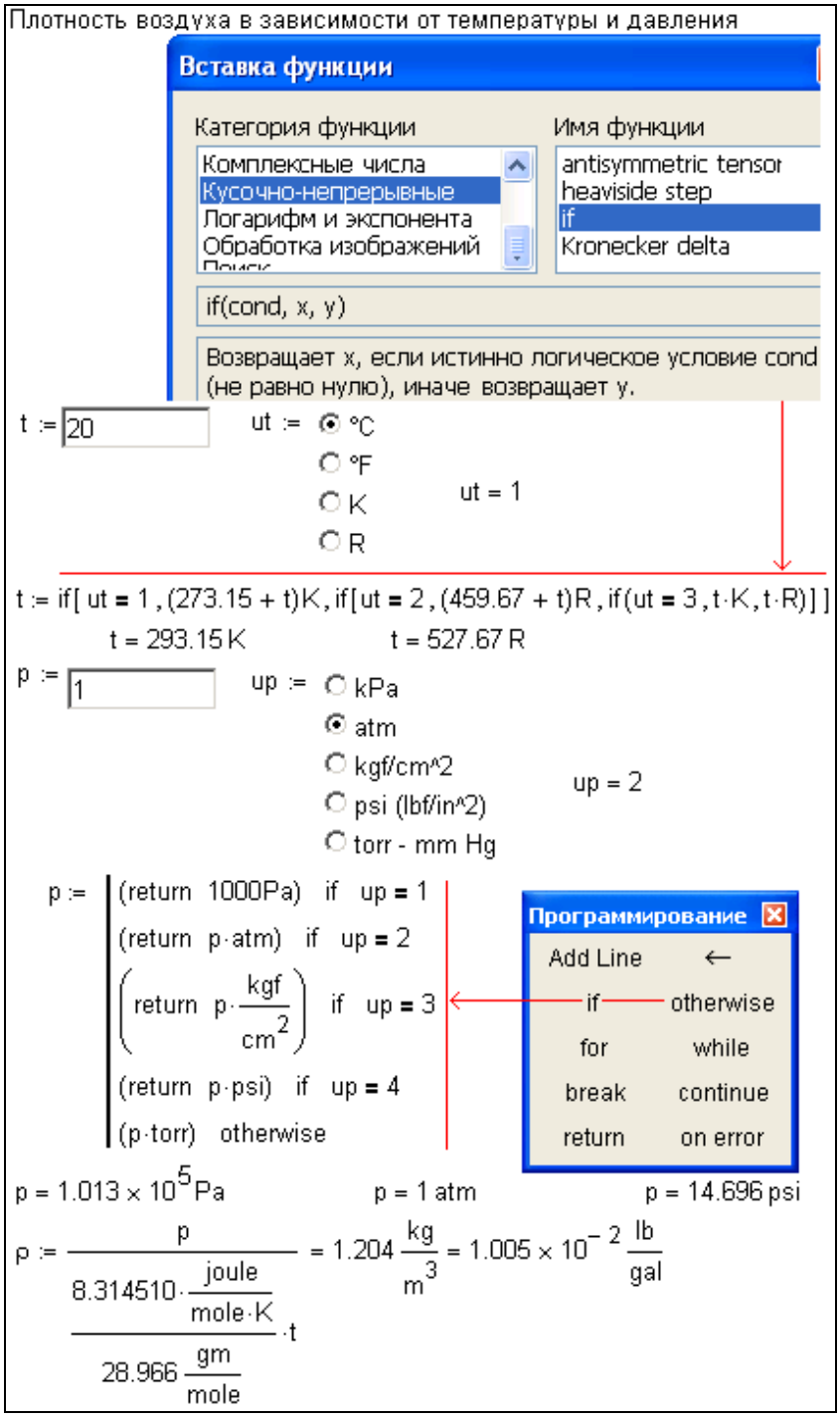

Рис. 9. Альтернатива — реализация через функцию и оператор

Кстати о компактности. В Mathcad-программе по умолчанию можно записать на одной строке только один оператор. Из-за этого любая более-менее объемная программа становится очень длинной и ее можно просматривать только через вертикальную прокрутку экрана дисплея или через изменение масштаба экрана (zoom). Это затрудняет работу с программой. Считается, что если уж не вся программа, но ее отдельные смысловые области (подпрограммы) должны целиком помещаться на экране дисплея стандартного размера. Основным способом повышения компактности программ является размещение на одной строке нескольких операторов.

> ***Примечание***
> Основным способом повышения компактности программ является, конечно, их оптимизация, удаление из программ лишних операторов, переход к "изящным" и, как правило, кратким алгоритмам и т. д.

В среде языка программирования Mathcad[8] нет встроенных средств создания многооператорных программных строк. Но их можно ввести в программу недокументированным способом и мы этим уже неоднократно пользовались. Так на рис. 8, например, да и на многих других рисунках книги отображено как несколько операторов вводятся на одну программную строку в виде матрицы с одной строкой и несколькими рядами ("горизонтальный вектор", вектор-строка, элементы (компоненты) которого — отдельные операторы программы). Здесь может быть только одно ограничение — операторы, собранные в матрице, должны возвращать либо безразмерные величины, либо величины одной размерности (длины, массы, силы и т. д.). Это связано с тем, что массивы (векторы и матрицы) Mathcad не могут хранить разноразмерные величины, и мы об этом уже говорили в *разд. 1.4*.

Но и тут есть выход из положения. Операторы, возвращающие разноразмерные величины, можно разместить на одной программной строке в виде сомножителей произведения, сделав сам знак умножения невидимым. В такое "произведение" можно дополнительно вставить невидимые единицы-сомножители для того, чтобы отодвинуть операторы от левого края[9] или вставить между ними дополнительные пробелы. Но это будет уж слишком сильной экзотикой. И опять же, такая строка-произведение[10] застопорится, если один из операторов будет возвращать не числовое, а текстовое значение. Одним словом, несколько операторов на одной стороне — это недокументированный прием, чреватый сбоями. И если читатель видит такую строку в Mathcad-программе этой или другой книги, то он должен считать такую программу заархиви-

---

[8] В традиционных языках программирования сначала нельзя было писать несколько операторов на одной строке. Затем по мере развития языков такая возможность появилась.

[9] Так повышают наглядность программы: чем правее расположен оператор, тем он глубже сидит в структуре программы.

[10] Ее автор использовал в своих Mathcad-программах, пока не додумался заменить их на векторы-строки.

рованной (сжатой). При вводе такой программы в компьютер "с листа" ее следует разархивировать — оставить только по одному оператору на одной строке.

Кнопку **Add Line** следовало бы назвать **Add Operator**, т. к. через нее в программу вводится не строка (операторов), а только один оператор.

Еще один недокументированный прием — *комментарии* в Mathcad-программах.

Язык программирования Mathcad не имеет специальных средств (операторов) для этих целей. В Mathcad 12, 13 и 14, как уже отмечалось ранее, появились средства скрытого комментирования отдельных операторов, в том числе и программ (см. рис. 1.62). Но эти комментарии неудобны тем, что их нельзя увидеть все сразу, а только поочередно после подвода курсора к оператору и вызова соответствующего диалогового окна. Кроме того, такие комментарии не видны на распечатках Mathcad-документов. Те или иные операторы программы можно прокомментировать, поместив правее программы текстовые вставки. Но из-за возможных различий в шрифтах формул и текстовых вставок трудно бывает поставить комментарий именно там где нужно, т. е. точно у комментируемого оператора. Кроме того, такую россыпь операторов трудно двигать (вырезать и вставлять в другом месте) — что-то можно потерять по дороге.

Но комментарии можно вставлять в программы в виде отдельных операторов — текстовых констант. Можно поступать и так — записывать в программу вектор-строку с двумя элементами, первый из которых — комментарий, а второй — оператор программы, или наоборот. Недостаток тут известный — невозможность проверки орфографии таких констант, даже если они написаны по-английски. Здесь можно посоветовать поступать следующим образом: записывать комментарии, например, в Word, вылавливать возможные ошибки с помощью этого текстового процессора, а потом переносить текстовые константы в Mathcad.

> ***Примечание***
> Для того чтобы можно было прописывать комментарии (текстовые константы) русским шрифтом (кириллицей), необходимо стилю Constants поставить в соответствие шрифт, работающий и с русскими буквами.

Но вернемся к альтернативе.

Альтернативу можно считать вспомогательной структурной, управляющей конструкцией — оператором, без которого можно обойтись.

Так на рис. 10 показано, как в программе поиска нуля функции методом половинного деления альтернатива заменяется на два цикла `while`, операторы которых попеременно выполняются либо раз, либо ни разу.

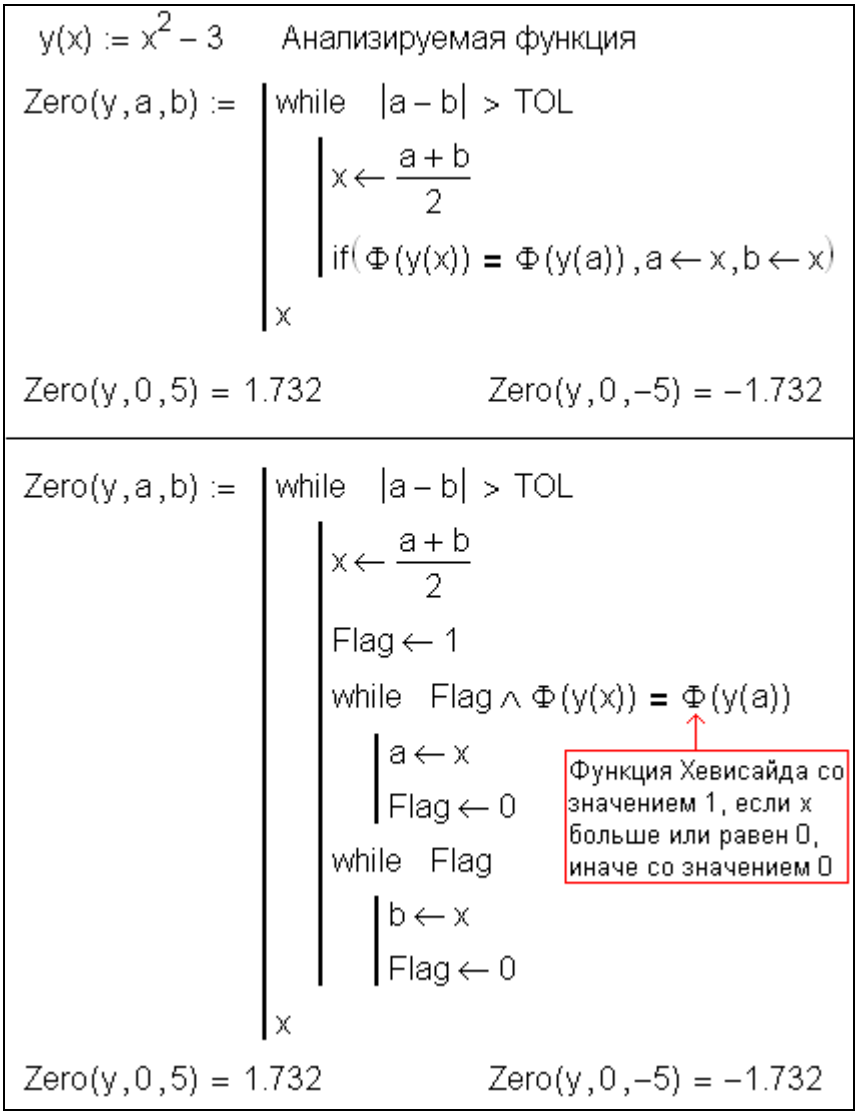

Рис. 10. Два оператора `while` вместо одного оператора `if`

Программу, показанную на рис. 10, можно считать неким курьезом, программистской шуткой. А можно считать подтверждением того факта, что основная структурная теорема, гласящая, что алгоритм любой сложности можно реализовать через триаду "следование-повторение-выбор", не совсем верна — оператор `if` оказывается лишним. Дело в том, что...

Очень часто при решении задачи необходимо пускать расчеты по разным путям в зависимости от исходных данных. И не просто пускать, а так пускать, чтобы пользо-

ватель готовой расчетной методики видел не все эти пути (группы операторов), а только тот, по которому ведется расчет в настоящее время. На рис. 11 показано одно из решений этой проблемы.

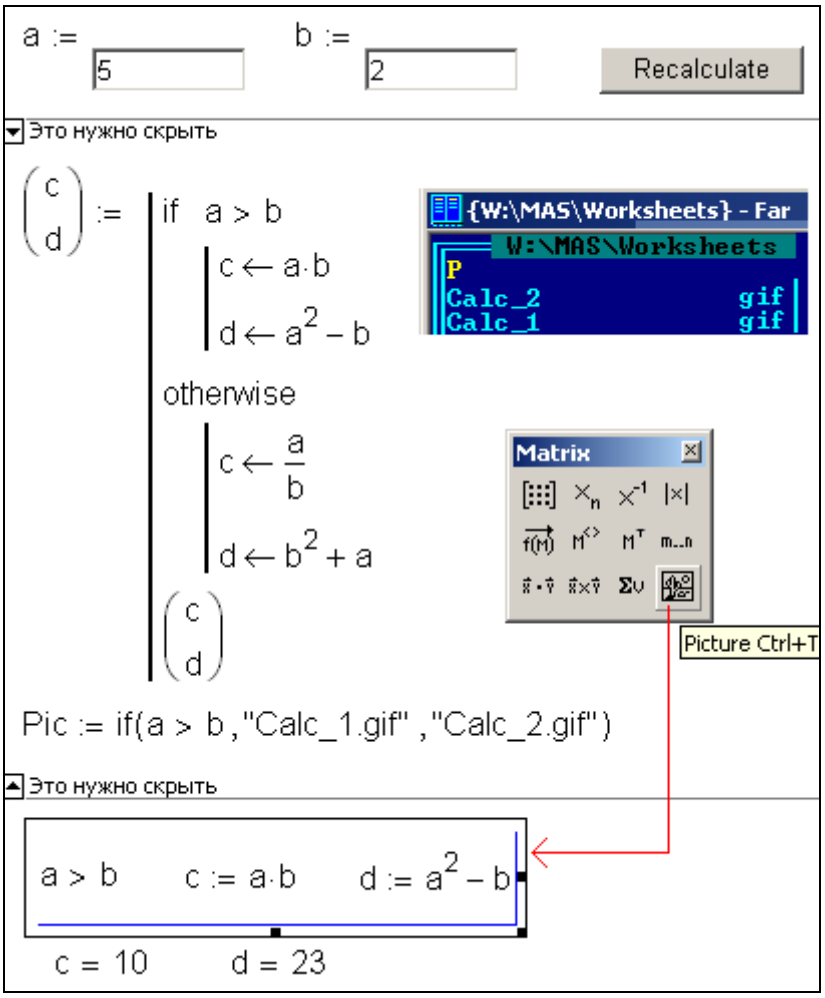

Рис. 11. "Визуальная" альтернатива в Mathcad-документе

На рис. 11 показан ход решения простенькой "альтернативно-визуальной" задачи: переменные `c` и `d` равны `a·b` и a^2-b, соответственно, при `a>b`, но `a/b` и b^2+a — в противном случае. При этом после ввода значений `a` и `b` пользователь видит только те операторы, по которым в данный момент ведется расчет. Решение тут такое — пары операторов, по которым ведутся расчеты, вернее, их изображения вместе с их условием (`a>b` или `a≤b`) записываются на диск в виде двух графических файлов Calc_1.gif и Calc_2.gif (на рис. 11 изображен фрагмент файлового менеджера с этими

фалами), которые попеременно вставляются в расчет. На рис. 12 показано, как такой расчет будет представлен на экране дисплея при разных значениях переменных `a` и `b`.

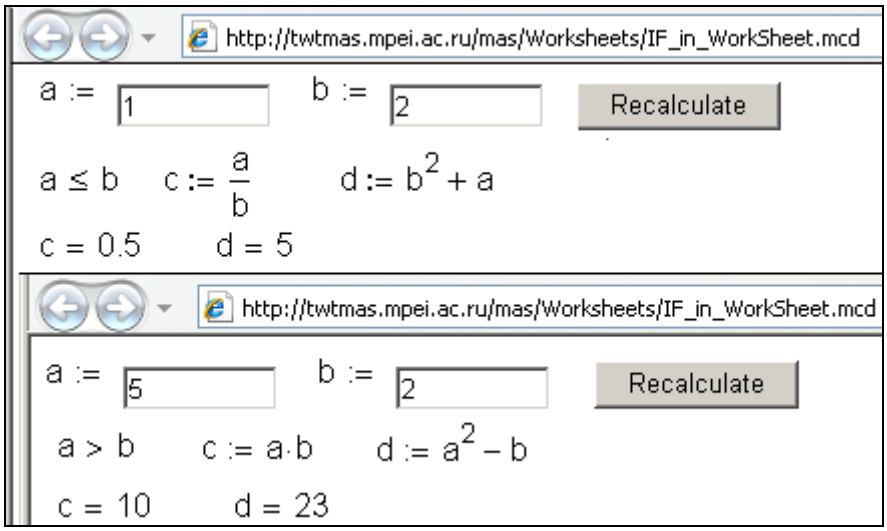

Рис. 12. "Визуальная" альтернатива в Интернете (MCS)

Конечно, расчет, показанный на рис. 11, не предназначен для редактирования и расширения (WorkSheet), а только для конечного использования в Интернете, например (WebSheet).

В связи с этим (проведение всяческих "закулисных" действий с расчетом) в WorkSheet можно сравнить с *театром*, а WebSheet — с кинематографом, иллюзионом, где возможно то, что невозможно или "шито белыми нитками" в театре — в расчетных документах, открытых в среде Mathcad (WorkSheets), а не в Интернете (WebSheets).

Оператор `if` в сочетании с операторами `return` и `otherwise` — довольно гибкая конструкция, позволяющая легко программировать в среде Mathcad разного рода альтернативы с вкрапленными в них вычислительными операторами. Один из примеров — на рис. 13, где показана программа анализа квадратного уравнения вида $A \cdot x^2 + B \cdot x + C = 0$.

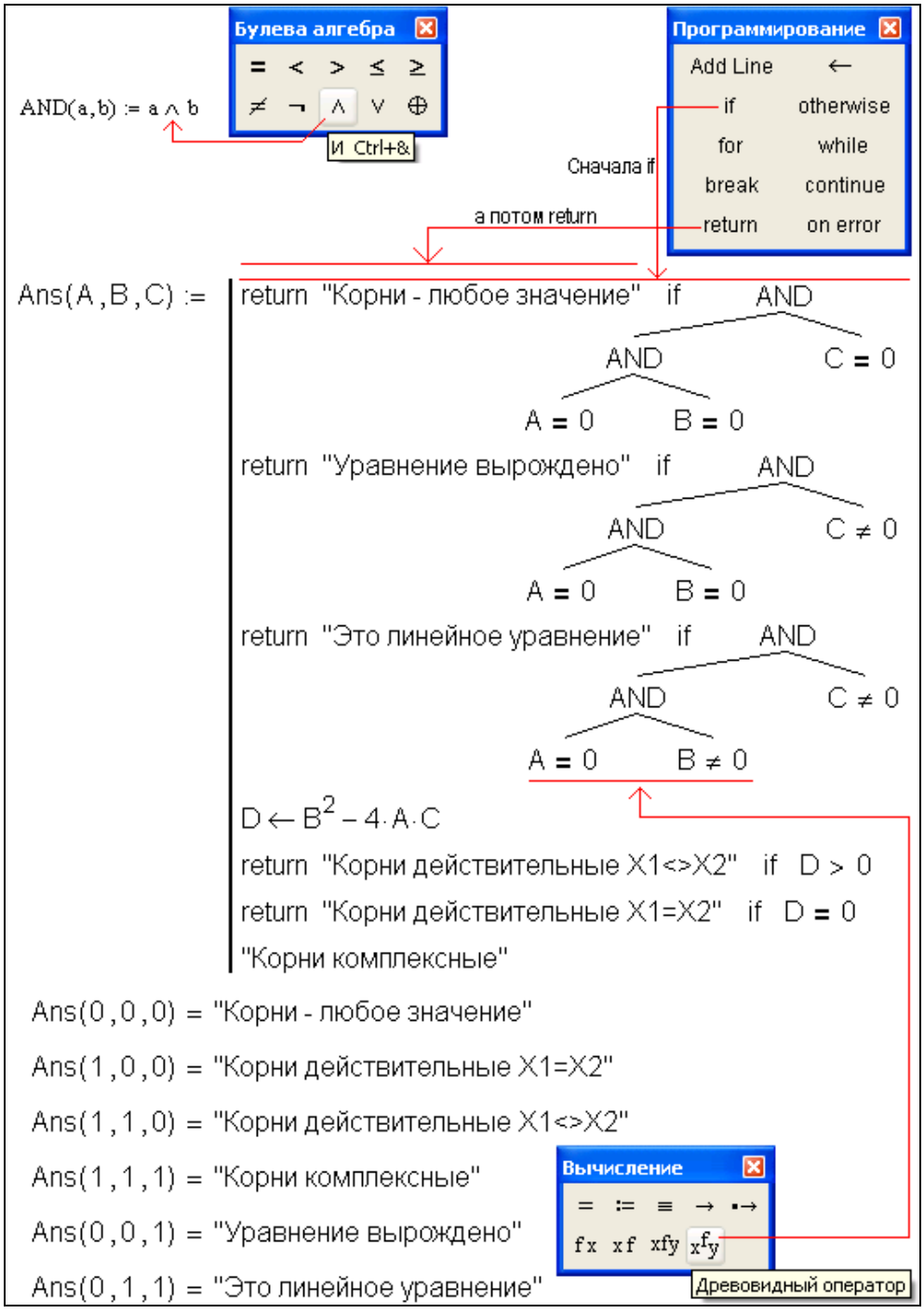
Булева алгебра
= < > ≤ ≥
≠ ¬ ∧ ∨ ⊕
И Ctrl+&
AND(a,b) := a ∧ b
Программирование
Add Line ←
if otherwise
for while
break continue
return on error
Сначала if
а потом return
Ans(A,B,C) :=
return "Корни - любое значение" if AND
AND C = 0
A = 0 B = 0
return "Уравнение вырождено" if AND
AND C ≠ 0
A = 0 B = 0
return "Это линейное уравнение" if AND
AND C ≠ 0
A = 0 B ≠ 0
D ← B^2 – 4·A·C
return "Корни действительные X1<>X2" if D > 0
return "Корни действительные X1=X2" if D = 0
"Корни комплексные"
Ans(0,0,0) = "Корни - любое значение"
Ans(1,0,0) = "Корни действительные X1=X2"
Ans(1,1,0) = "Корни действительные X1<>X2"
Ans(1,1,1) = "Корни комплексные"
Ans(0,0,1) = "Уравнение вырождено"
Ans(0,1,1) = "Это линейное уравнение"
Вычисление
= := ≡ → ▪→
fx xf xfy xfy
Древовидный оператор

Рис. 13. Конструкция "выбор"

В программе рис. 13 можно было обойтись без оператора `return`, что вернуло бы данную программу в "правовое поле" структурного программирования с главным "законом" — один вход, один выход. Но оператор `return` ускоряет расчет, отсекая ненужные действия.

Кнопка **for** вводит в программы *цикл с параметром*.

Когда заранее известно, сколько раз нужно выполнить какую-то часть программы (тело цикла), то используют не цикл `while`, а цикл `for`, в заголовке которого пишут не булево выражение, а *параметр цикла*, и указывают, какие значения он должен принимать при каждом выполнении цикла.

Цикл с параметром в среде Mathcad намного более гибок, чем его аналоги в языках программирования BASIC, Pascal или C (см. примеры на рис. 14).

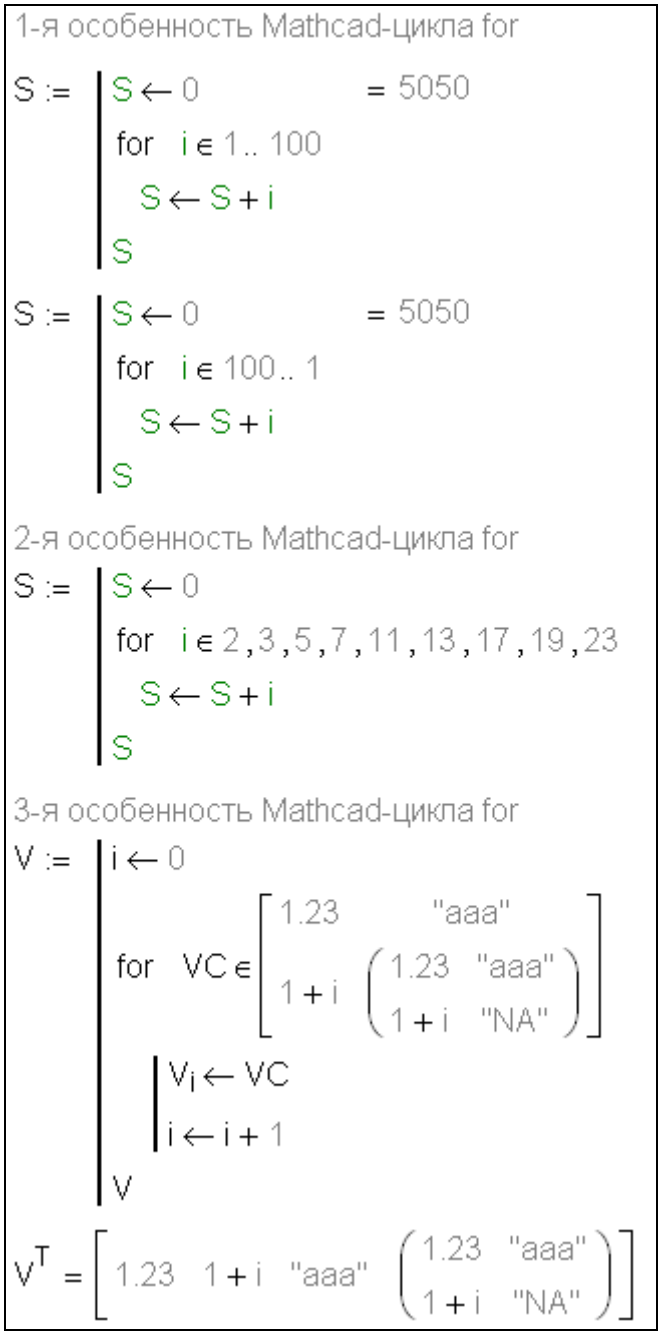

Рис. 14. Три особенности цикла `for`

Первая особенность ("гибкость") цикла `for` в среде Mathcad заключается в том, то ему не нужно указывать направление шага изменения параметра в случае, когда первое значение параметра меньше второго. В такой ситуации Mathcad-цикл `for` сам разберется, в какую сторону вести отчет. В средах языков BASIC или Pascal отрицательный отсчет шага в цикле `for` требует дополнительных ключевых слов `Step-1` или `downdo`, соответственно.

Вторая особенность (опять же — "гибкость") цикла `for` в среде Mathcad — это возможность работы со списком своих параметров вместо переменной области (Range Variable) или вектора, который обычно пишут на этом месте.

Третья и главная особенность Mathcad-цикла `for` в том, что переменная цикла может принимать значение различных *типов* — вещественное число, комплексное число, текст, массив и т. д. Кроме того, значения параметра цикла могут выбираться и из матрицы (упакованного вектора), а не только из переменной области, вектора, или списка *(см. ранее)*.

Кнопки **break** и **continue** позволяют досрочно выходить из циклов `while` и `for`, а кнопка **return** — совсем из программы, что мы уже проиллюстрировали (см. рис. 13). О кнопках **break** и **continue** разговор особый. Сейчас же проведем такую аналогию.

Все структурные управляющие конструкции Mathcad можно усмотреть в простой житейской ситуации: потчевание гостей чаем и кофе. Хозяйка проверяет, нет ли на столе пустой чашки (булева переменная, управляющая циклом `while`), и наполняет ее (тело цикла) чаем или кофе (альтернатива). Добавление в чашку кусочков сахара — новый, вложенный цикл. При разливе чая чашка (стакан) может лопнуть, что прерывает цикл, из которого выходят в конец цикла (`break` — гости встают из-за стола и занимаются чем-то другим) или в начало цикла (`continue` — на столе меняется скатерть, а чаепитие возобновляется). Третий сценарий: разбитая чашка так расстраивает хозяйку, что вечеринка досрочно заканчивается (`return`). Хорошая хозяйка умеет переключить разговор, если он затронет тему, неприятную одному из гостей. Умеет она незаметно сглаживать и другие нештатные ситуации. Здесь очень пригодится оператор `on error`, обрабатывающий ошибки. Он, по сути, является альтернативой с одной ветвью и завуалированной (запрятанной) булевой переменной, управляющей альтернативой. Мы использовали оператор `on error`, когда рисовали "картину" поиска корней уравнений (см. рис. 3.29).

Язык программирования Mathcad по своей идеологии очень похож на язык FRED интегрированного пакета Framework[11]. Говорят, что один из "погорельцев" фирмы Ashton-Tate (разработчика Framework) перешел в фирму Mathsoft[12] и приложил руку к созданию языка программирования Mathcad. Внешне же своими вертикальными линиями, фиксирующими операторные блоки, пакет Mathcad напоминает алгоритмические конструкции из книги А. П. Брудно "Программирование в содержательных

[11] Следом появится интегрированный пакет Works, а уж потом Office, интегрирующий текстовый и табличный процессоры (Word и Excel), а также базы данных (Accses).
[12] Фирма Mathsoft сама в 2006 году перешла под крыло фирмы PTC.

обозначениях"[13]. В свое время автор очень увлекался подобными линиями, втискивая программы в рамки структурных диаграмм[14]. Вертикальные линии программ Mathcad более наглядны (особенно для обучения структурному программированию), чем просто операторные скобки (`begin...end` на языке Pascal, фигурные скобки языка C, оператор `list()` языка FRED, конец строки BASIC-программы, круглые скобки математических выражений и т. д.).

Говоря о структурном программировании, нельзя не отметить тот факт, что разработчики языка Mathcad сразу отказались от *метки* и операторов *условного* и *безусловного перехода* к метке как инструмента реализации разветвленных алгоритмов. Заодно был игнорирован и цикл с постпроверкой. Для некоторого смягчения этой категоричной позиции и были введены операторы `return`, `break` и `continue`, двумя последними из которых автор не рекомендует пользоваться, т. к. они сильно путают программиста.

Рассмотрим еще несколько особенностей Mathcad-программ на занимательных задачах.

3. Турецкий платок

Очень часто, решая ту или иную проблему, мы оказываемся в шкуре буриданова осла. Задача может иметь два альтернативных решения, как две охапки сена слева и справа от упомянутого животного. Все доводы "за" и "против" уравновешены. Как в этом случае поступить? Ехать или не ехать в командировку? Покупать или не покупать еще один винчестер? Поистине, гамлетовские вопросы задает нам жизнь!

Некоторые в таких ситуациях бросают монетку, другие загадывают мужчину или женщину и смотрят в окно, ожидая, кто первый появится. Но все это ненаучные методы. Монетка может куда-нибудь закатиться, а по улице как назло за целый час кошка только и пробежит...

Есть проверенный столетиями метод принятия подобных решений. Достаточно разложить пасьянс. Сошелся — решение принято и все сомнения прочь. Можно подыскать дополнительные доводы в его пользу, и начать воплощать в жизнь. Пасьянс психологически нас на это настраивает.

Но принять решение подобным образом иногда бывает трудно, т. к. не всегда под рукой есть колода карт, да и не совсем удобно раскладывать их на рабочем месте. Но это можно сделать и на экране дисплея. В среде Windows, например, есть игры-пасьянсы ("Солитер" и "Косынка"), но мы придумаем что-нибудь новенькое, а главное, более занимательное и поучительное — разложим в среде Mathcad старинный пасьянс "Турецкий платок". На это есть три причины:

---

[13] Брудно А. П. Программирование в содержательных обозначениях. — М.: Наука, 1968.

[14] Очков В. Ф. 128 советов начинающему программисту. — М.: Энергоатомиздат, 1991 (см. **http://twt.mpei.ac.ru/ochkov/128/index.htm**).

1. Чтобы лучше освоить программную среду, нужно постараться решить в ней задачу, для этого крайне неподходящую (своего рода программистское извращение[15]— деривация).
2. В детстве каждый нормальный человек, наигравшись, ломал игрушку, чтобы посмотреть, как она устроена. Посмотрим и мы, как тасуется колода и раскладывается пасьянс.
3. По традиции гадать на картах и раскладывать пасьянсы разрешается только в святки[16]. Наш же пасьянс можно считать просто числовой головоломкой, которую позволительно решать круглый год.

Mathcad-документ (рис. 15) позволяет разложить пасьянс "Турецкий платок" по следующим правилам. Из одной перетасованной колоды в 52 листа выкладывают картинкой вверх пять рядов по 10 карт в каждом. Последние две карты кладут в шестой неполный ряд на любое место, как правило, к первому и второму столбцам[17]. Требуется распустить этот "платок", снимая из разных столбцов за один ход по две нижние одинаковые карты — тройки, дамы, тузы и т. д.

[15] В первых трех изданиях этой книги гл. 3 так и называлась — "Взгляд на Mathcad шутника, дериватора и эстета".

[16] В период между Рождеством и Крещением. Программа на рис. 15 написана автором (вернее, подправлена для Mathcad 8) 11 января 1998 г. Так что здесь все в порядке.

[17] У нас они будут нулевым и первым — мы не будем трогать переменную `ORIGIN`.

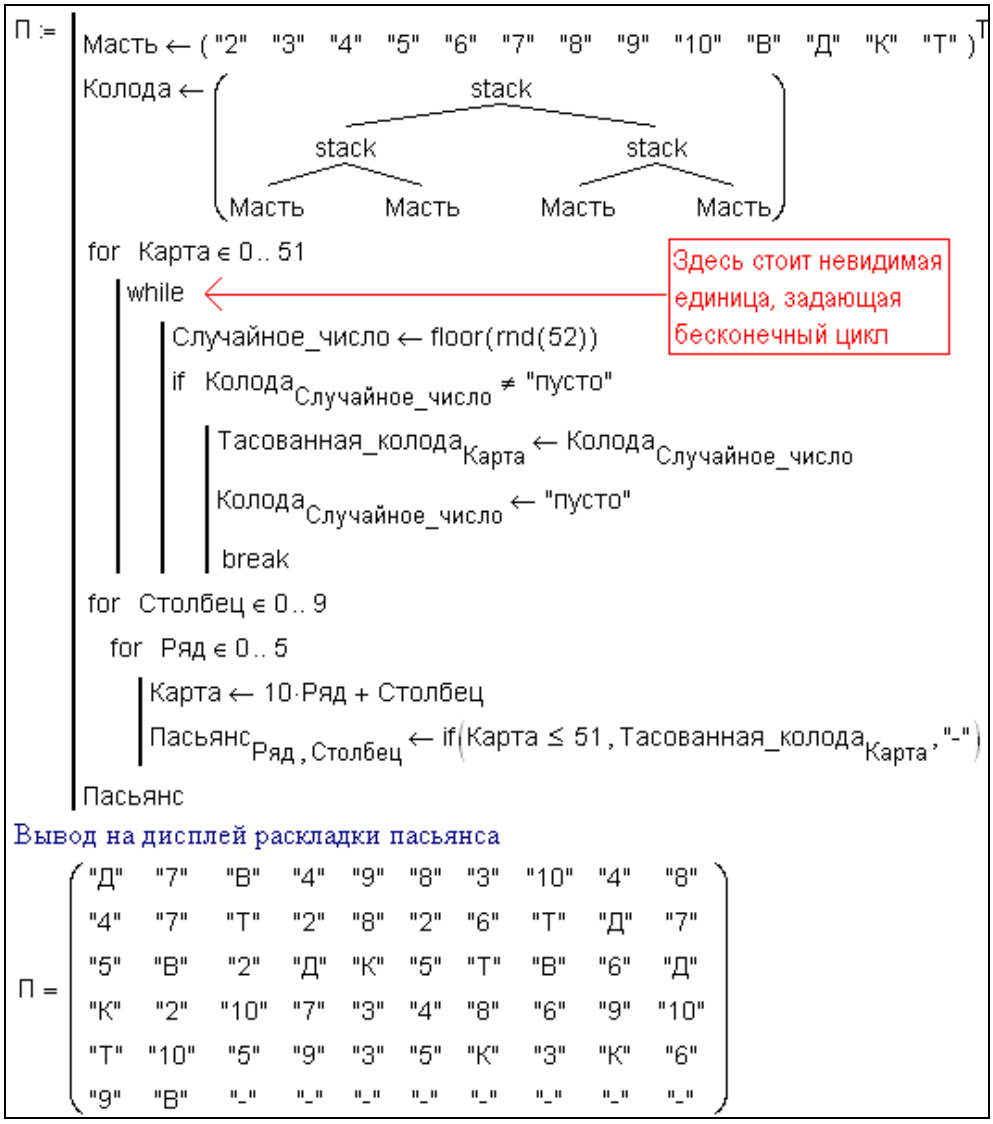

Рис. 15. Программа пасьянса "Турецкий платок"

Для снятия карт нужно скопировать правую часть оператора `П=...` (саму матрицу) на свободное место, подвести курсор к нужной текстовой константе (к карте), щелкнуть левой кнопкой мыши и нажать клавишу <Delete>. Вместо числа появится пустой квадратик.

Из разложенного пасьянса можно снять две девятки, две тройки, две пятерки или двух королей. Если удастся снять все 52 карты, то это означает, что в командировку все-таки ехать придется, а винчестер покупать надо.

Составление программы, формирующей матрицу П (раскладка пасьянса) — прекрасное и, что не менее важно, занимательное средство изучения таких базовых понятий линейной алгебры, как *вектор* и *матрица*[18]. Так, при формировании матрицы П транспонируется матрица Масть (она ставится "на попá" — матрица с одной строкой превращается в матрицу с одним столбцом, т. е. в вектор). Далее с помощью функции stack формируется новая нерастасованная колода карт, где одна отсортированная масть идет за другой (вектор Колода). Затем в цикле с параметром (for...) с помощью цикла while[19] и функции rnd идет формирование перетасованной колоды — вектора Тасованная_колода, который в конце программы двойным циклом с двумя параметрами (for... for...) складывается слоями в матрицу П.

Программу, формирующую матрицу П, можно развить: заставить программу отбраковывать явно нерешаемые раскладки — такие, например, где в одном столбце оказались три или даже четыре одинаковые карты. Еще одна тупиковая ситуация — две пары карт крест накрест закрывают друг друга. Это будет хорошим упражнением, закрепляющим навыки работы с "матричными" операторами и функциями в среде Mathcad. А вот более сложное задание читателю: доработать программу так, чтобы она сама раскладывала пасьянс, либо на худой конец сообщала, что его можно решить, просто снимая снизу первые подвернувшиеся одинаковые открытые карты. Более умная стратегия подразумевает выбор карты из трех или четырех одинаковых открытых карт.

Кроме линейной алгебры, наша программа затрагивает и другие интересные разделы математики — теорию вероятностей, статистику (см. функцию rnd, генерирующую псевдослучайные числа). Интересный вопрос: можно ли составить программу, высчитывающую вероятность сходимости того или иного пасьянса? Считается, что пасьянс "Солитер", входящий в стандартную поставку Windows, раскладывается при любых начальных раскладках. Но это только гипотеза...

На рис. 16 программа раскладки пасьянса переделана так, чтобы выпадали совершенно разные раскладки при открытии файла. Для этого используется функция time, возвращающая время в секундах, пошедшее с 1970 г. От этого значения отрезается дробная часть (генератор не псевдо-, а истинослучайных чисел), которая множится на 1000. Полученное целое число становится числом выполнения цикла for, в теле которого истинослучайное количество раз вызывается функция rnd. Этим и достигается случайная и неповторяющаяся раскладка пасьянса за счет тщательной тасовки колоды.

[18] Кстати, в Mathcad 13 и 14, как заверяют разработчики, существенно улучшены инструменты работы с линейной алгеброй, базовыми понятиями (объектами) которой и являются вектора и маршруты.

[19] Здесь более уместен цикл с постпроверкой, но его нет в среде Mathcad. Как в этом случае поступить, будет рассказано далее.

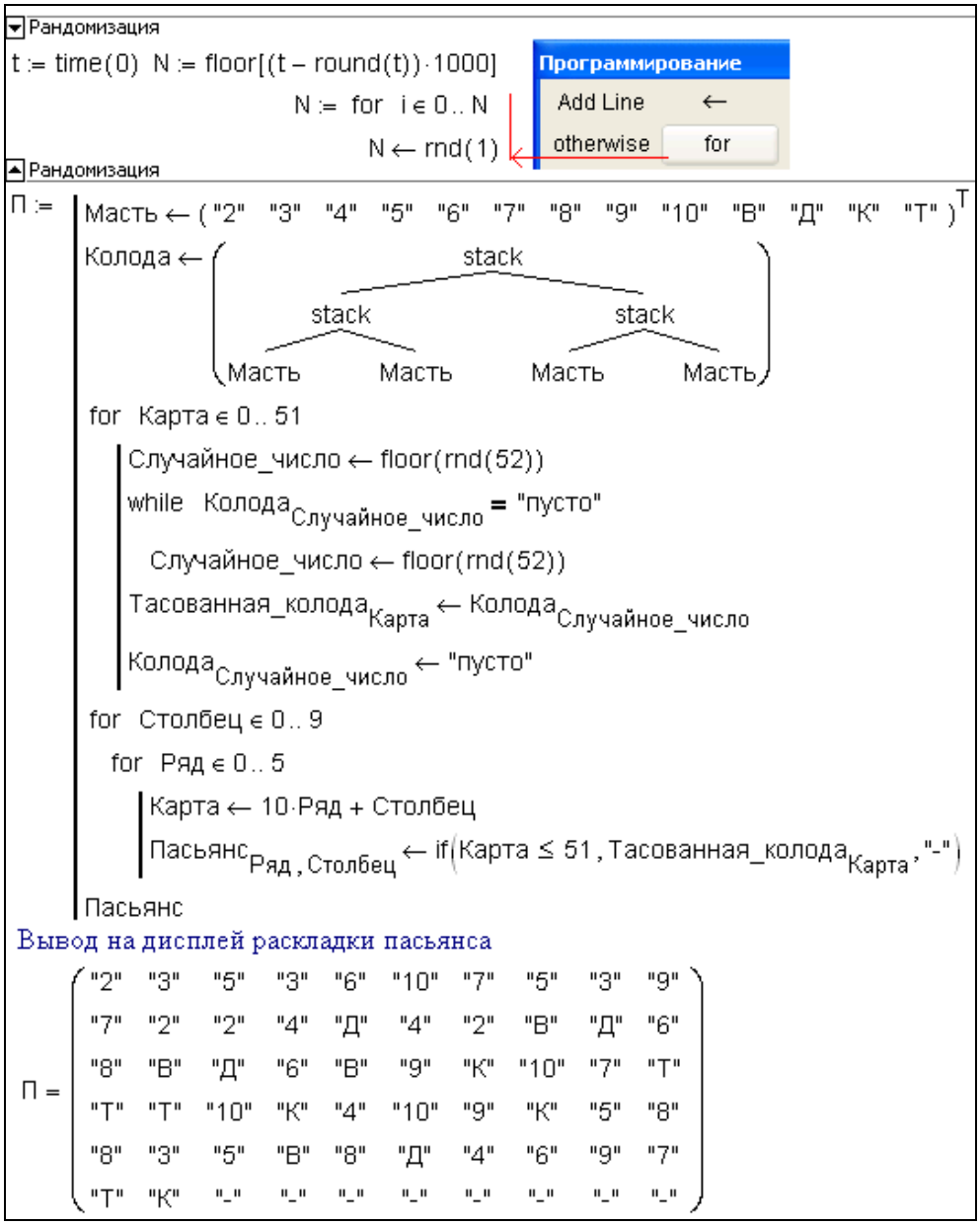

Рис. 16. Программа пасьянса "Турецкий платок" — рандомизация

Кроме того, изменен вид реализации цикла с постпроверкой (а его, повторяем, нет в Mathcad). На рис. 15 он ведется через бесконечный цикл и оператор `break`, а на

рис. 16 через "конечный" цикл `while` и дублирование оператора `Случайное_число ← floor(rnd(52))`.

Часто бывает так, что человек прекрасно делает то, чему он учился *играючи* и с большим удовольствием: ходит, говорит, плавает, катается на велосипеде и т. д., и т. п. Автор, например, задолго до школы научился считать, играя в карты: валет — двойка, дама — тройка и т. д. И сейчас, обращаясь к своим студентам, не устает повторять, что главное, что нужно получать от учебы, — это не *знания*, не *практические навыки*, а... *удовольствие*. Не получая радости от учебы, от повседневного труда, человек тратит свою жизнь впустую...

Начинать изучение векторов и матриц (массивов данных) в курсе программирования (информатики) можно со знакомства со встроенными функциями и операторами конкретной программной среды. А можно раскладывать пасьянсы.

Как уже отмечалось ранее, переменные в Mathcad могут быть *локальными*, самообъявляющимися в программах (как, например, переменные `Случайное_число`, `Колода`, `Карта`, `Столбец` и `Ряд` в программе на рис. 15 и 16). Значение локальных переменных пропадает по выходу из программы. Разработчики языка Mathcad посчитали лишним обязательное объявление переменных до их использования (как это делается при работе с языком Pascal, например). Наверное, переменные не объявляются из-за того, что они все однотипные[20]. Но необъявление переменных может приводить к ошибкам, которые трудно выявить. Ввел программист в программу переменную `day`, а через пару операторов написал `dey` (что по произношению более соответствует английскому слову day (день); можно умудриться написать и `day`, где вторая буква будет из русского алфавита) — программа выдает неверный ответ. Кроме того, переменная `day` уже хранит встроенную единицу времени, что может также быть причиной ошибки. Опечатки в программе часто бывают намного страшней в плане отладки, чем ошибки алгоритма. Кроме локальных и *глобальных* переменных, значения которых заданы вне программы и автоматически в нее проникают, в среде Mathcad есть и *системные (предопределенные)* переменные и константы. Пример — числа `e` и π, значение которых определено самой системой (математикой), а не пользователем.

Мы уже не первый раз используем буквы *кириллицы* в именах переменных и функций. В программе на рис. 15 *все* переменные прописаны по-русски. *Pro* и *Contra* этого приема.

❒ Pro.

Полные имена переменных, написанные на родном языке программиста (`Столбец, Ряд` вместо `i, j`) делают программу более простой для понимания, но более сложной для написания (рекомендуется длинные переменные писать один раз, а потом копировать в нужных местах). Конечно, можно было написать и английские термины в качестве имен переменных, но они будут выглядеть чужеродными

[20] До Mathcad 7 Pro они имели вещественный тип и при необходимости комплексный. В версиях выше 7-й тип переменных — `Variant` (помесь числа с текстом), если исходить из стандарта языка Visual Basic и технологии OLE.

в пасьянсе, программу которого для русскоязычного читателя написал человек, считающий себя русским. Да и автор, честно говоря, не знает, как будет по-английски "Масть", "Колода". Лезть же в словарь не с руки. Проще написать `Mast`, `Koloda`. Проще, да некрасиво *(см. далее)*.

❒ Contra.

- Русские имена переменных порождают "смешенье языков французского с нижегородским[21]": `for Столбец`?! Правильнее и грамотней писать `for Солб-ца`[22] (`для Столбца`); на многих программистов русское имя объекта программирования (файла, переменной, функции и т. д.) действует как красная тряпка на быка[23].
- В латинском и русском алфавитах многие буквы (например, `a`, `c`) совпадают по написанию. Из-за этого в программе могут оказаться переменные, одинаковые для человека, но разные для Mathcad.
- Выбор для переменных шрифта с окончанием Cyr приводит к перекодировке символов. Из-за этого, например, может пропасть восклицательный знак в факториале и др.

4. Рекурсия

Рекурсивная функция — это такая функция, которая вызывает сама себя в момент ее создания[24]. Рекурсия — очень мощный и удобный инструмент решения задач. Так, например, один из самых быстрых алгоритмов сортировки массива использует рекурсию. Вычислить определитель квадратной матрицы также помогает рекурсия и т. д.

Языки программирования в своем развитии обычно проходят три стадии:

❒ рекурсия невозможна;

❒ рекурсия не разрешена, но применяется по принципу: "Если нельзя, но очень хочется, то можно". Так, на старых версиях языка BASIC рекурсия реализовывалась через оператор `ON N GOTO`, передающий управление программой на `N`-ю строку;

❒ рекурсия разрешена.

Mathcad вторую стадию проскочил.

---

[21] Такое "смешенье языков" имеет место в меню, когда в английский Mathcad внедряют русский Ворд, пардон, Word или Excel.

[22] Такое было возможно в ранних версиях языка BASIC, где допускались длинные переменные, но они идентифицировались только по двум первым символам.

[23] Некоторые полагают, что лет через 10—20 мы перейдем к написанию русских текстов латиницей: sejchas tak pishut teksty mejdunarodnych telegram. Молдавия, Азербайджан и некоторые другие страны СНГ уже перешли от кириллицы к латинице. Основная причина в сфере информатики. Вспомним, какие абракадабры нередко приходят к нам по e-mail.

[24] У алхимиков есть символ: "Змея, глотающая свой хвост", который подходит и для рекурсии — маленькая программа, заглатывающая память компьютера.

И вот уже трещат морозы
И серебрятся средь полей...
(Читатель ждет уж рифмы *розы*;
На вот, возьми ее скорей!)

Читатель ждет уж примеров рекурсии в среде Mathcad? Скорее всего, нет. Традиционные примеры ("розы-морозы") ему оскомину набили. Но мы попробуем что-нибудь свеженькое. Например, *двустороннюю (ретроспективную)* рекурсию или, если так можно выразиться, *самораскрывающуюся* рекурсию.

Как запомнить число `e` (основание натурального логарифма) с девятью цифрами после запятой? Очень просто — две целых семь десятых плюс два раза Лев Толстой — 2.718281828, т. е. к числу 2.7 нужно приписать два раза год рождения классика (1828). Остается самая малость — запомнить, в каком году родился Лев Толстой, или, на худой конец, сообразить, что это случилось в позапрошлом веке, чтобы вспомнить хотя бы три знака числа `e` после запятой: 2.718. Ретроспектива, обращенная в XIX век, поможет запомнить довольно-таки точное значение одной из фундаментальных констант математики.

Как запомнить, что факториал нуля равен не нулю (типичная ошибка), а единице? Очень просто. Нужно применить *ретроспективный* метод поиска факториала числа: сообщить машине факториал какого-либо положительного числа `N` (`5!=1·2·3·4·5=120`, например) и то, что факториал предыдущего числа `N−1` равен факториалу первого (`N!`), разделенному на `N`. Факториал пяти (`120`) — это такая же тривиальная истина, как и то, что Лев Толстой родился в XIX веке, а грубая оценка числа `e` — 2.7.

Работая по программе с двусторонней рекурсией, показанной на рис. 17, можно не только правильно определить факториал нуля (единица), но и получить факториалы отрицательных (?!) чисел. Только нужно обучить этому машину — заложить в программу двустороннюю рекурсию для поиска факториала на всем целочисленном диапазоне. "Двусторонность" здесь проявляется в том, что факториалы ищутся не только *справа*, но и *слева* от пяти.

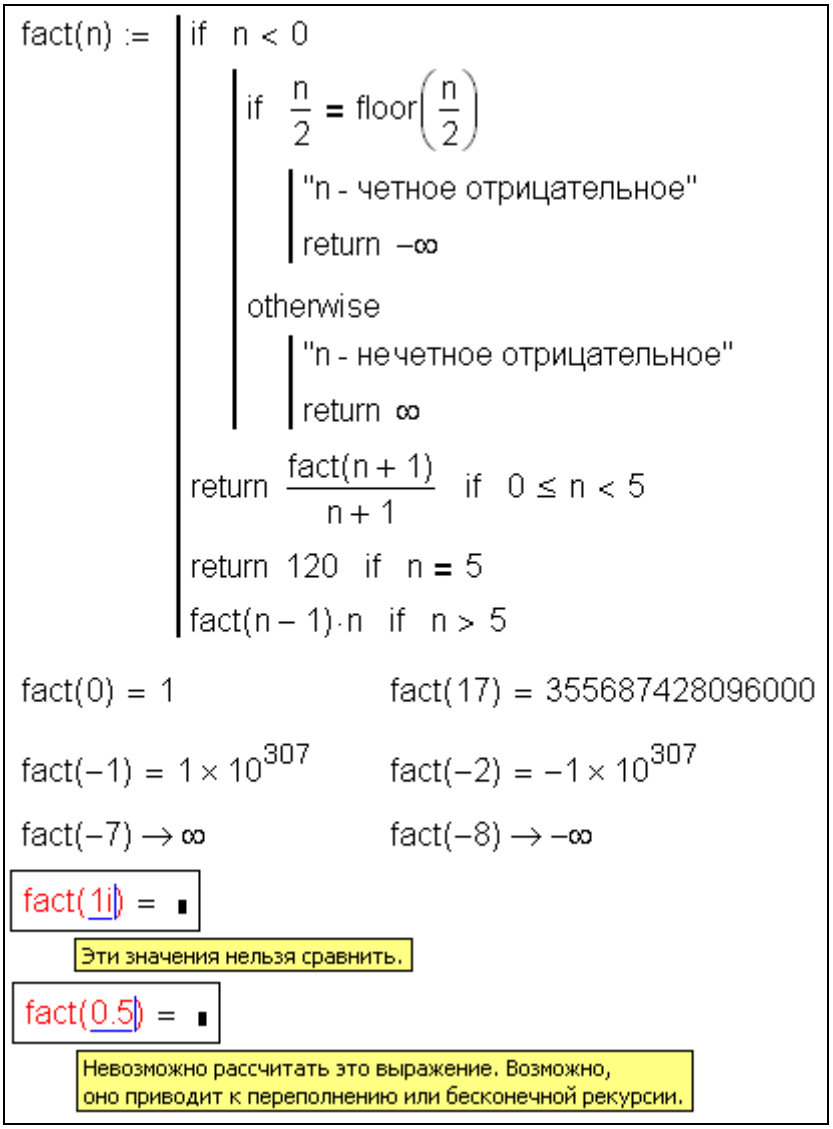

Рис. 17. Расчет факториала (двусторонняя рекурсия)

Рассмотрим числа Фибоначчи, которые связаны с условными кроликами (табл. 1).

Таблица 1. *Изменение популяции кроликов*

Характеристика	**Значения**									
Поколение	...	4	5	6	7	8	9	10	11	...

Число кроликов	...	3	5	8	13	21	34	55	89	...

Приведенный ряд специально начат не с традиционного места (первое поколение), а с четвертого поколения (три кролика), для того чтобы задать читателю вопрос, подобный тому, который стоял в задаче о факториале: "Чему равно минимальное количество кроликов в популяции — каково наименьшее число Фибоначчи?" Нормальный ответ, приводимый во всех учебниках — ноль. Но не будем спешить и напишем программу с двусторонней рекурсией, взяв за базовые числа Фибоначчи не традиционную пару 0 и 1, а 3 и 5 (четвертое и пятое поколения) — см. рис. 18.

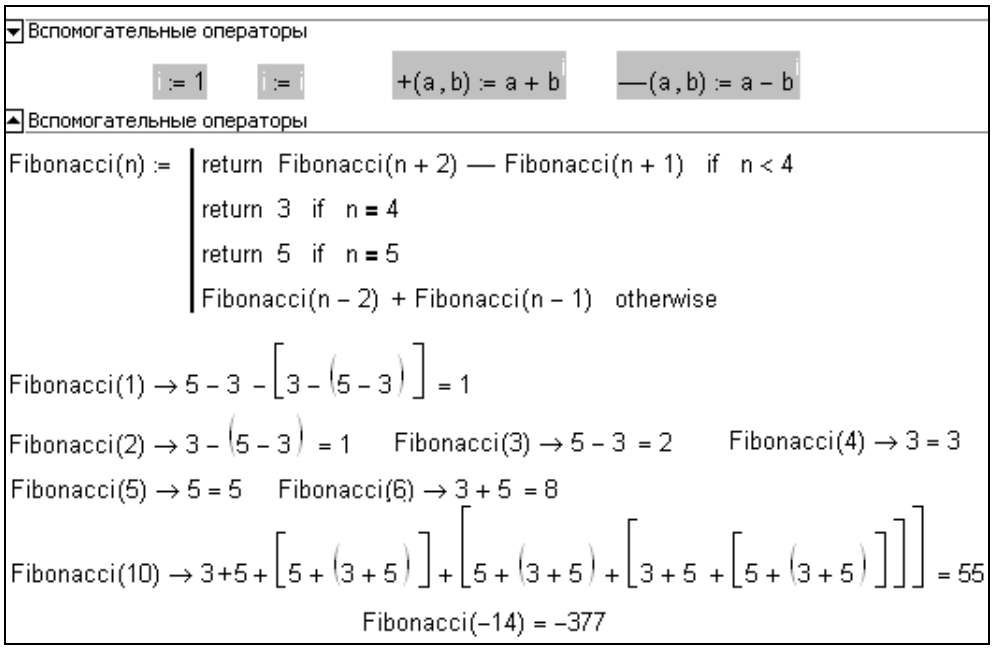

Рис. 18. Расчет чисел Фибоначчи (двусторонняя рекурсия)

Ряд кроликов Фибоначчи в "отрицательных поколениях" зеркально отображает значения в "положительных поколениях", но с переменным знаком.

Числа Фибоначчи в наше время широко применяются в вычислительной математике, в том числе и для иллюстрации рекурсии, как, например, в нашей книге. Кроме того, метод Фибоначчи используется для поиска минимума. Частный случай метода Фибоначчи — метод золотого сечения (см. рис. 1.66): пара смежных чисел Фибоначчи при N, стремящемся к бесконечности, соотносится по золотому сечению.

Но "изюминка" программы, показанной на рис. 18, не только и не столько в "двусторонности", а в том, что возвращаются *все* рекурсивные вызовы функций. Это достигается заменой встроенных операторов сложения и вычитания, на пользовательские аналогичные операторы с маленьким секретом, который более подробно раскрыт на

рис. 19, где представлена рекурсивная программа, возвращающая так называемую непрерывную (цепную) дробь. В этой задаче ответ повторяет исходные данные, а сама суть решения задачи сосредоточена в рекурсивных вызовах.

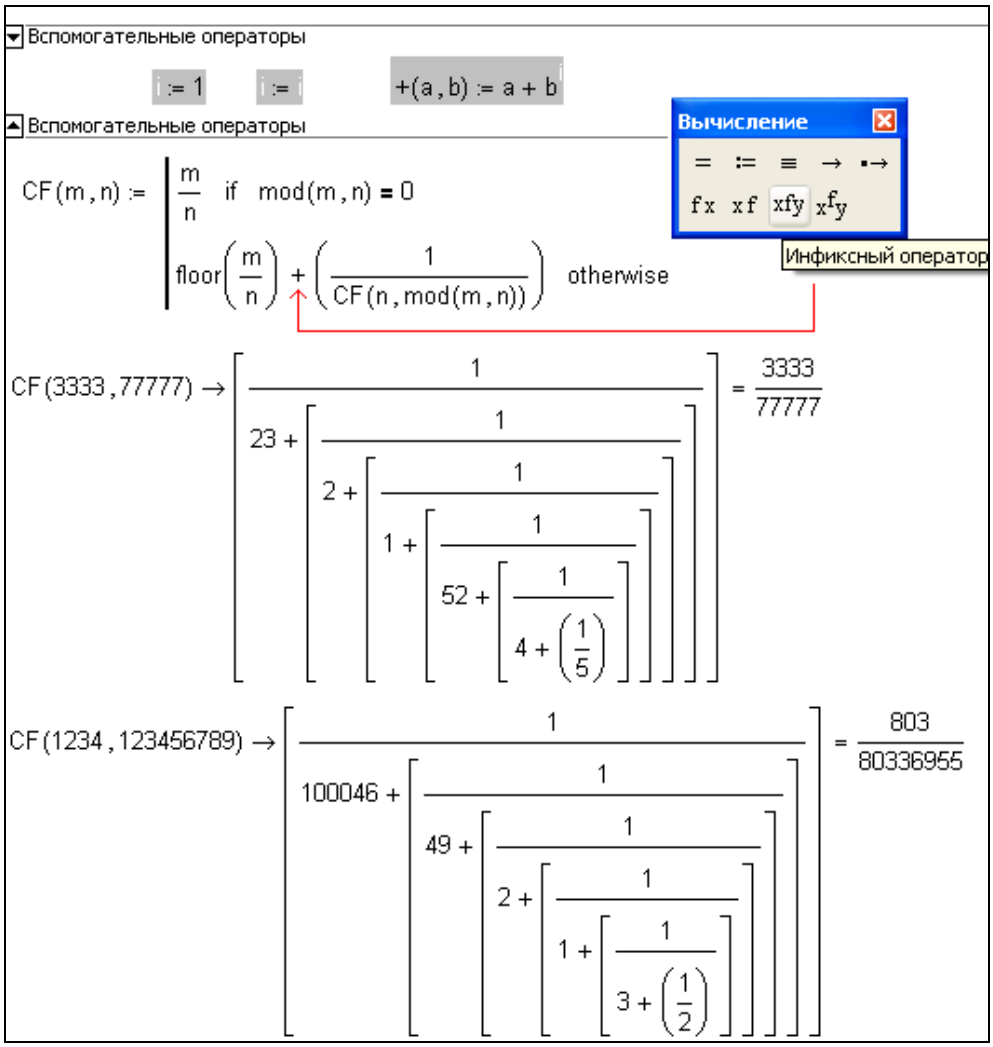

Рис. 19. Непрерывная дробь

Пользовательский оператор сложения на рис. 19 имеет степень у второго слагаемого. Это видно за счет того, что оператор прописан на цветном фоне. Показатель степени равен одному, что никак не влияет на вычисления, но позволяет выводить их, если заглушить численное значение переменной `i`. Этот же прием скрытой степени позволяет осуществить "вековую мечту" пользователей Mathcad — выводить на печать не

только результат, но и все числа, участвующие в расчете (см. **http://twt.mpei.ac.ru/mas/worksheets/202_All_Values.mcd**).

5. Сказка о рыбаках и рыбке, или проблема метки

Как было уже отмечено ранее, язык программирования Mathcad не имеет *меток* и операторов условного и безусловного перехода к меткам.

Проблему метки, вернее, проблему избавления от метки при переписывании "меточных" программ для "безметочного" пакета Mathcad, мы рассмотрим на примере решения старинной английской задачи о *рыбаках и рыбке*.

"Три рыбака легли спать, не пересчитав и не поделив улова. Ночью один из рыбаков проснулся и решил уйти, забрав свою долю. Но число рыб не делилось на три. Тогда он, не долго думая, выбросил одну рыбку, а из остатка забрал треть. Второй и третий рыбаки поступили аналогичным образом — ушли по-английски и по-джентельменски, выбросив по одной рыбке и оставив спящим товарищам четное число рыб. Спрашивается, какое минимальное число рыб в улове отвечает условию задачи".

Пусть читатель сначала попробует решить задачу без компьютера. Компьютер же мы привлечем к этой работе для двух целей. Во-первых, на задаче о рыбаках и рыбке мы рассмотрим некоторые методы *структурирования* программ в среде Mathcad — методы освобождения программ от меток и "втискивания" их в узкие рамки структурных управляющих конструкций, описанных выше (см. рис. 3). Во-вторых, мы покажем, что задача о рыбаках и рыбке до сих пор решалась неправильно...

Эту задачу можно решить и "в лоб" методом последовательных приближений — задается первое приближение к ответу (50 рыб, например), а затем от этого числа отнимается по единице до тех пор, пока убывающий улов не будет представлять собой целочисленный ряд: было 25 рыб (искомый ответ задачи), первый рыбак выбросил одну, забрал треть и оставил товарищам 16 рыб (по 8 каждому); второй рыбак (не зная, что первый ушел) оставил 10 рыб, а третий — 6. Задача решена, но с применением "ручной" работы, состоящей в наблюдении за значениями переменной `Улов` и в изменении (уменьшении на единицу) значения переменной `Ответ`.

Попробуем автоматизировать поиск ответа в задаче о рыбаках и рыбке, используя "меточный" язык BASIC.

```
' 1. Исходная неструктурированная BASIC-программа
Input "Предположение"; Ответ
label: Улов = Ответ
  For Рыбак = 1 To 3
    Улов = Улов — 1
    Улов = Улов - Улов / 3
    If Улов > Int(Улов) Then Ответ = Ответ - 1: Goto label
  Next
Print "Ответ "; Ответ; "рыб"
```

```
' 2. Первый шаг структурирования - разбег
Input "Предположение"; Ответ
Ответ = Ответ + 1              ' Шаг назад
label: Ответ = Ответ - 1       ' Шаг вперед
  Улов = Ответ
  For Рыбак = 1 To 3
    Улов = Улов — 1
    Улов = Улов - Улов / 3
    If Улов > Int(Улов) Goto label
  Next
Print "Ответ "; Ответ; "рыб"

' 3. Второй шаг структурирования — ввод признака
Input "Предположение"; Ответ
Ответ = Ответ + 1
label: Ответ = Ответ — 1
Улов = Ответ
Поделили = "да"                ' Признак дележа улова
  For Рыбак = 1 To 3
    Улов = Улов — 1
    Улов = Улов - Улов / 3
    If Улов > Int(Улов) Then Поделили = "нет"
  Next
If Поделили = "нет" Goto label
Print "Ответ "; Ответ; "рыб"

' 4. Третий шаг структурирования — отказ от метки
Input "Предположение"; Ответ
Ответ = Ответ + 1
Do                             ' Начало цикла с постпроверкой
  Ответ = Ответ — 1
  Улов = Ответ
  Поделили = "да"
  For Рыбак = 1 To 3
    Улов = Улов — 1
    Улов = Улов - Улов / 3
    If Улов > Int(Улов) Then Поделили = "нет"
  Next
Loop Until Поделили = "да"     ' Конец цикла
Print "Ответ "; Ответ; "рыб"
```

Выше представлены три BASIC-программы, решающие задачу о рыбаках и рыбке в автоматическом режиме: оператор `Input` запрашивает первое приближение (те же 50 рыб, к примеру), а оператор `Print` выдает ответ — 25 рыб. В первой BASIC-

программе один к одному реализован алгоритм "ручного" расчета: как только в теле цикла с параметром (три последовательных ухода рыбаков) переменная `Улов` обретает дробный "хвостик" (встроенная BASIC-функция `Int` этот "хвостик" обрезает; ее Mathcad-аналог — функция `floor`), то (`Then`) предположение уменьшается на единицу (`Ответ=Ответ-1`), а управление программой передается оператору, помеченному меткой (`Goto label`). Прежде чем эту программу перевести на язык Mathcad, ее нужно освободить от метки[25]. И не только потому, что в арсенале средств программирования Mathcad нет метки и операторов условного и безусловного перехода к метке, но по другим причинам, не связанным с Mathcad. В нашей коротенькой программе-безделушке (см. в ней п. 1) метка вполне уместна и естественна, но если программа с метками разрастается, то в ней становится трудно разбираться и ее практически невозможно отлаживать и расширять. Программа, как справедливо подчеркивают адепты структурного программирования, становится похожа на спагетти: вытаскиваешь (вилкой) блок операторов для отдельной отладки или компиляции, а к нему намертво привязаны нити (макаронины) `Goto`-переходов. Кроме того, такую программу невозможно создавать группой разработчиков (технология снизу вверх). Первые реализации языка Pascal совсем не имели меток, т. к. этот язык разрабатывался Н. Виртом в первую очередь для обучения студентов структурному программированию. Метка появилась только в поздних версиях данного языка. Так от детей в период, когда они учатся жить (выживать!), прячут спички.

Первый шаг структурирования BASIC-программы — это превращение конструкции:

```
If Улов>Int(Улов) Then Ответ=Ответ-1: Goto label
```

в оператор условного перехода к метке:

```
If Улов>Int(Улов) Goto label
```

Это сделать несложно (см. п. 2), применив в программе технику разбега спортсмена перед прыжком: шаг назад от черты-метки (`Ответ=Ответ+1`) и разбег (`Ответ=Ответ-1`). Структурирование программы, как правило, несколько усложняет алгоритм: "За все нужно платить!", "Красота требует жертв!" и т. д.

Второй шаг структурирования — это вытаскивание оператора безусловного перехода из тела цикла `For`. Для этого в программу (см. п. 3) вводится вспомогательная булева переменная-признак `Поделили`, а в теле цикла оператор условного перехода к метке заменяется на оператор "альтернатива с одной ветвью" (одна из основных структурных управляющих конструкций). Сам же оператор условного перехода "сползает" вниз. После этого в программе "вырисовывается" еще одна основная структурная управляющая конструкция — цикл с постпроверкой, который в третьем варианте программы реализован через метку и оператор условного перехода к метке. После этого программу наконец-то можно совсем освободить от метки, реализуя алгоритм

[25] Структурная революция началась со статьи Э. Дейкстры (Edsger Wybe Dijkstra) "Программирование без GOTO" ("Programming without GOTO"). Сейчас многие программисты-снобы гордятся и хвастают тем, что они написали не одну сотню серьезных программ, ни разу не обратившись к метке.

через цикл с постпроверкой (`Do... Loop Until...`), в тело которого вписан цикл `For`, а в него — альтернатива с одной ветвью (см. п. 4).

После всех этих манипуляций BASIC-программу можно один к одному переписать для Mathcad (рис. 20). Придется только оформить ее в виде функции пользователя: в языке Mathcad нет операторов `Input` и `Print`[26]. Их аналоги в среде Mathcad (операторы ■ := ■ и ■ =) работают, увы, только вне программ.

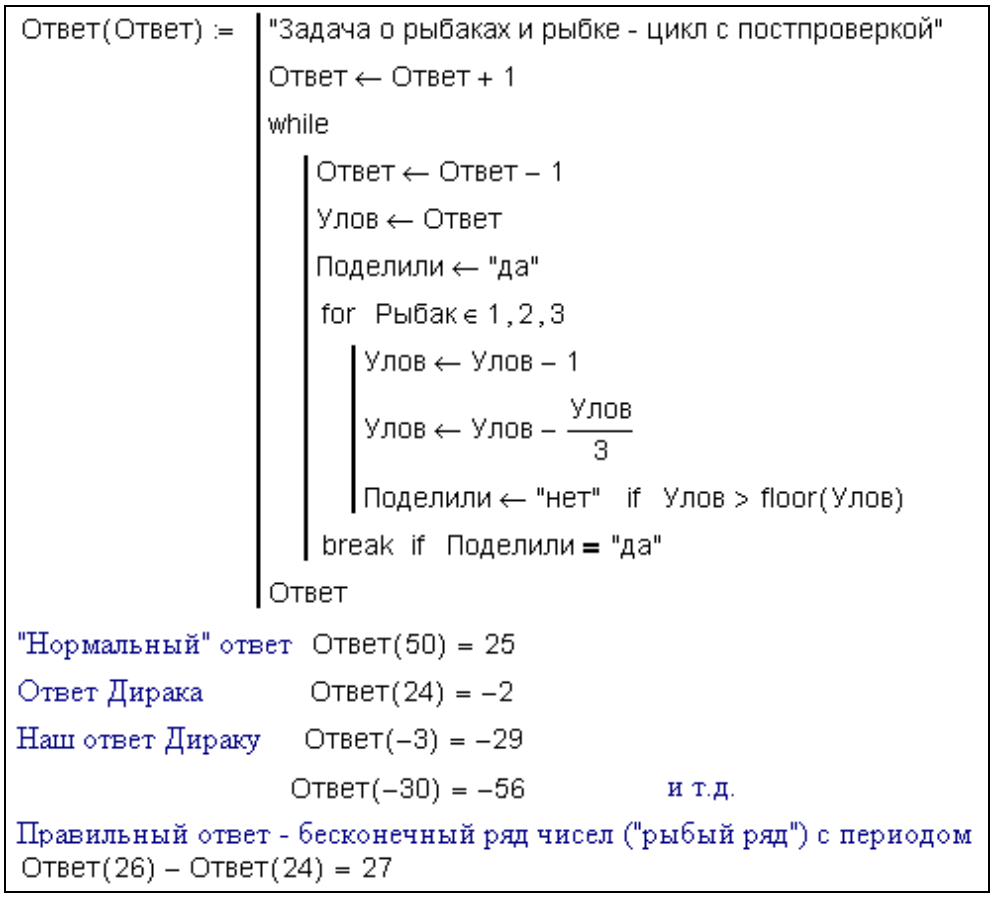

Рис. 20. Задача о рыбаках и рыбке — проблема метки

На рис. 20 показаны вызовы функции `Ответ` при различных значениях предположений (50, 24, –3 и даже –30 рыб). Английский физик Поль Дирак придумал не только античастицы, но и "антирыбы": он сказал, что задача о рыбаках и рыбке решалась неправильно (25 рыб). Правильный ответ — минус две рыбы (плюс две антирыбы): выбрасываем одну — остается минус три, забираем треть — остается минус две и так

[26] В языке Visual Basic оператора `Print` тоже нет, а есть *метод* `Print`.

до бесконечности. Наше компьютерное решение задачи показывает, что и Дирак ошибался: "Поль, ты не прав!" Условию задачи отвечает бесконечный ряд чисел (назовем его "рыбный ряд Дирака") с шагом 27.

Чтобы не прослыть совсем уж полным педантом (программистом-занудой), можно в конец цикла `for` на рис. 20 вставить оператор

```
break if Поделили = "нет"
```

прерывающий выполнения цикла `for`. Если рыбаков будет не трое, а больше (тридцать три рыбака, например — задача для читателя), то этот прием существенно ускорит работу программы.

Можно еще усложнить задачу, заставив любое количество рыбаков выбрасывать или подлавливать любое количество рыб.

Задачу о рыбаках и рыбке можно решать другим способом — перебором с другого конца: задать не начальное число рыб в улове, а предположить, что последний рыбак оставляет две рыбы (меньше не может быть), и увеличивать их число на единицу, если условия задачи не выполняются. Задача будет решаться быстрее, но –2 рыб, а, тем более, –29 рыб мы не получим: "Keep it simple, stupid! — Делай это проще, дурачок!" Человеку психологически трудно спуститься к отрицательным числам в ответе, машина же делает это спокойно, без всяких предрассудков. Не дает отрицательного ответа и аналитическое решение задачи — поиск целочисленных корней одного уравнения с двумя неизвестными.

6. Размерность в Mathcad-программах

По знаменитой формуле Н. Вирта программа — это сумма алгоритма и структуры данных. Алгоритм программы в среде Mathcad задается нажатием десяти кнопок на панели **Программирование** (см. рис. 3), отвечающих за циклы, альтернативы и прочее. Структура данных в среде Mathcad также проста. Там есть только вещественные переменные двойной точности, которые могут превращаться в комплексные или целочисленные, группироваться в векторы и матрицы (простые и вложенные), принимать, если надо, текстовые значения.

Тип числовой переменной в среде Mathcad, в какой-то мере заменяется *размерностью* хранимой величины, что очень удобно в инженерных расчетах. По программе на рис. 21 рассчитываются два параметра треугольника. Аргументами функции `Параметры_треуг` могут быть величины с разной размерностью длины (метры — `m`, сантиметры — `cm`, дюймы — `in`). Система Mathcad сама в них разберется, сделает нужные пересчеты и выдаст правильный результат в выбранной пользователем системе единиц (кг-м-с, г-см-с, СИ или британская).

$\text{Параметры\_треуг}(a,b,c) := \left| \begin{array}{l} p \leftarrow a+b+c \\ S \leftarrow \sqrt{\frac{p}{2}\cdot\left(\frac{p}{2}-a\right)\cdot\left(\frac{p}{2}-b\right)\cdot\left(\frac{p}{2}-c\right)} \\ \begin{pmatrix} p \\ S \end{pmatrix} \end{array} \right.$

$\text{Параметры\_треуг}(1.1\cdot m, 112\cdot cm, 700\cdot mm) = \blacksquare\, m$ Mathcad 11

The units in this expression do not match.

$\text{Параметры\_треуг}(1.1\cdot m, 112\cdot cm, 700\cdot mm) = \cdot m$ Mathcad 12, 13 и 14

Решение 1

Значение имеет единицы измерения: Length, а должно иметь единицы измерения: Unitless.

$\text{Параметры\_треуг}(a,b,c,i) := \left| \begin{array}{l} p \leftarrow a+b+c \\ S \leftarrow \sqrt{\frac{p}{2}\cdot\left(\frac{p}{2}-a\right)\cdot\left(\frac{p}{2}-b\right)\cdot\left(\frac{p}{2}-c\right)} \\ \text{return } p \text{ if } i = \text{"Периметр"} \\ \text{return } S \text{ if } i = \text{"Площадь"} \end{array} \right.$

Mathcad 11

$\text{Параметры\_треуг}(1.1\cdot m, 112\cdot cm, 700\cdot mm, \text{"Периметр"}) = 2.92\, m$

$\text{Параметры\_треуг}(1.1\cdot m, 112\cdot cm, 700\cdot mm, \text{"Площадь"}) = 0.369\, m^2$

Mathcad 12

$\text{Параметры\_треуг}(1.1\cdot m, 112\cdot cm, 700\cdot mm, \text{"Периметр"}) =$

$\text{Параметры\_треуг}(1.1\cdot m, 112\cdot cm, 700\cdot mm, \text{"Площадь"}) = \blacksquare$

This value must be dimensionless.

Решение 2

$\text{Параметры\_треуг}(a,b,c) := \left| \begin{array}{l} p \leftarrow a+b+c \\ S \leftarrow \sqrt{\frac{p}{2}\cdot\left(\frac{p}{2}-a\right)\cdot\left(\frac{p}{2}-b\right)\cdot\left(\frac{p}{2}-c\right)} \\ \begin{pmatrix} \frac{p}{m} \\ \frac{S}{m^2} \end{pmatrix} \end{array} \right.$

$\text{Периметр\_треуг}(a,b,c) := \text{Параметры\_треуг}(a,b,c)_0 \cdot m$

$\text{Площадь\_треуг}(a,b,c) := \text{Параметры\_треуг}(a,b,c)_1 \cdot m^2$

$\text{Периметр\_треуг}(1.1\cdot m, 112\cdot cm, 700\cdot mm) = 292\, cm$

$\text{Площадь\_треуг}(1.1\cdot m, 112\cdot cm, 700\cdot mm) = 571.224\, in^2$

Рис. 21. Размерная Mathcad-программа

Система Mathcad, кроме того, ведет контроль размерностей и не позволяет складывать, например, метры с килограммами или секундами.

Функция в среде Mathcad, как уже отмечалось, может возвращать несколько значений, объединенных в вектор. Программно заданная пользовательская функция на рис. 21 должна возвращать значение периметра и площади треугольника. Но если пользователь захочет сделать размерными аргументы, чтобы функция вернула ему значение периметра и площади треугольника с соответствующими размерностями, то его ждет неудача. При этом будут выданы дезинформирующие сообщения об ошибке, которые в переводе гласят: "Несоответствие единиц измерения" (Mathcad 11), "Это значение должно быть безразмерным" (Mathcad 12), или... Пользователь будет думать, что он складывает метры с килограммами, но причина в другом — вектор в среде Mathcad может хранить переменные только одной размерности.

Ошибок в среде Mathcad, как и в любой другой программной среде, достаточно. Автор акцентирует внимание на этой ошибке по двум причинам. Во-первых, автор когда-то потратил уйму времени, отлаживая программу, подобную программе на рис. 21, и не понимая, в чем суть ошибки. Во-вторых, дал об этом знать разработчикам Mathcad, но эта ошибка почему-то так и не была исправлена. Кстати, об ошибках...

Считается, что по-настоящему красивая женщина ("чертовски красивая") непременно должна иметь внешний дефект (ошибку Природы), небольшой, но сразу бросающийся в глаза: вздернутый нос, родинка, веснушки... Такие "украшения" лишний раз напоминают о том, что это не богиня, от которой лучше держаться подальше, не бездушная "кукла восковая", а земная женщина. Вот хрестоматийный пример: Наталья Николаевна Пушкина (урожденная Гончарова) — петербургская красавица, которая, тем не менее, чуть-чуть косила. А вот другой пример — Шекспир воспел "смуглую леди сонетов" в те времена, когда белизна лица считалась непременным атрибутом женской красоты.

Слово "чуть-чуть", только что промелькнувшее в тексте, напоминает о кратком, но точном определении художественного вкуса: "Искусство — это чувство меры". Рафинированное произведение искусства, созданное на основе чистых канонов, находится как бы в неравновесном состоянии. Маленький щелчок ("глюк" в программе, родинка на лице красавицы или ее легкое косоглазие, кривая колокольня в Пизе, корявый автограф в углу картины или темный штрих в биографии художника или программиста[27], на худой конец) сталкивает эту шаткую балансирующую конструкцию либо в чулан поделок (кич), либо в сокровищницу шедевров.

А вот еще пример, поворачивающий проблему на новую грань, где пересекаются плоскости формы и содержания. Сергей Довлатов в своих записках упоминает об известном профессоре-филологе с такими косыми глазами, что с ним трудно было общаться — непонятно, в какой глаз нужно смотреть. Этот профессор, прикрывая

[27] Который, например, что-то "позаимствовал" у другого программиста, но не признается в этом даже на Страшном суде.

рукой левый глаз, говорил собеседнику: "Смотрите в правый. На левый не обращайте внимания. Левый — это дань формализму". Хорошо дурачиться, создав предварительно целую филологическую школу — ошибки простительны только в гениальных программах.

Если читатель увидит опечатки или даже ошибки в этой книге, то автор просит, во-первых, сообщить ему об этом, а, во-вторых, перечитать то, что написано выше.

Итак. Работа с размерностями вообще требует особой аккуратности. Вот хороший совет при работе в среде Mathcad. Вводить в Mathcad-документ новую переменную лучше оператором `m=` (вывод значения), а не оператором `m:=` (ввод значения). Этим пользователь проверяет лишний раз, не занята ли уже переменная `m` хранением заданной ранее величины. В Mathcad 7 и выше этот прием не обременителен, т. к. в среде этих версий оператор `m=` автоматически (SmartOperator) превращается в оператор `m:=`, если переменная `m` свободна от хранения чего-либо.

Предпроверку переменных особо можно рекомендовать в расчетах с использованием размерностей физических величин. В этом режиме (а он включается по умолчанию) предопределенными (системными) будет огромное число "популярных" переменных (`A`, `c`, `C`, `F`, `g`, `H`, `J`, `K`, `L`, `m`, `N` и т. д.), хранящих единичные значения физических величин (сила тока, скорость, заряд, емкость, ускорение, индуктивность, энергия, температура, длина, количество вещества и т. д.). В таком расчете "невинное" выражение `m:=3` развалит весь стройный порядок единиц измерения: вместо метров появится, черт знает что.

> ***Примечание***
> Если единицы физических величин не используются, то нужно их совсем отключить (диалоговое окно **Система единиц измерения,** вызываемое командой **Параметры документа** в меню **Сервис**).

Можно заставить функцию, показанную на рис. 21, выдавать две величины не сразу, а попеременно — в зависимости от значения дополнительного аргумента (решение 1): если `i="Периметр"`, то функция `Параметры_треуг` возвращает периметр треугольника, если `i="Площадь"`, то — площадь. В среде Mathcad 11 такой прием был допустим, а в среде Mathcad 12, 13 и 14 — нет.

Кардинальное решение проблемы — лишение переменных размерностей в массиве, и возврат их вне программы (решение 2 на рис. 21).

7. Отладка программ

Одна из самых слабых сторон средств программирования, встроенных в Mathcad до 13 и 14 версий, — это практически полное отсутствие каких-либо специализированных инструментов *отладки программ*. Программу можно написать и за час (день, неделю, месяц...), но искать ошибку в ней можно весь оставшийся день (неделю, месяц, год...). Поэтому все традиционные среды программирования, как правило, имеют минимум три инструмента отладки — трассировку программы, пошаговое (пооператорное) выполнение программы и наблюдение за значениями переменных при

выполнении программы. С другой стороны, Mathcad-программы — это обычно небольшие фрагменты, совсем не похожие на тех монстров в несколько тысяч строк, с какими приходится сталкиваться в традиционном программировании и которые без специальных средств отладить практически невозможно.

Уже отмечалось, что механизм локальных переменных Mathcad, задуманный как средство реализации принципа программирования "снизу вверх" (см. рис. 4), препятствует наблюдению за значениями переменных программы вне программы. Это ограничение можно обойти, заставив Mathcad-программу возвращать не требуемое (запланированное) значение, а массив (матрицу или вектор), хранящий в качестве своих элементов все локальные переменные. После такой отладки программы массив можно заменить на скаляр либо оставить все как есть, а оператором ■[■(■■) изымать из массива, возвращаемого программой, нужный элемент. Тут, правда, нужно быть осторожным в том случае, если переменные программы хранят разноразмерные значения.

Если необходимо следить за значением локальной переменной или группы переменных в *цикле*, то можно порекомендовать в тело цикла вставить оператор `return ■ if ■`, поставив за словом `if` управляющую булеву переменную. Все псевдоанимации, о которых было рассказано в книге *(см. разд. 1.7),* — это по своей сути Mathcad-программы в режиме отладки, когда нажатие кнопки отправки увеличивает на единицу значение переменной, связанной с этой кнопкой, что влечет за собой вывод новых данных из цикла с последующей их графической иллюстрацией.

Но изменяющиеся в цикле значения переменных можно видеть и непосредственно в самой программе, если вставить в цикл тандем операторов ■←■→ (локальное присваивание и символьный вывод). В программе, показанной на рис. 22, этот тандем позволил создать в среде Mathcad... секундомер.

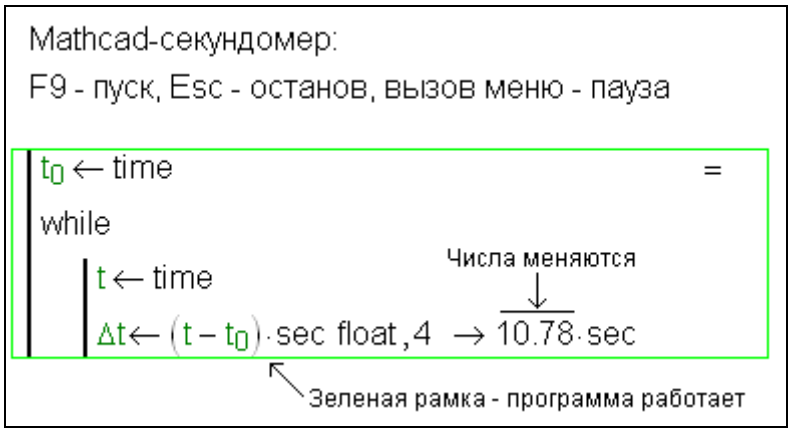

Рис. 22. Секундомер в Mathcad

Если в Mathcad-программу ввести бесконечный цикл и вызывать в нем функцию `time`, а разницу во времени запуска программы и вызовом функции `time` выводить

символьно[28], то и получится секундомер — справа от символа → будут мелькать числа, отсчитывающие секунды, дающие информацию о том, что происходит в самом цикле, а это и есть суть отладки.

Но наблюдение за значениями переменных в цикле, показанное на рис. 22, — это скорее курьез[29], чем серьезный инструмент отладки Mathcad-программ. "По серьезному" делать трассировку цикла можно через другой недокументированный прием — через замену оператора, добавляемого нажатием кнопки **Add Line**, на оператор вставки массива — вектора или матрицы (рис. 23).

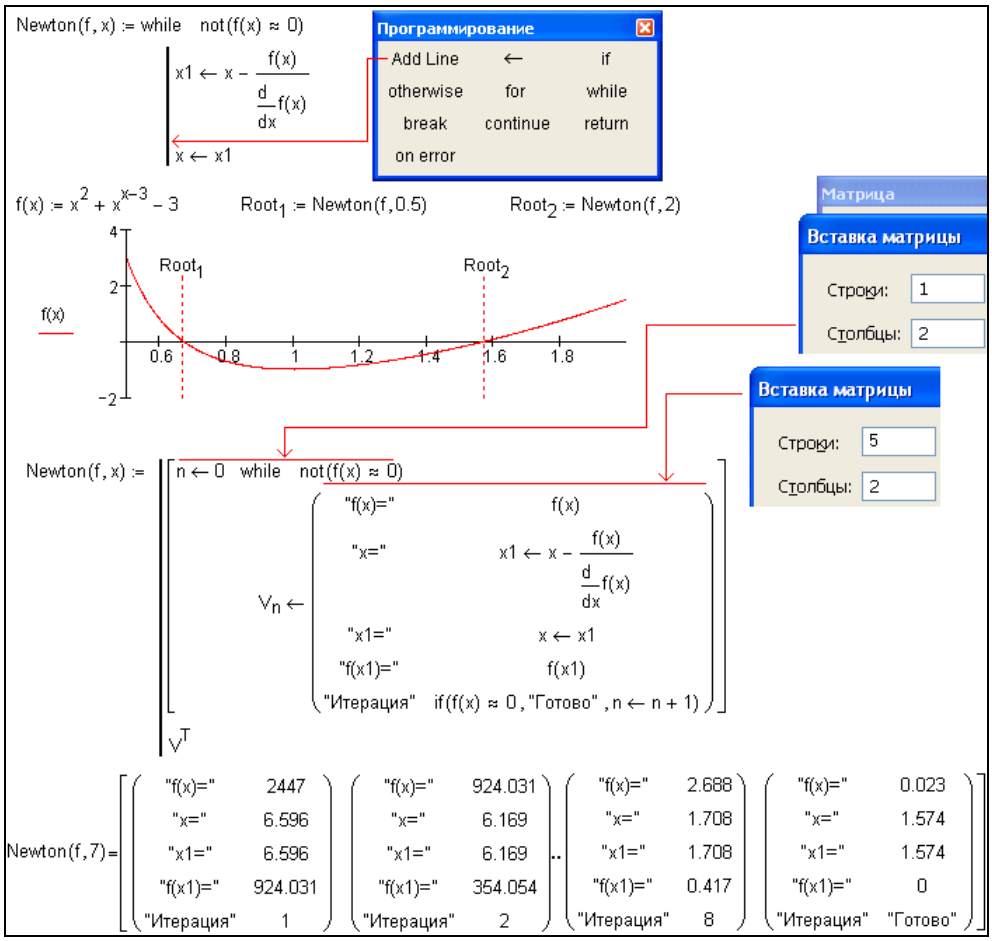

Рис. 23. Программа-матрица

[28] А почему здесь нельзя вставить оператор =? Это бы существенно ускорило процесс отладки программ: символьная математика — это медленная математика.

[29] Цифры в таком цикле часто сменяются слишком быстро. Из-за этого программу приходится спутывать как пасущуюся лошадь — вставлять в цикл новые пустые, замедляющие циклы.

В верхней части рис. 23 показана Mathcad-программа поиска нуля функции методом Ньютона (касательных). В нижней части рис. 23 представлена та же программа, но со вставленным счетчиком цикла `n` и телом цикла в виде матрицы (а не виде цепочки операторов, введенных через нажатие кнопки **Add Line**), значение которой присваиваются вектору `V`. По завершении работы программы этот составной массив `V` (вектор, элементы которого — массивы) выводится на дисплей и отображает всю историю поиска нуля функции.

На рис. 24 показана Mathcad-программа вычисления суммы ряда.

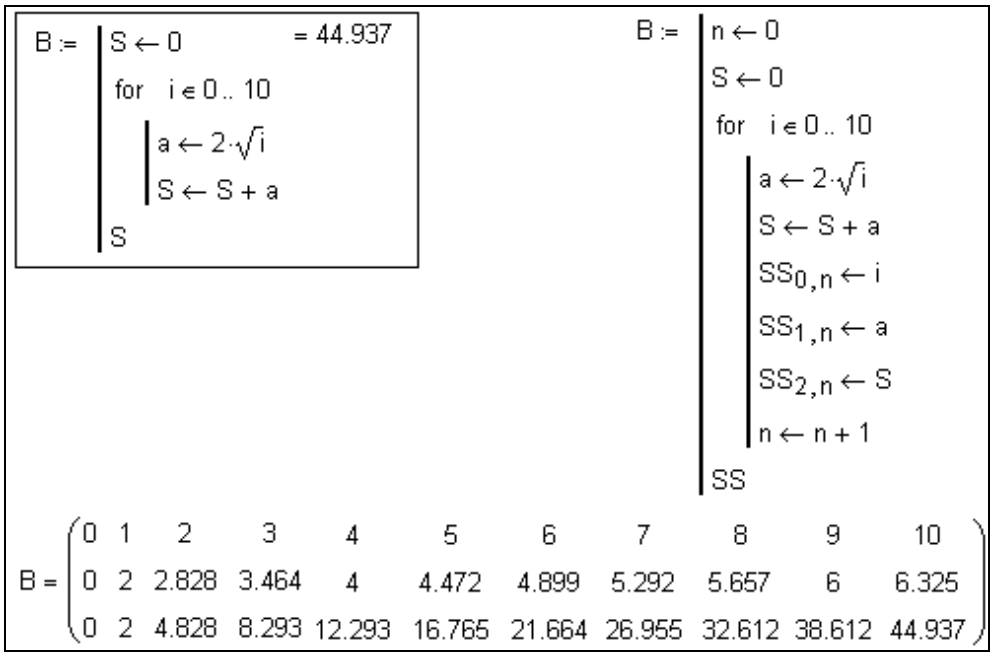

Рис. 24. Программа вычисления суммы ряда (слева) и она же с операторами сохранения промежуточных результатов (справа)

Первый вариант программы, показанный на рис. 24, возвращает только итоговый результат. Второй же вариант за счет ввода в программу дополнительных операторов и счетчика `n` выводит всю историю суммирования. Пользователи Mathcad шли и на другие ухищрения, чтобы иметь возможность видеть значения локальных переменных, которые они принимают в цикле. Например, записывали эти значения на диск операторами `WRITEPRN(file)` и `APPENDPRN(file, [M])`. Наконец-то этот прием стал встроенным в среде Mathcad 13/14, где появились следующие инструменты отладки программы:

- ❒ две встроенные функции `trace` и `pause`;
- ❒ окно трассировки программы;
- ❒ панель с четырьмя кнопками отладки;

На рис. 25 показана работа этих инструментов.

Главным недостатком пользовательских приемов отладки, показанных на рис. 23 и 24, было то, что если программа зависнет, то нельзя будет узнать ни сам итоговый результат, ни содержание "трассировочной" матрицы.

Этот недостаток устранен с вводом в Mathcad 13/14 встроенных средств отладки.

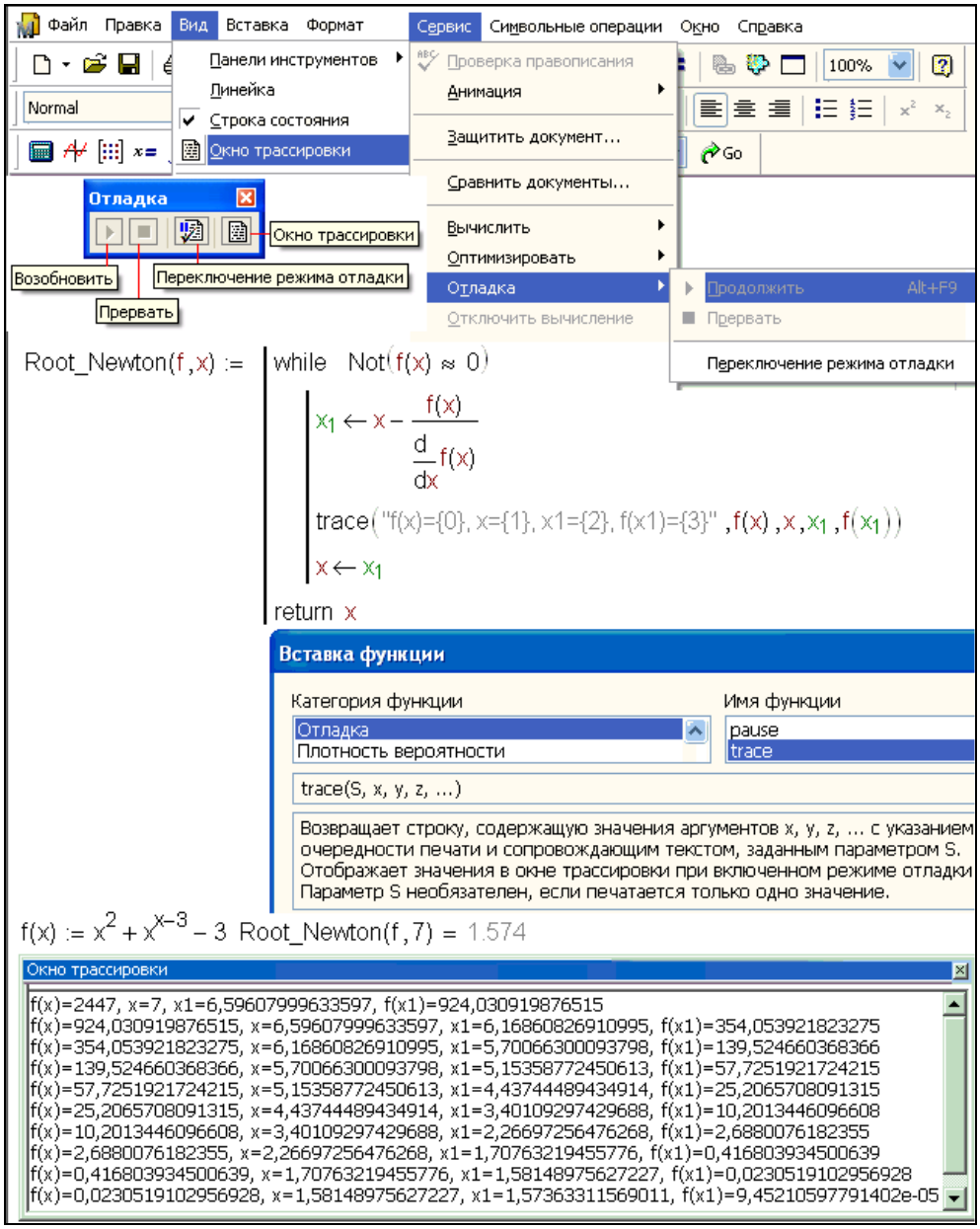

Рис. 25. Трассировка программы

На рис. 25 показано, как в программу поиска нуля функции `f(x)` вставлена дополнительная отладочная строка с новой функцией `trace` (трассировка). Кроме того, через команду **Вид | Окно трассировки** открыто одноименное окно отладки и включен

режим **Отладка** через меню **Сервис**. После этих манипуляций при вызове функции `Root_Newton`, в окне трассировки будут отображаться значения переменных, перечисленных в качестве аргументов функции `trace`.

Если в "отладочную" программу вставить не функцию `trace`, а функцию `pause`, то в окне трассировки будет выдаваться очередная строка значений аргументов функции `pause`, а сам расчет прерываться (детская игровая команда "замри!"). Продолжить расчет можно, нажав кнопку **Возобновить** новой панели инструментов **Отладка**, появившейся в Mathcad 13/14 и показанной в левом верхнем углу рис. 25.

8. Локальная функция

В Mathcad 12/13/14 появилась возможность работы с *локальными функциями*. Теперь в любой строке программы можно записать `f(x,...)`, нажать кнопку ← в панели инструментов **Программирование** и ввести в программу локальную функцию, область видимости которой ограничена местом вода этой функции и концом самой программы. На рис. 1.66 показана программно созданная функция с именем `Min_GR` поиска минимума функциональной зависимости методом золотого сечения, содержащая локальную функцию с именем `Golden_Ratio`, делящую отрезок неопределенности в золотом соотношении (Golden Ratio). Функция `Golden_Ratio` вызывается только в теле программы-функции `Min_GR` и нигде более. Этим и оправдана ее локальность. Но!

Если функцию `GR` вызвать вне тела цикла программы `Min_GR`, то ее работа прервется сообщением об ошибке, в переводе гласящим: "Данная переменная или функция не определена". Хорошо ли это или плохо?! С одной стороны, хорошо. Различные Mathcad-программы, разработанные разными людьми и собранные в одном Mathcad-документе, могут иметь одноименные локальные функции (`y(x)`, например), которые не будут "глушить" друг друга. Так был задуман и механизм локальных переменных, появившийся одновременно с появлением программирования в Mathcad (6-я версия). Тогда о локальных функциях, по-видимому, забыли или у разработчиков руки до них не дошли. С другой стороны, механизм локальности переменных и функций препятствует процессу отладки программ. Это тем более досадно, если учесть, что в среде Mathcad не так уж много встроенных средств отладки, которые часто приходится дополнять всякими пользовательскими хитростями и трюками для поиска ошибок в программах.

Как пользователь вводит локальную функцию в программу? Он, конечно, должен писать ее не в теле программы[30], а вне ее, чтобы иметь возможность тестировать ее — изменять значение аргументов и смотреть, что она возвращает. После отладки у функции нужно будет изменить знак (оператор) := на ←, а саму функцию перенести (перетащить) в программу, где она автоматически станет локальной и невидимой вне тела программы. А можно просто наложить `GR`-функцию на тело программы `Min_GR`,

[30] Мы не рассматриваем случай, когда локальные переменные, объявленные в программе выше места ввода локальной переменной, "проникают" в локальную переменную, минуя "вход" через список аргументов локальной функции.

заменив знак := на ≡ (или с самого начала иметь не знак :=, а знак ≡), а стиль имени функции с Variables на Local Var, например. Так, собственно, и поступал автор уже давно (см. рис. 1.66 и сайт **http://twt.mpei.ac.ru/mas/worksheets/Gold_Ratio.mcd** с программой поиска минимума функции методом золотого сечения (технология Mathcad Application Server), а также совет 185 из книги "Советы пользователям Mathcad" — **http://twt.mpei.ac.ru/ochkov/Sovet_MC**).

О знаках := и ←. Хорошо бы было, чтобы оператор := в программах присваивал соответствующей функции или переменной стиль Variables, "видимый" и вне программы, а оператор ← — стиль Local Var, видимый только в программе, а может быть, и вне программы тоже. Это внесло бы существенный вклад в процесс создания встроенных инструментов отладки программ Mathcad, о которых мы говорили выше. Еще было бы лучше, если бы одной командой (переключателем) можно было бы менять с локального на глобальный стиль всех объявленных в программе переменных и функций.