Н.Н. ДМИТРИЕВ, В.Ю. САХАРОВ

МОДЕЛИРОВАНИЕ ИНЖЕНЕРНЫХ ЗАДАЧ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ FREE PASCAL В СРЕДЕ LAZARUS

Министерство образования и науки Российской Федерации
Балтийский государственный технический университет «Военмех»

Н.Н. ДМИТРИЕВ, В.Ю. САХАРОВ

МОДЕЛИРОВАНИЕ ИНЖЕНЕРНЫХ ЗАДАЧ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ FREE PASCAL В СРЕДЕ LAZARUS

Учебное пособие

Санкт-Петербург
2012

УДК 004.43(075)
Д53

Дмитриев, Н.Н.

Д53 Моделирование инженерных задач на языке программирования Free Pascal в среде Lazarus: учебное пособие / Н.Н. Дмитриев, В.Ю. Сахаров; Балт. гос. техн. ун-т. – СПб., 2012. – 56 с.

ISBN 978-5-85546-721-5

Излагаются основы языка программирования Free Pascal. Рассматриваются алгоритмы решения некоторых математических задач. Разбирается алгоритм численного решения задачи Коши методом Рунге–Кутта четвертого порядка на примере задач механики.

Предназначено для студентов 2-4-го курсов всех специальностей.

УДК 004.43(075)

Р е ц е н з е н т ы: канд. техн. наук, проф. БГТУ *Н.Н. Смирнова*; канд. физ.-мат. наук, доц. БГТУ *А.Л. Илихменев*

Утверждено
редакционно-издательским
советом университета

ISBN 978-5-85546-721-5

П Р Е Д И С Л О В И Е

Данное учебное пособие предназначено для студентов, которым необходимы основы программирования на языке высокого уровня Free Pascal. В частности оно будет весьма полезным для студентов, которые изучают механику, и при решении задач нужно численно найти решение задачи Коши для системы дифференциальных уравнений. Такого вида задачи возникают в курсах теоретической механики, теории колебаний, основ теории трения и др.

Подчеркнем, что Free Pascal и среда разработки Lazarus – свободно распространяемый программный продукт. Установка Free Pascal или среды Lazarus на домашнем или рабочем компьютере дает возможность работать на легальном программном обеспечении. Кроме того, Lazarus устанавливается на ПК с ОС Windows, Linux или Mac OC X (http://lazarus.freepascal.org).

Для более глубокого изучения Lazarus и Free Pascal можно рекомендовать книгу К.Т. Мансурова «Основы программирования в среде Lazarus» (текст этой книги можно найти на сайте http://www. freepascal.ru).

1. ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

1.1. Структурное программирование

С момента зарождения программирования было создано множество языков общения человека с ЭВМ. Сейчас, по видимости, наибольшее распространение имеют языки программирования C++, Delphi, Visual Basic (Visual Basic for Application), Free Pascal и некоторые другие. При этом в каждом из этих языков есть поддержка всех классических управляющих конструкций.

К 70-м годам XX в. стало ясно, что программные проекты стали слишком сложными для успешного проектирования, кодирования и отладки в приемлемые сроки. Размер программ достиг величин, при которых программисты не могли с уверенностью сказать, что созданный программный продукт всегда выполняет то, что требуется и что он не выполняет ничего такого, что не требуется. Назрела проблема изменения подходов к созданию больших программных продуктов.

В 1969 г. Э. Дейкстра на международной конференции по программированию впервые использовал термин «структурное программирование» и предложил принципиально новый способ создания программ. Программа рассматривалась им как совокупность иерархических абстрактных уровней, которые позволяют четко структурировать программу и лучше ее понимать, доказывать корректность ее работы и тем самым повышать надежность функционирования программы и сократить сроки ее разработки.

Правила структурной методологии разрабатывались такими учеными как Вирт, Дейкстра, Дал, Хоар, Иордан и др. В этой связи следует отметить знаменитые книги Вирта [8, 9] и сборник [4].

1.1.1. Цели структурного программирования

Обеспечить дисциплину программирования в процессе создания программных комплексов. Дейкстра дал следующее определение: ***«Структурное программирование – это дисциплина, которую программист навязывает сам себе».***

Улучшить читабельность программы. Для этого следует избегать использования языковых конструкций с неочевидной семантикой; стремиться к локализации действия управляющих конструкций и применения структур данных; разрабатывать программу так, чтобы ее можно было читать от начала до конца без управляющих переходов на другую страницу.

Повышать эффективность программы. Данное положение достигается при структурировании программы, разбиении ее на модули так, чтобы можно было легко находить и корректировать ошибки, а также чтобы текст любого модуля с целью увеличения эффективности можно было переделать независимо от других.

Повышать надежность программы. Надежность обеспечивается хорошим структурированием программы при разбивке ее на модули и выполнением правил написания читабельных программ, что ведет к возможности сквозного тестирования и не создает проблем для организации процесса отладки.

Уменьшать время и стоимость программной разработки. Этот пункт выполняется, когда повышается производительность труда программиста, чему способствуют правила структурного программирования.

1.1.2. Основные принципы структурной методологии

Принцип абстракции. Абстракция позволяет придумать решение задачи без сиюминутного учета множества деталей. Поэтому проблема может рассматриваться по уровням: верхний уровень показывает большую абстракцию, упрощает взгляд на проект, нижний – показывает мелкие детали реализации задачи.

Принцип формальности. Формальность позволяет перейти от импровизации к строгому инженерному подходу при написании программ. Более того, этот принцип дает основания для доказатель-

ства правильности программ, так как позволяет изучать алгоритмы как математические объекты.

Принцип «разделяй и властвуй». Этот принцип означает возможность разделения программы на отдельные модули, которые просты по управлению и допускают независимую отладку и тестирование.

Принцип иерархического упорядочения. Данный принцип тесно связан с принципом «разделяй и властвуй» и выдвигает требования иерархического структурирования взаимосвязей между модулями программного комплекса, что облегчает достижение структурного программирования.

1.2. Модульное программирование

Модульное программирование – это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определенным правилам. Первое основное свойство программного модуля сформулировал Парнас (Parnas): «Для написания одного модуля должно быть достаточно *минимальных* знаний о тексте другого». Но только в 1975 г. Н. Вирт впервые предложил специализированную синтаксическую конструкцию модуля и включил в новый язык программирования **Modula.** Сейчас аналогичные конструкции имеются во многих языках.

Некоторые из этапов и приемов программирования и разработки алгоритмов [4]:

1. *Никаких трюков и заумного программирования*. Не нужно применять сложные методы там, где можно использовать простые.

2. *Как можно меньше переходов*. Программирование без **go to** – это еще не структурное программирование. В некоторых ситуациях переход по **go to** является лаконичным, простым и ясным средством. Но разумное применение конструкций **if-then-else** и **for-do** способствует большей ясности, чем использование оператора **go to.**

3. *Выбор с использованием конструкции* **if-then-else**. Цель данного положения – обеспечение простоты хода вычисления, без каких то ни было переходов извне внутрь рассматриваемой структуры.

4. *Простота циклов.* Переходы по программе назад почти всегда можно представить как некоторую форму цикла. Существуют формы цикла **for-do**, **while-do**, **repeat-until**.

Можно написать

k:=0; **while** k<1000 **do begin** операторы; k:=k+1 **end**.

Но проще и более ясно записать так

for k:=0 **step** 1 **until** 999 **do** операторы.

Таким образом, каждое средство языка программирования следует применять по назначению.

5. *Сегментация.* Громоздкие программы, состоящие из одной основной программы, как правило, невозможно читать. В отсутствие конструкций **if-then-else** она будет перегружена метками и переходами и ее структура будет неясна. Если имеется возможность использования **if-then-else**, то глубина вложенности циклов может стать настолько большой, что отыскание для каждого **if-then** соответствующего **else** будет затруднительно.

Чтобы обойти указанные трудности, каждую большую программу можно разбить на множество модулей или процедур (подпрограмм или функций), спроектированных так, что цель для каждой из них определена (логическая часть исходной задачи) и в каждой по возможности используются собственные локальные переменные. Не нужно стремиться разделить программу на равные куски («куски» в этом случае – самое подходящее слово для того, что получится при делении программы на равные части), а следует выделять логические фрагменты. Некоторые из них окажутся достаточно малыми для того, чтобы их можно было в таком виде оставить. Другие же, в свою очередь, потребуют разбиения на процедуры следующего уровня.

6. *Рекурсия.* Все, что можно запрограммировать при помощи простого цикла, как правило, так и следует программировать. Но в некоторых ситуациях рекурсия оказывается естественной и понятной. Примером может служить программа вычисления факториала:

```
function fact2(i:integer):int64; // рекурсивная функция
var mult:int64;
begin
if i=0 then fact2:=1;
if i>0 then begin
```

```
  if i=1 then mult:=1 else mult:=fact2(i-1);
  fact2:=mult*i
          end
  end{fact2};
function fact2_1(i:integer):int64; // вычисление факториала
//перемножением последовательности чисел
var mult:int64;
   j:integer;
begin
if i=0 then fact2_1:=1;
if i>0 then begin
fact2_1:=1;
for j:=1 to i do
fact2_1:=fact2_1*j
        end
end{fact2};
```

7. *Содержательные обозначения.* Может показаться, что краткие имена ускоряют и упрощают написание программы. Но, когда через некоторое время программу потребуется модифицировать, простота понимания, обеспечиваемая содержательными обозначениями, полностью окупит затраченные при программировании усилия.

Очевидно, что к приведенным семи правилам можно добавить некоторые другие. Но, если придерживаться перечисленных правил, то можно писать программы, которые легко понимать и сопровождать.

1.3. Некоторые понятия об объектно-ориентированной методологии программирования

Следующим этапом развития науки программирования стало создание объектно-ориентированной методологии.

Объектно-ориентированная методология программирования преследует те же цели, что и структурная, но решает их с другой отправной точки. По определению Г. Буча [1], «объектно-ориентированное программирование – это методология программирования, которая основана на представлении программы в виде совокупно-

сти объектов, каждый из которых является реализацией определенного класса (типа), а классы (типы) образуют иерархию на принципах наследуемости».

Один из принципов управления сложностью проекта – декомпозиция. Г. Буч выделяет две разновидности декомпозиции: алгоритмическую, которую поддерживают структурные методы, и объектно-ориентированную, отличие которой состоит в следующем: «Разделение по алгоритмам концентрирует внимание на порядке происходящих событий, а разделение по объектам придает особое значение факторам, либо вызывающим действия, либо являющимся объектами приложения этих действий». Другими словами, *алгоритмическая декомпозиция учитывает в бóльшей степени структуру взаимосвязей между частями сложной проблемы, а объектно-ориентированная уделяет бóльшее внимание характеру взаимодействий.*

На практике следует использовать обе разновидности. При создании крупных проектов сначала следует применить объектно-ориентированный подход для конструирования общей иерархии объектов, отражающих сущность программной задачи, а затем использовать алгоритмическую декомпозицию на модули для упрощения разработки, отладки и сопровождения программного комплекса.

Следует подчеркнуть [3], что в некоторых областях, таких как интерактивная графика, объектно-ориентированное программирование весьма полезно. В других задачах, таких как классические арифметические типы и вычисления, основанные на них, похоже трудно найти применение чему-то большему, чем абстракция данных, а средства, необходимые для поддержки объектно-ориентированного программирования, выглядят бесполезными.

2. ОСНОВЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ FREE PASCAL

В настоящее время существуют среды разработки программ, которые распространяются свободно (бесплатно). Для некоторых из них требуется регистрация, другие после закачки с соответствующего сайта сразу могут быть установлены на компьютер. Отметим интересующий нас источник: Free Pascal (www.freepascal.ru или http://lazarus.freepascal.org).

Приведем некоторые начальные сведения о программировании на языке Pascal, являющемся основой Free Pascal. Особенности среды разработки здесь затрагивать не будем. Но изучение нижеизложенного позволяет приступить к написанию своих собственных, вначале несложных, приложений.

2.1. Словарь языка Pascal

В любом языке программирования имеются символы и слова, которые компьютер понимает изначально. **Эти значки и слова нельзя употреблять как имена переменных**. Имена переменных часто называют идентификаторами. **Идентификатор –** это последовательность букв и цифр, начинающаяся с буквы.

З а м е ч а н и е. Пробелы в идентификаторах не допускаются!

Ограничители и специальные символы

+	:=	:	<	(	..
-	.	‘	<=	)	{
*	,	=	>	[	}
/	;	<>	>=	]	^

Ключевые слова

Absolute	And	Array	Asm
Begin	Case	Const	Constructor
Destructor	Div	Do	Downto
Else	End	External	File
For	Forward	Function	Goto
If	Implementation	In	Inline
Interface	Interrupt	Label	Mod
Nil	Not	Object	Of
Or	Packed	Procedure	Program
Record	Repeat	Set	Shl
Shr	String	Then	To
Type	Unit	Until	Uses
Var	Virtual	While	with
Xor			

2.2. Данные

Всякая программа предназначена для обработки данных различной природы (числа, тексты, последовательности двоичных разрядов или битов и т.д.). В зависимости от способа их хранения и обработки, данные можно разбить на две группы: *константы* и *переменные*.

Константы – это данные, которые не изменяются в процессе работы программы. *Переменные* – это данные, которые могут изменяться во время выполнения программы.

В Pascal константы могут быть трех видов: числовые, булевские (логические) и символьные.

Описание констант:

Const a=5; d=’ABC’; L=True;

Каждая переменная должна быть объявлена до своего первого применения. Тип переменной определяет множество допустимых для нее значений. Приведем некоторые стандартные типы данных:

Тип данных	Диапазон
Integer	-32768…32767
Int64	$-2^{63}…+2^{63}-1$ целое число со знаком
Byte	0…255
Word	0…65535
Real 48	±2.9E-39… ±1.7E+38
Double	±5.0E-324… ±1.7E+308
Boolean	False, True
Char	Множество символов ASCII
String	Строка символов (до 255 символов)

Имеются и другие типы данных.

Пример описания переменных:

```
Var
A,B:real;
S:string;
S1,S2:string;
I,j,k:integer;
AA, BB: double;
```

2.3. Структура программы

Программы, написанные на языке программирования Pascal и в основанных на нем средах разработки, имеют четкую структуру – состоят из заголовка и блока. Заголовок – это ключевое слово **Program** и имя программы, после которых ставится точка с запятой. Например:

Program P;

З а м е ч а н и е. Заголовок программы – неисполняемый оператор, то есть рассматривается компьютером как комментарий.

Блок программы состоит из шести разделов:

1) раздел меток (**Label**);
2) раздел констант (**Const**);
3) раздел типов (**Type**);
4) раздел переменных (**Var**);
5) раздел процедур и функций (**Procedure**, **Function**);
6) раздел действий.

З а м е ч а н и е. Раздел действий должен присутствовать всегда, остальные могут отсутствовать.

Первые четыре раздела начинаются с соответствующих ключевых слов (**Label, Const, Type, Var**).

Раздел меток (Label). Любой выполняемый оператор может быть снабжен меткой – целочисленным числом от 0 до 9999 или идентификатором. Каждая описанная метка должна появиться в программе, причем ровно один раз. Метка отделяется от оператора двоеточием.

Пример:

```
Label L1, 25;
……
  L1: log:=true;
  25: a:=b;
……
```

Раздел констант (Const). Если в программе используются константы (не обязательно числа), то удобно их обозначить каким-либо именем и далее обращаться к этой постоянной величине по имени. Отметим, что при построении выражения для определения значения констант можно использовать только ранее описанные константы, соединенные знаками операций и функциями (Abs, Chr,

Hi, Length, Odd, Ord, Round, Trunc и некоторые другие). Имеются предопределенные константы

Константа	Тип
Maxint=32767	**Integer**
MaxLongInt=2147483647	**LongInt**
False, True	**Boolean**
Pi=3.1415926	**Real**

Раздел типов (Type). Если в программе вводится тип, отличный от стандартного, то этот тип описывается в разделе **Type**. Синтаксис этого действия состоит из ключевого слова **Type**, идентификаторов определяемых типов, значка равно «=» и описания вида типа:

Type T=<вид типа>;
TT=<вид типа>;

Пример:
Type Index=1..20;
A=array[index] of real;

Раздел переменных (Var). Каждая переменная, встречающаяся в программе, должна быть описана до ее использования и отнесена к одному и только одному типу.

Пример:
Var r1, r2:real;
I,j,k:integer;
B:**array**[1..25] **of** real;

Область действия переменных определяется по правилу: **переменные локальны в блоке, где описаны, а также во всех вложенных блоках, если в них они не описаны повторно**.

Раздел действий. Эта часть программы начинается с ключевого слова **Begin** и заканчивается словом **End**, после которого ставится точка. Раздел действий – это выполняемая часть программы, состоящая из операторов.

2.4. Операторы

В языке программирования Pascal под операторами подразумеваются только действия. Операторы отделяются друг от друга точ-

кой с запятой. Точку с запятой можно не ставить перед словами **End** и **Until**, поскольку такая запись будет означать наличие пустого оператора между этой точкой с запятой и указанным служебным словом. Категорически нельзя ставить точку с запятой перед словом **Else**, поскольку между **Then** и **Else** должен стоять ровно один оператор, а не два и более.

Оператор присваивания. Пусть V – переменная, A – выражение. Тогда операция присваивания записывается в виде

V:=A;

Выражение A может содержать константы, переменные, знаки операций, функции, скобки. Любое выражение в скобках вычисляется раньше, чем операция, предшествующая скобкам.

З а м е ч а н и я 1. Присваивание допускается для всех типов, за исключением типа файл.

2. Переменной типа **real** (**double**) можно присвоить выражение типа **integer**, но переменной имеющей тип **integer** присвоить выражение типа **real** нельзя.

Составной оператор начинается ключевым словом **Begin** и заканчивается словом **End**. Между этими словами (их еще называют операторными скобками) помещается последовательность операторов, которые выполняются в порядке следования.

З а м е ч а н и е. Нельзя извне составного оператора передавать управление внутрь его.

Пример:

```
Begin
  R:=5;
  B:=R+2*pi;
End;
```

Условный оператор IF указывает, какие действия выполняются при истинности или ложности некоторого условия. Синтаксис этого оператора следующий:

If <Условие> **Then** <Оператор 1> **Else**<Оператор 2>;

Если <Условие> принимает значение **True**, то выполняется <Оператор 1>, иначе (при наличии **Else**) <Оператор 2>, расположенный за ключевым словом **Else**.

З а м е ч а н и е. Если при выполнении условия или его нарушении должно выполняться более одного действия, то эти действия

заключают в операторные скобки **Begin** и **End** и <Оператор 1> и <Оператор 2> представляют собой составной оператор.

Пример:

```
If x>=0 Then begin a:=x*pi; B:=A*sqrt(x) end
        Else begin A:=0; B:=exp(A); Writeln('A=0') end;
```

Существует и короткий вариант условного оператора: без использования служебного слова **Else**.

Пример:

```
If s>1 Then s:=1;
```

Оператор перехода Goto. Его синтаксис следующий:

Goto <Метка**>**;

Этот оператор (его еще называют оператором безусловного перехода) передает управление оператору, непосредственно следующему за меткой.

Пример:

```
Program Enter_one;
Uses  SysUtils;
Label back;
Var a:integer;
Begin
back:writeln('Enter one');readln(a);
If a<>1 Then Begin writeln('Mistake');Goto back End;
writeln('Correctly');readln
End.
```

Операторы цикла. В языке программирования Pascal имеется три вида оператора цикла.

1. Оператор For.

For <переменная**>:= <**начальное значение**> To <**конечное значение**> Do <**оператор**>;**

Здесь переменная цикла увеличивается на единицу.

For <переменная**>:= <**начальное значение**> Downto <**конечное значение**> Do <**оператор**>;**

Здесь переменная цикла уменьшается на единицу.

Начальное и конечное значения имеют тип **Integer**. В теле цикла **For** его параметр меняться не должен. Циклы такого типа назы-

вают финитными, имея в виду то, что они всегда завершат свою работу и поэтому не могут служить причиной «зацикливания» программы.

Пример:

```
For i:=1 To 15 Do Begin
                    a:=a*I;
                    writeln(a:12)
                  End;
For i:=15 Downto 1 Do Begin
                    a:=I*sqrt(i);
                    writeln(a:12)
                  End;
```

2. Оператор While. Синтаксис этого оператора следующий:

While <Выражение> **Do** <Оператор>;

Выражение имеет тип **Boolean,** оператор выполняется пока значение <Выражения> остается истинным (True) (проверка истинности осуществляется перед выполнением тела цикла).

3. Оператор Repeat.

Общий вид этого оператора следующий:

Repeat

<группа выполняемых операторов>

Until <Выражение>;

Тело цикла (операторы находящиеся между ключевыми словами **Repeat** и **Until**) выполняется пока значение <Выражения> – False. Проверка истинности осуществляется после выполнения тела цикла. Поэтому один раз <группа выполняемых операторов> отрабатывает в любом случае.

2.5. Массивы

Базируясь на простых типах, в языке Pascal можно строить более сложные типы переменных. Одним из таких типов является массив – структура, состоящая из фиксированного числа компонентов одного типа. Синтаксис:

Var <идентификатор> : **array**[<тип индекса>,..,< тип индекса >] **of** <тип элемента>;

Пример. Описание двумерного массива с элементами целого типа:

Var M : **array**[1..3, 1..4] **of integer**;

В этом примере первый индекс (от 1 до 3) – номер строки, второй индекс (от 1 до 4) – номер столбца:

$$M = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix}.$$

Обращаться к элементу массива следует так: A:=M[i,j], где $i \in [1..3]$, $j \in [1..4]$. Переменной A присвоено значение M[i,j]. Если записать M[i,j]:=A, то элементу массива M[i,j] присвоено значение переменной A.

2.6. Функции и процедуры

Очень часто, почти всегда, программу следует разбивать на несколько блоков, где каждый блок может восприниматься как отдельная программа. Такое разбиение позволяет придать всей программе понятную структуру. В языке программирования Free Pascal (Delphi) такое разбиение осуществляется с помощью процедур и функций.

2.6.1. Процедуры

Процедура в рассматриваемом нами языке программирования имеет ту же структуру, что и главная программа (**Program**): разделы **Label, Const, Type, Var** и выполняемую часть от **Begin** до **End.** После слова **End** ставится точка с запятой в отличие от основной программы, где ставится точка. Процедура помещается в главной программе, после раздела описания переменных **Var,** перед **Begin** главной программы.

В а ж н о е з а м е ч а н и е. Переменные, описанные внутри процедуры, действуют только в самой процедуре и называются локальными переменными. Если программа содержит много переменных и много процедур, то может случиться, что локальная и глобальная переменные будут иметь один и тот же идентификатор (пе-

ременные описаны в главной программе и процедуре). При вызове процедуры глобальный параметр временно «забывается». Когда же процедура выполнится, то глобальная переменная принимает снова первоначальное значение (значение, которое она имела перед выполнением процедуры). В результате никакой путаницы не происходит.

Следующее важное понятие в описании процедур – это формальные параметры. Под формальными параметрами понимаются переменные, через которые передаются данные из программы в процедуру либо из процедуры в программу. В качестве примера опишем программу, вычисляющую значение функции $y = a^x$ по формуле $y = e^{x \ln a}$.

Пример:

```
program degree;
uses
  SysUtils;
var p,q,z:real;
    fdeg:text;
procedure deg(a,x:real; var y:real);
   begin
     y:=exp(x*ln(a));
   end;

begin
  assignfile(fdeg,'fdeg.txt'); rewrite(fdeg);
  read(p,q);
  deg(p,q,z);
  write(fdeg,'возводим число ',p,' в степень ',q, ' получаем ',z:12);
  close(fdeg)
end.
```

В этой программе вызов процедуры производится оператором deg(p,q,z), где deg – имя процедуры; p, q, z – фактические параметры. При этом устанавливается взаимно-однозначное соответствие между фактическими и формальными параметрами, после чего управление передается процедуре. Когда процедура закончит свою

работу, управление передается оператору, который следует за вызовом процедуры (в нашем случае – оператору write).

Следует подчеркнуть:

1) число фактических параметров должно быть равно числу формальных параметров;

2) соответствующие фактические и формальные параметры должны совпадать по порядку и по типу.

З а м е ч а н и е. Если имеется запись

```
Var A:array [1..25] of integer;
    B:array [1..25] of integer;
```

то A и B считаются переменными разных типов. Поэтому при использовании массива в качестве параметра процедуры, для обеспечения совместимости, следует сначала ввести тип нужной структуры.

Пример:

```
Type MM=array [1..25] of integer;
Var A:MM;
Procedure P(B:MM); // Описание процедуры P
Begin
{Текст процедуры}
…
End;
Begin
{Текст основной программы}
…
P(A); // Вызов процедуры P
…
End;
```

Параметры процедур могут быть параметрами-значениями и параметрами-переменными.

Если в качестве формального параметра указана переменная (*только параметр и его тип*), то такой параметр называется ***параметром-значением***.

Для параметров-значений машина при вызове процедур выделяет место в памяти для каждого формального параметра, вычисля-

ет значение фактического параметра и засылает его в ячейку, соответствующую формальному параметру. Если фактический параметр есть имя переменной, то значение этой переменной пересылается в ячейку, соответствующую формальному параметру, и далее эти параметры никак не связаны, т.е. в памяти компьютера эти параметры занимают разные ячейки. Параметр-значение может быть константой, переменной, выражением.

Если перед именем формального параметра стоит ключевое слово **Var,** то такой параметр есть ***параметр-переменная***. В вышеприведенном примере это *y*:

procedure deg(a,x:**real**; **var** y:**real**);

Фактический параметр, соответствующий параметру-переменной, может быть только переменной (не константой, не выражением). Для параметра-переменной используется именно та ячейка, которая содержит соответствующий фактический параметр.

З а м е ч а н и е. Под формальные и фактические параметры-значения Pascal отводит разные области памяти. Поэтому результат выполнения процедуры может быть передан только через параметры-переменные или глобальные переменные (не рекомендуется).

2.6.2. Функции

Функция может аналогично процедуре восприниматься как отдельная подпрограмма. Но результат выполнения функции – это всегда определенное значение.

Описание функции начинается со слова **function**, затем идет имя описываемой функции, далее в скобках параметры функции по тем же правилам, что и для процедуры, после закрывающей скобки ставится двоеточие и пишется тип результата функции. Вышеописанный пример может быть переписан так:

```
program degree;
var p,q,z:real;
   fdeg:text;
function deg1(a,x:real):real;
  begin
    deg1:=exp(x*ln(a));
  end;
```

```
begin
  assignfile(fdeg,'fdeg.txt'); rewrite(fdeg);
  read(p,q);
  write(fdeg,'возводим число ',p,' в степень ',q, ' получаем ',deg1(p,q):15);
  close(fdeg)
end.
```

После заголовка, так же как и у процедур, описываются метки, постоянные, типы, переменные и так далее. Затем между словами **begin** и **end** следуют действия, которые вычисляют функцию. В блоке функции обязательно должен присутствовать оператор присваивания имени функции значения:

deg1:=exp(x*ln(a))

Сходство между процедурой и функцией заключается в том, что в обоих случаях, используется блок-структура, при которой могут быть объявлены новые переменные, метки и т.д.

Отличие состоит в том, что значение у функции всегда должно вычисляться. Тип этого значения должен объявляться в заголовке. Кроме того, у функции, по крайней мере, один раз должно встречаться имя функции, стоящее слева от оператора присваивания и выражения, из которого находится конечное значение.

З а м е ч а н и е. В качестве параметра процедуры или функции может быть использована другая процедура или функция.

Приведем программу, вычисляющую определенный интеграл методом средних прямоугольников:

```
program integral;
uses SysUtils;
type ff=function(x:real):real;//описываем тип функции
{описание функции, которая будет фактическим параметром}
function f(x:real):real;
begin f:=sqr(x) end;
{описание функции, в которой параметром является функция}
function integr(a,b:real;g:ff;n:integer):real;
var s,d,d2:real; i:integer;
begin d:=(b-a)/n; d2:=d/2; s:=0;
```

```
for i:=1 to n do s:=s+g(a+(2*i-1)*d2);
integr:=s*d end;
begin
{вызов функции с параметром-функцией}
writeln(integr(0,1,f,100)); readln
end.
```

З а м е ч а н и е. У функций, так же как и у процедур, в качестве формальных параметров могут быть параметры-переменные.

Текст программы, заключенный между фигурными скобками, не воспринимается компьютером и там можно писать все, что угодно на любом языке. Также не воспринимается текст в правой части строки после двух слеш. В ***комментариях*** рекомендуется указывать назначение процедур, функций, исполняемых операторов, назначение идентификаторов. Кроме того, сами идентификаторы используемых констант, переменных и меток могут быть словами, что-то говорящими программисту, и таким образом быть неявными комментариями. При сложной структуре программы, когда одни блоки вложены в другие, разумно начало и конец каждого из блоков помечать одинаковым комментарием и тогда будет видно, какой именно **End** соответствует какому **Begin**.

3. РЕАЛИЗАЦИЯ НЕКОТОРЫХ МАТЕМАТИЧЕСКИХ АЛГОРИТМОВ СРЕДСТВАМИ FREE PASCAL

3.1. Произведение матриц

Пусть даны две матрицы

$$A=\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \quad \text{и} \quad B=\begin{pmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{np} \end{pmatrix},$$

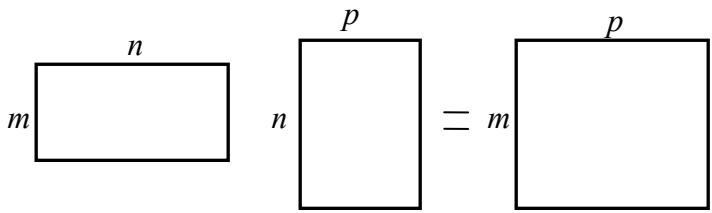
причем число столбцов матрицы A равно числу строк матрицы B. Произведением матрицы A на матрицу B называется матрица C,

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1p} \\ c_{21} & c_{22} & \dots & c_{2p} \\ \dots & \dots & \dots & \dots \\ c_{m1} & c_{m2} & \dots & c_{mp} \end{pmatrix},$$

обозначаемая через AB, элементы которой вычисляются по формулам

$$c_{ij} = \sum_{s=1}^{n} a_{is} b_{sj} \text{ , где } i = 1,2,\dots,m \text{ ; } \quad j = 1,2,\dots,p \text{ .}$$

Еще раз отметим, что произведение двух матриц имеет смысл только при условии, которое графически выглядит так:



Программа, перемножающая две матрицы представленные в файле, записывается в виде

```
Program multiplication_matrix;
var m,n,p:integer;
   A,B,C:array [1..20,1..20] of double;
   i,j,s:integer;
   ABC_MAT:text;
begin
//параметры и исходные матрицы читаем из файла, туда же
//записываем результат произведения. Считаем, что размерности
//файлов удовлетворяют правилу произведения матриц
assign(ABC_MAT,'c:\fpc\ABC_MAT.txt');  reset(ABC_MAT);
//читаем параметры матриц
read(ABC_MAT,m,n,p);
//читаем матрицу A
for i:=1 to m do
```

```
  for j:=1 to n do read(ABC_MAT,A[i,j]);
//читаем матрицу B
for i:=1 to n do
  for j:=1 to p do read(ABC_MAT,B[i,j]);
//находим произведение матриц – матрицу C
for i:=1 to m do
  for j:=1 to p do begin
    C[i,j]:=0;
    for s:=1 to n do
      C[i,j]:=C[i,j]+A[i,s]*B[s,j]
                end;
//выводим результат в тот же файл
close(ABC_MAT);
append(ABC_MAT);
writeln(ABC_MAT);
writeln(ABC_MAT,'result');
for i:=1 to m do begin
  for j:=1 to p do write(ABC_MAT,C[i,j]:12,' ');
  writeln(ABC_MAT)
              end;
close(ABC_MAT)
end.
```

3.2. Вычисление определителя квадратной матрицы

Во многих математических дисциплинах используется понятие определителя квадратной матрицы n-го порядка, под которым понимается *сумма всех* $n!$ *произведений элементов этой матрицы, взятых по одному из каждой строчки и по одному из каждого столбца; при этом каждое произведение снабжено знаком плюс или минус по некоторому правилу*.

Произведения, о которых говорится в определении, можно представить в виде

$$P = a_{1\alpha} a_{2\beta} \ldots a_{n\omega}. \qquad (3.1)$$

Первый индекс у сомножителя a_{ij} в произведении (3.1) соответствует номеру строки, второй – номеру столбца. Номера столбцов при упорядоченности по номерам строк дают перестановку $(\alpha,\beta,\ldots,\omega)$. Если эта перестановка четная, то величина P, входящая в определение определителя, берется со знаком плюс, если нечетная – то со знаком минус.

З а м е ч а н и е. Непосредственное вычисление определителя по определению представляет собой трудоемкий процесс, так как нужно находить $n!$ произведений и выбрать знак каждого из них. Но можно заметить, что если все элементы, стоящие выше или ниже главной диагонали равны нулю, то определитель равен произведению элементов, стоящих на этой диагонали: $a_{11}a_{22}\ldots a_{nn}$.

Полезными при вычислении определителей оказываются следующие их свойства:

- определитель матрицы n-го порядка не изменится, если к элементам одной ее строки прибавить соответствующие элементы другой строчки этой же матрицы, умноженной на одно и то же произвольное число;
- если все элементы какой-нибудь строчки матрицы n-го порядка умножить на число c, то ее определитель также умножится на число c;
- перестановка двух строк в матрице определителя приводит к изменению знака определителя.

Из сделанного замечания и перечисленных свойств следует, что матрицу определителя целесообразно привести к треугольному виду и найти произведение элементов, стоящих на главной диагонали. При этом перед процедурой диагонализации следует матрицу определителя привести к виду, при котором на главной диагонали будут стоять ненулевые элементы (напомним, что перестановка строк меняет знак определителя).

Программа, реализующая сказанное, имеет следующую запись:

```
Program Determinant;
var n,i,j,k,kk:integer;
   DetA,dd,p:double;
   A,B:array [1..20,1..20] of double;
   F_det:text;
```

```
begin
assignfile(F_det,'F_det.txt'); reset(F_det);
//input n - dimension of determinant
read(F_det,n);
//read matrix
for i:=1 to n do
  for j:=1 to n do begin read(F_det,A[i,j])
                    end;

  closefile(F_det);  append(F_det); writeln(f_det);
  writeln(f_det,n);  writeln(f_det);

//Проверка введенной матрицы
  for i:=1 to n do begin
  for j:=1 to n do  write(F_det,A[i,j]);
    writeln(f_det)
                end;

dd:=1;
//установка на главной диагонали ненулевых элементов
//При этом учитывается знак при перестановке строк в опреде-
//лителе, вначале dd=1, а при перестановке dd меняет знак
for i:=1 to n-1 do
  if (A[i,i]=0) then begin
    j:=i+1;
    while((A[j,i]=0) and (j<n)) do j:=j+1;
    //перестановка строк i, j,если A[j,i]<>0
      if A[j,i]<>0 then begin dd:=dd*(-1);
      for k:=1 to n do begin
          p:=A[i,k]; A[i,k]:=A[j,k]; A[j,k]:=p
                      end;
                      end;
                    end;
//матрица после перестановки строк
for i:=1 to n do begin
  for j:=1 to n do write(F_det,A[i,j]:12,' ');
  writeln(F_det);
            end;

//Приведение матрицы определителя к треугольному виду
```

```
//Учитываем, что при умножении строки на число определитель
//увеличивается в такое же число раз – переменная dd предназ-
//начена для учета этого изменения
for k:=1 to n-1 do begin
 for i:=1 to k do
   for j:=1 to n do B[i,j]:=A[i,j];

  for i:=k+1 to n do begin
   for j:=k to n do
     B[i,j]:=-(A[k,j]*A[i,k]-A[i,j]*A[k,k]);
     dd:=dd*A[k,k];
                  end;
  for i:=1 to n do
   for j:=1 to n do A[i,j]:=B[i,j];

//В процессе приведения определителя к треугольному виду
//какой-либо элемент на главной диагонали может оказаться
//равным нулю. Поэтому проверяем эту ситуацию и меняем
//местами строки как и ранее, но не трогаем строки с номерами
//меньше k
  for i:=k to n-1 do
  if (A[i,i]=0) then begin
   j:=i+1;
   while((A[j,i]=0) and (j<n)) do j:=j+1;
   //перестановка строк i, j,если A[j,i]<>0
     if A[j,i]<>0 then begin dd:=dd*(-1);
     for kk:=1 to n do begin
         p:=A[i,kk]; A[i,kk]:=A[j,kk]; A[j,kk]:=p
                  end;
                   end;
                 end;
               end{k};
//Вывод треугольной матрицы определителя
writeln(F_det);
writeln(F_det,'triangl-matrix');  writeln(F_det);

for i:=1 to n do begin
  for j:=1 to n do write(F_det,B[i,j]:12,' ');
  writeln(F_det);
```

```
                    end;

  //calculate of determinant
  DetA:=1;
  for k:=1 to n do DetA:=DetA*B[k,k];
  writeln(F_det);

  if DetA=0 then writeln(F_det,'DetA=0')
       else begin
          DetA:=DetA/dd;
          writeln(F_det,'DetA= ',DetA:12)
          end;

  closefile(F_det);
  end.
```

3.3. Решение системы линейных алгебраических уравнений методом Гаусса

Пусть дана система линейных алгебраических уравнений, которая в матричной форме имеет вид

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}. \qquad (3.2)$$

Составим расширенную матрицу

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix}. \qquad (3.3)$$

Определение. Под элементарными преобразованиями системы алгебраических уравнений понимают следующие операции:

1) умножение какого-либо уравнения системы на число, отличное от нуля;

2) прибавление к одному уравнению другого уравнения этой же системы, умноженного на произвольное число;

3) перемену местами двух уравнений в системе.

В курсе алгебры доказывается теорема: при элементарных преобразованиях система переходит в равносильную (две системы линейных уравнений называются равносильными, если каждое решение одной из них является решением другой и наоборот).

Метод Гаусса решения системы линейных уравнений состоит в том, что при помощи элементарных преобразований систему приводят к такому виду, чтобы ее расширенная матрица из коэффициентов оказалась трапециевидной. После чего решение системы становится значительно проще.

Программа, решающая систему линейных алгебраических уравнений, записывается так:

```
program Gauss;
 {Программа Lazarus}
{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes
  { you can add units after this };

Var A,B:array [1..20, 1..21] of Double;
   X:array[1 .. 20] of Double;
     N,i,j,k,k1,n1:Integer;
     s,r:double;
     fg:text;
Begin
assignfile(fg,'fg.txt'); reset(fg);

{читаем число уравнений из файла fg.txt}
read(fg,N);
n1:= N + 1;
```

```
{читаем расширенную матрицу из файла fg.txt}
For i:= 1 To N do
  For j:=1 To n1 do
    read(fg,A[i, j]);

B:=A; {матрица B потребуется для проверки решения}

For k:=1 To N do begin
  k1:=k + 1;
  s:= A[k, k];
  j:=k;
  For i:=k1 To N do begin
    r:=A[i, k];
    If Abs(r) > Abs(s) Then begin s:=r;  j:=i end;
                end{i};
  If s = 0 Then writeln(fg,'det=0 Единственность решения отсут-
ствует');

    For i:= k To n1 do begin
      r:=A[k, i];  A[k, i]:=A[j, i]; A[j, i]:=r
    end{ i};

For j:= k1 To n1 do begin
     A[k, j]:=A[k, j] / s
   end{ j};
   For i:= k1 To N  do begin
     r:=A[i, k];
     For j:=k1 To n1 do
       A[i, j]:=A[i, j] - A[k, j] * r

   end{ i}
            end{k};
For i:=N downTo 1 do begin
  s:=A[i, n1];
  For j:= i + 1 To N do s:= s - A[i, j] * X[j];
X[i]:=s
end{ i};
// дописываем в файл fg решение системы
 append(fg);  writeln(fg);
For i:=1 To N do
```

```
 writeln(fg,X[i]); writeln(fg);
//Проверка решения: в конце файла fg.txt должны получить
//столбец из нулей
  For i:=1 To N do begin
    s:=0;
    for j:=1 to N do begin
       s:=s+B[i,j]*X[j];
                end;
       writeln(fg,s-B[i,N+1])
                end;

 close(fg);
End.
```

3.4. Сортировка

Задача сортировки множества формулируется следующим образом [6]: задано конечное множество A, состоящее из n элементов, а на нем задано отношение линейного порядка P. Требуется перенумеровать элементы A числами от 1 до n таким образом, чтобы из неравенства $i < j$ следовало $(a_i, a_j) \in P$.

Обычно задача сортировки сводится к упорядочению чисел по значению.

Сортировка вставкой – один из простейших способов сортировки, состоящей в следующем:

1) последовательность из одного элемента множества уже «отсортирована»;

2) берем следующий элемент и сравниваем его с предыдущим, при этом переставляем его вперед до тех пор, пока он не займет свое место.

```
Program Inset;
{$mode objfpc}{$H+}

Uses   crt,  fileutil;
var i,j,k,N:integer;
   b:double;
   A:array[1..20] of double;
begin
```

```
//вводим длину массива N
read(N);
//читаем массив
for i:=1 to n do read (A[i]);
for i:=2 to N do begin
b:=A[i]; j:=1;
//Определяем место вставки элемента;
//начинаем поиск с первого элемента массива до i-го
while b>A[j] do
  j:=j+1;
  for k:=i-1 downto j do
    A[k+1]:=A[k];
    A[j]:=b
              end{i};

writeln;
for i:=1 to N do write(A[i]:12,' ');
readkey;
end.
```

Принцип ***сортировки выбором***:

1) выбираем в массиве элемент с минимальным значением на интервале от 1-го до *n*-го элемента и меняем его местами с первым элементом;

2) находим элемент с минимальным значением на интервале от 2-го до *n*-го элемента и меняем его местами со вторым элементом и так далее до (*n*-1)-го элемента.

```
Program sort2;
uses crt;
var i,N,imin,s:integer;
   min:double;
   A:array [1..20] of double;
begin
//вводим количество элементов в массиве  N
read(N);
//вводим массив чисел
for i:=1 to N do read(A[i]);
for s:=1 to N do begin
//находим минимальный элемент и его номер среди элементов
```

```
//с номерами от s до N
  min:=A[s];
  imin:=s;
  for i:=s+1 to N do
    if A[i]<min then begin
      min:=A[i]; imin:=i
                  end{i};
//минимальный элемент из A[s]…A[N] ставим на место
//элемента с номером s, а на его место A[s]
  A[imin]:=A[s]; A[s]:=min;
              end{s};
//Вывод отсортированного массива
for i:=1 to N do write(A[i]:12,' ');
readkey;
end.
```

Содержание ***сортировки обменом («пузырьковая» сортировка)***:

1) слева направо до конца массива поочередно сравниваются два соседних элемента, и если их взаиморасположение не соответствует условию упорядоченности, то они меняются местами;

2) после первого прохода на *n*-м месте стоит элемент с максимальным значением («всплыл» первый пузырек). Второй проход выполняется до (*n*-1)-го элемента, так как *n*-й уже на своем месте. Всего требуется *n*-1 проходов.

```
program bubble;
{$mode objfpc}{$H+}
uses crt;
var s,i,n:integer;
   a:array [1..20] of double;
   p:double;
{$IFDEF WINDOWS}{$R sort3_2.rc}{$ENDIF}
begin
//вводим число элементов в массиве
read(n);
//вводим массив
for i:=1 to n do read(a[i]);

for s:=1 to n-1 do
```

```
  for i:=1 to n-s do
//сравнение двух соседних элементов. Если (i+1)-й элемент
//меньше i-го, то элементы массива меняются местами
   if a[i]>a[i+1] then begin
     p:=a[i]; a[i]:=a[i+1]; a[i+1]:=p
                  end ;
//Вывод отсортированного массива
writeln;
for i:=1 to n do write(a[i]:12);
readkey
end.
```

Метод фон Неймана. Пусть даны два упорядоченных массива. Возьмем из них первые элементы и выберем тот, который должен следовать вначале в соответствии с некоторым условием, и запишем его в массив результата. Изменим рассмотрение индексов в массиве, из которого взят элемент. Применим изложенную процедуру к оставшимся массивам. Если один из массивов исчерпан, то остаток другого просто дописывается в результирующий массив. Описанная процедура называется *слиянием массивов*.

Дж. фон Нейман предложил метод сортировки, основанный на операции слияния:

1) исходный массив из n элементов вначале представляется в виде n «отсортированных» массивов, имеющих длину, равную единице;

2) сольем эти массивы попарно, что приведет к появлению массивов длины два;

3) повторим процедуру слияния, получая массивы длиной четыре и так далее. В итоге получается отсортированный массив длиной n.

З а м е ч а н и е. Недостатком этого метода является то, что для почти отсортированного массива тратится столько же времени, как и на самый «перемешанный» массив.

3.5. Рекурсия

Рекурсивным называется объект, который частично определяется через самого себя. В программировании рекурсивная функция

или процедура – это функция или процедура, которая вызывает сама себя.

Классический пример – определение функции факториала. С одной стороны, факториал определяется следующей формулой:

$$n! = 1 \cdot 2 \cdot \ldots \cdot n,$$

с другой – рекуррентными соотношениями

I. $0! = 1$;

II. для $\forall n > 0 \quad n! = n \cdot (n-1)!$

Другим примером может служить определение чисел Фибоначчи

$F(1) = 1$;

$F(2) = 1$;

для $\forall n > 2 \quad F(n) = F(n-1) + F(n-2)$.

Программы, включающие рекурсивные процедуры, наглядны и содержат компактный код. Однако использование рекурсивных процедур требует больше размера оперативной памяти при выполнении кода, чем нерекурсивных. Это вызвано тем, что при каждом рекурсивном вызове для параметров процедуры и локальных переменных выделяются новые ячейки памяти.

Глубиной рекурсии называется максимальное число рекурсивных вызовов, которое происходит во время выполнения программы. ***Текущим уровнем*** рекурсии – число рекурсивных вызовов в каждый конкретный момент времени.

3.5.1. Формы рекурсивных процедур

Любая рекурсивная процедура включает в себя некоторое множество операторов S и один или несколько операторов рекурсивного вызова P. Следует особо подчеркнуть, что безусловные рекурсивные процедуры приводят к бесконечным процессам. Еще раз напомним, что каждая копия рекурсивной процедуры требует выделения дополнительной памяти, но память в любом устройстве конечна!

Таким образом, вызов рекурсивной процедуры должен выполняться по условию, которое на каком-то уровне рекурсии станет ложным.

Если условие истинно, то рекурсивный спуск продолжается. В момент принятия условием ложного значения спуск прекращается и начинается поочередный рекурсивный возврат из всех вызванных на данный момент копий рекурсивной процедуры.

Форма с выполнением действий до рекурсивного вызова (действия выполняются на рекурсивном спуске)

```
Procedure P(…);
begin
  S
  if <условие> then P(…)
end;
```

Форма с выполнением действий после рекурсивного вызова (действия выполняются на рекурсивном возврате)

```
Procedure P(…);
begin
  if <условие> then P(…)
  S
end;
```

Форма с выполнением действий как до, так и после рекурсивного вызова

```
Procedure P(…);
begin
  S1
  if <условие> then P(…)
  S2
end;
```

или

```
Procedure P(…);
begin
  if <условие> then
          S1
          P(…)
          S2
end;
```

Многие задачи безразличны к выбору формы рекурсивной процедуры. Но существуют целые классы задач, при решении которых требуется сознательно управлять ходом работы рекурсивных процедур и функций. К таким задачам, в частности, относится обработка списков и древовидных структур данных.

Выполнение действий на рекурсивном спуске. Для реализации алгоритма вычисления факториала, работающего на спуске, вводим параметры: *mult* для выполнения на спуске операции умножения накапливаемого значения факториала на очередной множитель, *m* для обеспечения независимости рекурсивной функции от имени конкретной глобальной переменной.

```
program fact1;
{$mode objfpc}{$H+}
Uses crt;
 var N:integer;
function fact1(mult:longint; i,m:longint):longint;
begin
if m=0 then fact1:=1
     else begin
mult:=mult*i;
if i=m then fact1:=mult else fact1:=fact1(mult,i+1,m)
         end
end{fact1};
{$IFDEF WINDOWS}{$R fact1.rc}{$ENDIF}
begin
{ввод целого числа, от которого вычисляется факториал}
read(N);
writeln(fact1(1,1,N));
readkey
end.
```

Выполнение действий на рекурсивном возврате. Приведем программу, в которой вычисление факториала происходит на рекурсивном возврате, и при этом рекурсивный вызов и оператор накопления факториала разделены явным образом.

```
program fact2;
```

```
{$mode objfpc}{$H+}
uses crt,fileutil;
var n:integer;
function fact2(i:integer):int64;
var mult:int64;
begin
if i=0 then fact2:=1;
if i>0 then begin
if i=1 then mult:=1 else mult:=fact2(i-1);
fact2:=mult*i
            end
end{fact2};

{$IFDEF WINDOWS}{$R fact2.rc}{$ENDIF}

begin
{ввод целого числа, от которого вычисляется факториал}
writeln(utf8toconsole('введите целое число, от которого вычисляется факториал'));
read(n);
writeln;
writeln(n,'  ',fact2(n));
readkey;
end.
```

3.5.2. Быстрая сортировка (метод Quicksort)

Английский математик, профессор Оксфордского университета Хоар (Charles Antony Richard Hoare), предложил в 1962 г. метод, основанный на следующей идее: взять один из элементов массива и поставить его таким образом, что элементы, предшествующие ему в упорядочении, размещались до него, а следующие за ним – после. После этой операции исходный массив разбит на две части, которые можно упорядочить тем же способом.

Реализуем сказанное с помощью рекурсии [2]:

1. Выбираем центральный элемент массива и записываем его в переменную B. Затем элементы массива просматриваем поочередно слева направо и справа налево. При движении слева направо ищем элемент $A(i)$, который будет больше или равен B, и запоми-

наем его позицию. При движении справа налево ищем элемент $A(j)$, который меньше или равен B, и запоминаем его позицию. Найденные элементы меняем местами и продолжаем встречный поиск по указанным условиям. Этот процесс продолжаем, пока при очередной итерации поиска встречные индексы i и j не пересекутся. Таким образом, относительно значения B массив получается отсортированным. Остается упорядочить левую и правую части массива.

2. Описанную в предыдущем пункте процедуру повторяем для левой и правой частей по отдельности. После чего массив оказывается разбитым уже на четыре непересекающихся по сортировке части, которые можно упорядочить по отдельности. Процесс продолжается, пока длина сортируемых частей не станет равной одному элементу и, следовательно, все элементы массива упорядочены.

Так как на каждом этапе выполняемые действия одни и те же, но в разных индексных промежутках, то удобно воспользоваться рекурсией. Программа Quicksort имеет вид [2]

```
program Quicksort;

{$mode objfpc}{$H+}

uses crt,fileutil ;

const nn=25; //вводим постоянную для описания массива,
//который будем сортировать
var A:array [1..nn]of integer;
  i,n:integer;
procedure HSort(L,R:word);
  var b,tmp:integer;
    i,j:word;
begin
  b:=A[(l+r)div 2];
  i:=l; j:=r;
while i<=j do
begin
  while a[i]<b do i:=i+1;
  while a[j]>b do j:=j-1;
  if i<=j then begin
```

```
    tmp:=a[i];
    a[i]:=a[j];
    a[j]:=tmp;
    i:=i+1;
    j:=j-1
            end;
end;
if l<j then hsort(l,j);
if i<r then hsort(i,r);

end;

begin
  clrscr;
  writeln(utf8toconsole('вводим длину вектора для сортировки <=25'));
  read(n);
  writeln(utf8toconsole('ввод сортируемого вектора'));
  for i:=1 to n do read(a[i]);
  hsort(1,n);
  writeln;
  for i:=1 to n do write(a[i],' ');
  writeln;
  readkey
end.
```

4. РЕШЕНИЕ ЗАДАЧ МЕХАНИКИ

Рассмотрим задачи механики и некоторые решим численно, построим графики и проанализируем полученные результаты.

4.1. Движение точки по инерции под действием силы трения вдоль прямой

В соответствии со вторым законом Ньютона $m\mathbf{w} = \mathbf{F}$ и законом сухого трения Кулона $\mathrm{T} = -\mu N(\boldsymbol{v}/v)$, где T – вектор силы трения, μ – коэффициент трения, N – величина нормальной реакции, $\boldsymbol{v}$ – вектор скорости, v – величина скорости уравнение движения точки по горизонтальной плоскости будет иметь вид

$$m\mathbf{w} = -\mu N(\boldsymbol{v}/v).$$

Если предположить, что движение может происходить только вдоль некоторой прямой, то после очевидных преобразований получаем дифференциальное уравнение

$$\ddot{x} = -\mu g. \tag{4.1}$$

Аналитическое решение этого уравнения при начальных условиях $x(0) = 0$, $\dot{x}(0) = v_0$ имеет вид

$$x(t) = v_0 t - \frac{1}{2}\mu g t^2,\ v = v_0 - \mu g t. \tag{4.2}$$

Откуда следует, что скольжение будет продолжаться в течение $t_* = v_0/\mu g$, где t_* в секундах.

Получим численное решение уравнения (4.1) с помощью метода Рунге-Кутта 4-го порядка с постоянным шагом интегрирования. Реализация этого метода происходит по формулам [7]

$$k_1 = f(t_n, y_n), \qquad k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{hk_1}{2}\right),$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{hk_2}{2}\right),\ k_4 = f\left(t_n + h, y_n + hk_3\right), \tag{4.3}$$

$$y_{n+1} = y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right).$$

Следует подчеркнуть, что величины $k_1, k_2, k_3, k_4, f, y_n$ и y_{n+1} являются векторами, а h – скалярная величина, соответствующая шагу интегрирования по времени.

Для применения формул (4.3) уравнение (4.1) следует привести к нормальному виду

$$\begin{aligned} \dot{x} &= v \\ \dot{v} &= -\mu g \end{aligned} \qquad \text{или} \qquad \begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v \\ -\mu g \end{pmatrix}.$$

Программа, реализующая данный алгоритм, представлена ниже для системы с двумя степенями свободы. Отличие от программы, приведенной на с. 45…48 будет в процедуре вычисления правых частей системы дифференциальных уравнений.

После получения результатов расчета переведем данные в Excel и построим графики $v(t)$, $x(t)$ (рис. 4.1).

0	0	1
0,1	0,047638	0,901704
0,2	0,09027	0,803604
0,3	0,127998	0,705504
0,4	0,160821	0,607404
0,5	0,188738	0,509304
0,6	0,211751	0,411204
0,7	0,229859	0,313104
0,8	0,243062	0,215004
0,9	0,251359	0,116904
1	0,254752	0,018804

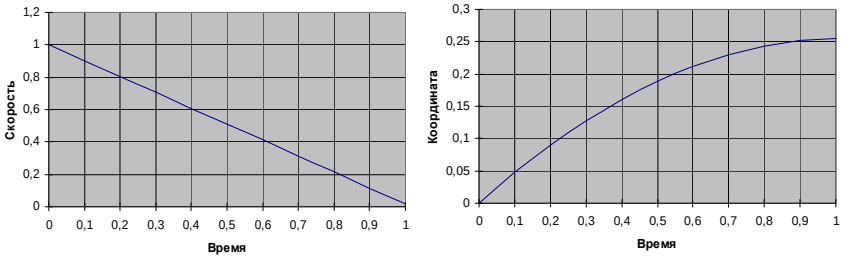


Рис. 4.1

4.2. Движение точки под действием восстанавливающей силы

Пусть точка массой m может перемещаться вдоль оси X. Предположим, что на точку действует восстанавливающая сила $-cx$. В соответствии с вышеизложенным $m\ddot{x} = -cx$ или $\ddot{x} + k^2 x = 0$, где $k^2 = c/m$. Дифференциальное уравнение движения следует привести к нормальному виду

$$\begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v \\ -cx \end{pmatrix}.$$

Численное решение данного уравнения для $m = 1$, $c = 0{,}5$ представлено на рис. 4.2 (начальные условия $x(0) = 0$, $\dot{x}(0) = 1$):

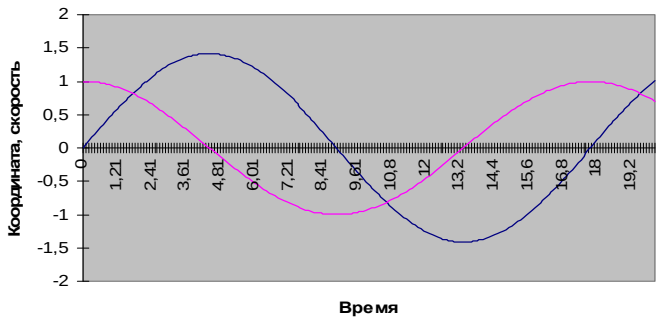


Рис. 4.2

4.3. Движение системы, состоящей из двух масс

Решим задачу № 48.46 из сборника задач по теоретической механики И.В. Мещерского [10]:

Определить движение системы, состоящей из двух масс m_1 и m_2, насаженных на гладкий горизонтальный стержень (ось Ox), (рис. 4.3), массы связаны пружиной с жесткостью c и могут двигаться поступательно вдоль стержня; расстояние между центрами масс при ненапряженной пружине равно l. Начальное состояние системы при $t = 0$ определяется следующими значениями скоростей и координат центров масс: $x_1 = 0$, $\dot{x}_1 = u_0$, $x_2 = l$, $\dot{x}_2 = 0$.

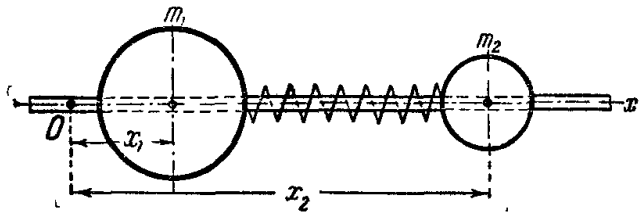


Рис. 4.3

Механическая система имеет две степени свободы. Пусть x_1 и x_2 – координаты центров первого и второго тела соответственно, m_1 и m_2 – их массы. Тогда кинетическая и потенциальная энергии системы записываются в виде

$$T = \frac{1}{2} m_1 \dot{x}_1^2 + \frac{1}{2} m_1 \dot{x}_2^2 ;$$
$$\Pi = \frac{1}{2} c\left(x_2 - x_1 - l\right)^2 .$$

Подставим эти выражение в уравнение Лагранжа второго рода

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{x}_i} - \frac{\partial T}{\partial x_i} = -\frac{\partial \Pi}{\partial x_i}, \quad i = 1,2 .$$

После дифференцирования получаем систему из двух уравнений второго порядка:

$$\begin{cases} m_1 \ddot{x}_1 = c\left(x_2 - x_1 - l\right); \\ m_2 \ddot{x}_2 = -c\left(x_2 - x_1 - l\right). \end{cases}$$

Откуда после деления на массы, стоящие в левых частях уравнений, получаем соотношения

$$\begin{cases} \ddot{x}_1 = \dfrac{c}{m_1}\left(x_2 - x_1 - l\right); \\ \ddot{x}_2 = -\dfrac{c}{m_2}\left(x_2 - x_1 - l\right). \end{cases} \qquad (4.4)$$

Система уравнений (4.4) может быть решена аналитически. Введем переменную $z = x_2 - x_1 - l$ и заметим, что $\ddot{z} = \ddot{x}_2 - \ddot{x}_1$, отни-

мем от второго уравнения системы (4.4) первое уравнение этой же системы:

$$\ddot{x}_2 - \ddot{x}_1 = -\left(\frac{c}{m_1} + \frac{c}{m_2}\right)\left(x_2 - x_1 - l\right)$$

или

$$\ddot{z} = -k^2 z\,,\ k^2 = \frac{c}{m_1} + \frac{c}{m_2}\,. \tag{4.5}$$

Решение уравнения (4.5):

$$z = C_1 \sin kt + C_2 \cos kt\ , \tag{4.6}$$

откуда

$$\dot{z} = C_1 k \cos kt - C_2 k \sin kt\,. \tag{4.7}$$

Начальные условия для переменной z имеют вид $z\big|_{t=0} = 0$ и $\dot{z}\big|_{t=0} = -u_0$. Подстановка этих условий в (4.6) и (4.7) дает возможность определить C_1 и C_2: $C_1 = -\dfrac{u_0}{k}$, $C_2 = 0$. В итоге

$$z = -\frac{u_0}{k}\sin kt\,. \tag{4.8}$$

Выражения (4.4) и (4.8) позволяют записать равенства

$$\ddot{x}_1 = \frac{c}{m_1} z = -\frac{c}{m_1}\frac{u_0}{k}\sin kt\,, \qquad \ddot{x}_2 = -\frac{c}{m_2} z = \frac{c}{m_2}\frac{u_0}{k}\sin kt\,,$$

которые интегрируются с начальными условиями $x_1\big|_{t=0} = 0$ и $\dot{x}_1\big|_{t=0} = u_0$, $x_2\big|_{t=0} = l$ и $\dot{x}_2\big|_{t=0} = 0$:

$$\begin{aligned} x_1 &= \frac{1}{m_1 + m_2}\left(\frac{m_2 u_0}{k}\sin kt + u_0 m_1 t\right); \\ x_2 &= l + \frac{1}{m_1 + m_2}\left(-\frac{m_1 u_0}{k}\sin kt + u_0 m_1 t\right). \end{aligned} \tag{4.9}$$

Решим систему уравнений (4.4) численным методом Рунге-Кутта четвертого порядка. Систему уравнений (4.4) приводим вначале к нормальному виду

$$\begin{aligned} \dot{x}_1 &= v_1; \\ \dot{v}_1 &= \frac{c}{m_1}(x_2 - x_1 - l); \\ \dot{x}_2 &= v_2 \ ; \\ \dot{v}_2 &= -\frac{c}{m_2}(x_2 - x_1 - l). \end{aligned} \tag{4.10}$$

Начальные условия принимают вид при $t = 0$ $x_1 = 0$, $\dot{x}_1 = u_0$, $x_2 = l$, $\dot{x}_2 = 0$.

Программа, реализующая метод Рунге-Кутта четвертого порядка для данной механической системы с начальными данными $t = 0$ $x_1 = 0$, $\dot{x}_1 = 1$, $x_2 = 1$, $\dot{x}_2 = 0$, массами тел $m_1 = 1$ кг, $m_2 = 0{,}5$ кг и жесткостью пружины $c = 200$ Н/м, имеет следующий вид:

```
program RungKutt4;

{$mode objfpc}{$H+}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes
  { you can add units after this };

const n=4;
 type vconst=array[1…10] of double;
 type vector=array [1…n] of double;
 var vp:vconst;
     Y:vector;
     frung:text;
     A,B,H,TH,t:DOUBLE;

(*{В процедуре PRAV VP-вектор постоянных, которые
используются при вычислении правых частей системы уравнений;
T-time;
```

```
Y-input vector begin value;
Z-output vector value} *)
procedure PRAV(vp:vconst; T:double; y:vector; var z:vector);
var m1,m2,c,l,x1,x2,vx1,vx2:double;
begin
m1:=vp[1]; m2:=vp[2]; c:=vp[3]; l:=vp[4];
x1:=y[1]; vx1:=y[2];
x2:=y[3]; vx2:=y[4];
z[1]:=vx1; z[2]:=(c/m1)*(x2-x1-l);
z[3]:=vx2; z[4]:=-(c/m2)*(x2-x1-l);
end;{prav}

PROCEDURE RK4( VP:vconst; H:DOUBLE; VAR T:DOUBLE;
          VAR Y:vector);
{Процедура, выполняющая один шаг метода Рунге-Кутта 4-го
порядка}
{VP-вектор постоянных, которые используются при
вычислении правых частей системы уравнений;
H-step of method;
T-begin value of time;
Y-begin value of vector;
 Перед процедурой RK4 описать процедуру правых частей
   PRAV(VP,T,Y,YY)}
VAR I,H_index_Y,L_index_Y:INTEGER;
   K,K1,K2,K3,K4:vector;
BEGIN
L_index_Y:=low(Y); H_index_Y:=high(Y);
PRAV(VP,T,Y,K1);
FOR I:=L_index_Y TO H_index_Y DO K[I]:=Y[I]+K1[I]*H/2;
T:=T+H/2; PRAV(VP,T,K,K2);
FOR I:=L_index_Y TO H_index_Y DO K[I]:=Y[I]+K2[I]*H/2;
PRAV(VP,T,K,K3);
FOR I:=L_index_Y TO H_index_Y DO K[I]:=Y[I]+K3[I]*H;
T:=T+H/2; PRAV(VP,T,K,K4);
FOR I:=L_index_Y TO H_index_Y DO Y[I]:=Y[I]+ H/6*(K1[I]+
+2*(K2[I]+K3[I])+K4[I])
END; {RK4}

PROCEDURE RUNG(N:INTEGER; A,B,H,TH:DOUBLE;
```

```
                VAR T:DOUBLE;
                VAR Y:VECTOR);
    {Процедура, вызывающая RK4 и выводящая результаты
    решения системы уравнений с определенным шагом
    N-ЧИСЛО УРАВНЕНИЙ;
    A,B - ПРЕДЕЛЫ ИНЕГРИРОВАНИЯ;
    H - ШАГ ИНТЕГРИРОВАНИЯ;
    TH - ШАГ ВЫВОДА РЕЗУЛЬТАТОВ;
    T - НАЧАЛЬНОЕ ВРЕМЯ;
    Y - НАЧАЛЬНЫЙ ВЕКТОР }
    VAR THP:DOUBLE;
       I:INTEGER;
    BEGIN
    WRITELN; WRITELN(FRUNG);
    WRITE(A:12); WRITE(FRUNG,A:12);
    FOR I:=1 TO N DO BEGIN WRITE(' ',Y[I]:12); WRITE(FRUNG,'
',Y[I]:12,' ') END;
    WRITELN; WRITELN(FRUNG);
    T:=A; THP:=A+TH;

    WHILE (T<=B) DO BEGIN

    RK4(VP,H,T,Y);

     IF (T>=THP) THEN BEGIN
       THP:=THP+TH;

       WRITE(T:12); WRITE(FRUNG,T:12,' ');
       FOR I:=1 TO N DO BEGIN WRITE(' ',Y[I]:12);
WRITE(FRUNG,' ',Y[I]:12,' ');
                  END;
       WRITELN; WRITELN(FRUNG);
                END ;

                          END ;

    WRITE(T:12); WRITE(FRUNG,T:12,' ');
       FOR I:=1 TO N DO BEGIN WRITE(' ',Y[I]:12);
WRITE(FRUNG,' ',Y[I]:12,' ');
                  END;
     WRITELN; WRITELN(FRUNG);
```

```
END{RUNG};

begin

{ввод параметров системы m1, m2, c, l}
vp[1]:=1; vp[2]:=0.5; vp[3]:=200; vp[4]:=1;
{Создание файла, в который записываются результаты счета}
assignfile(frung,'frungkutt.txt');  rewrite(frung);
{начальное и конечное значение времени, шаг интегрирования,
шаг печати}
A:=0; B:=5; h:=1e-4; th:=0.01;
{начальные условия}
t:=A;
y[1]:=0; y[2]:=1;
y[3]:=1; y[4]:=0;

RUNG(N,A,B,H,TH,T,Y);
closefile(frung);

end.
```

Графики, соответствующие решению задачи, приведены на рис. 4.4.

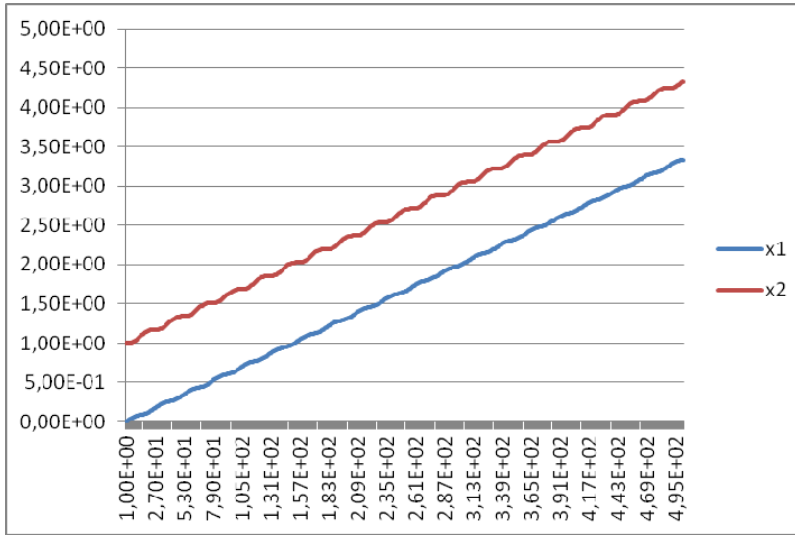


Рис. 4.4

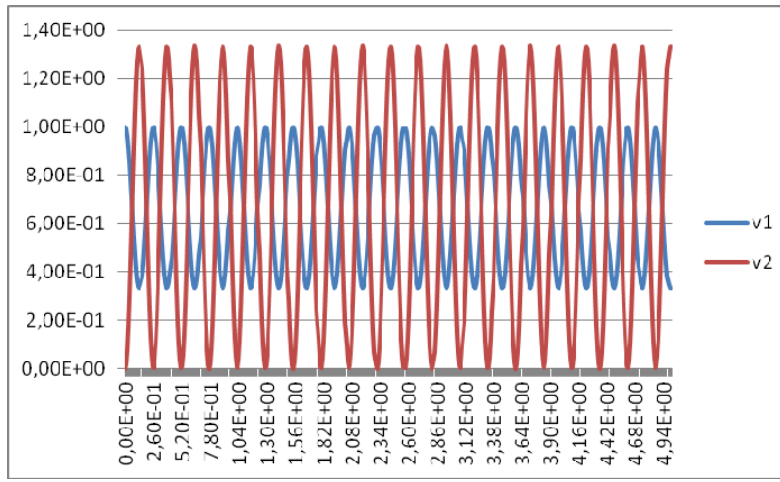


Рис. 4.4 (окончание)

З а м е ч а н и е. Данные для построения графиков после завершения работы программы RungKutt4 находятся в текстовом файле frungkutt.txt. Для построения графиков воспользуемся программой Microsoft Excel. Для этого запускаем Excel и открываем текстовый файл frungkutt.txt. Если на компьютере установлена рус-

ская версия Microsoft Excel, то меняем во всех числах десятичную точку на запятую. Следует помнить, что первый столбец с результатами – это переменная, по которой происходит интегрирование, второй – первая координата, третий – производная от первой координаты (скорость первой массы), четвертый столбец – вторая координата и, наконец, пятый столбец – производная от второй координаты (скорость второй массы). Выделяем нужные столбцы → выбираем в меню пункт **Вставка → Диаграммы → График → Вид графика.** Далее вызываем меню графика и выбираем пункт **Выбрать данные**, в котором устанавливаем подписи по горизонтальной оси (выбираем первый столбец) и легенду.

4.4. Перекачка энергии в колебательной системе

Рассмотрим систему, в которой два одинаковых маятника длины l и массы m каждый соединены на уровне a упругой пружиной с жесткостью c, прикрепленной концами к стержням маятников (рис. 4.5).

Определить малые колебания системы в плоскости равновесного положения маятников, после того как одному из маятников сообщено отклонение на угол α от положения равновесия; начальные скорости маятников равны нулю. Массами стержней маятников и массой пружины пренебречь [10].

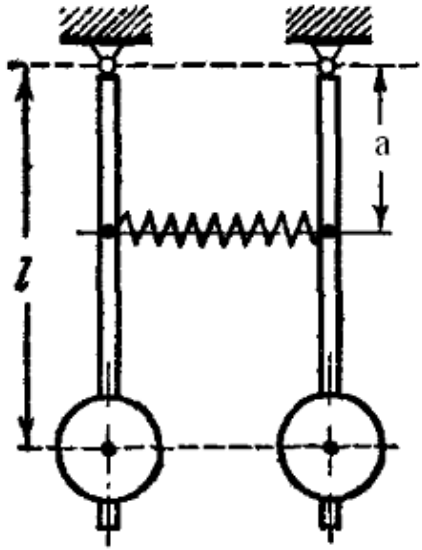


Рис. 4.5

Для составления дифференциальных уравнений, описываю-щих движение указанной системы, воспользуемся уравнениями Лагранжа второго рода. Для этого запишем кинетическую и потенциальную энергии системы:

$$T = \frac{1}{2} m_1 l_1^2 \dot{\varphi}_1^2 + \frac{1}{2} m_2 l_2^2 \dot{\varphi}_2^2 ,$$

$$\Pi = \frac{1}{2} c\left(a \sin\varphi_2 - a \sin\varphi_1\right)^2 + m_1 g l_1 \left(1 - \cos\varphi_1\right) + m_2 g l_2 \left(1 - \cos\varphi_2\right)$$

и подставим в уравнение Лагранжа второго рода

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{\varphi}_i} - \frac{\partial T}{\partial \varphi_i} = -\frac{\partial \Pi}{\partial \varphi_i}, \quad i = 1,2 .$$

В результате получаем систему двух дифференциальных уравнений второго порядка

$$\ddot{\varphi}_1 = \frac{1}{m_1 l_1^2}\left[ca^2\left(\sin\varphi_2 - \sin\varphi_1\right)\cos\varphi_1 - m_1 g l_1 \sin\varphi_1\right];$$

$$\ddot{\varphi}_2 = -\frac{1}{m_2 l_2^2}\left[ca^2\left(\sin\varphi_2 - \sin\varphi_1\right)\cos\varphi_2 + m_2 g l_2 \sin\varphi_2\right].$$

В программе численного решения дифференциальных уравнений методом Рунге-Кутта 4-го порядка следует ввести сле-дующую процедуру для вычисления правых частей полученной системы в нормальной форме:

```
procedure PRAV(vp:vconst; T:double; y:vector; var z:vector);
var m1,m2,c,l1,l2,phi1,phi2,omega1,omega2:double;
begin
m1:=vp[1]; m2:=vp[2]; c:=vp[3]; l1:=vp[4]; l2:=vp[5]; a:=vp[6];
phi1:=y[1]; omega1:=y[2];
phi2:=y[3]; omega2:=y[4];
z[1]:=omega1; z[2]:=(1/(m1*l1*l1))*(c*a*a*(sin(phi2)-sin(phi1))*
                        cos(phi1)-m1*10*l1*sin(phi1));
z[3]:=omega2; z[4]:=-(1/(m2*l2*l2))*(c*a*a*(sin(phi2)-sin(phi1))*
                        cos(phi2)+m2*10*l2*sin(phi2));
end;{prav}
```

и параметры системы, используемые для расчетов

```
{ввод параметров системы m1, m2, c, l1, l2, a}
vp[1]:=1; vp[2]:=1; vp[3]:=50; vp[4]:=1; vp[5]:=1; vp[6]:=0.1;
{Создание файла, в который записываются результаты счета}
assignfile(frung,'frungkutt.txt');  rewrite(frung);
{начальное и конечное значение времени, шаг
интегрирования, шаг печати}
A:=0; B:=100; h:=1e-4; th:=0.01;
{начальные условия}
t:=A;
y[1]:=pi/6; y[2]:=0;
y[3]:=0; y[4]:=0;
```

З а м е ч а н и е. Рис. 4.6, в отличие от предыдущего, построен в Microsoft Excel, но с использованием **точечной диаграммы**, которая допускает другие параметры построения диаграмм.

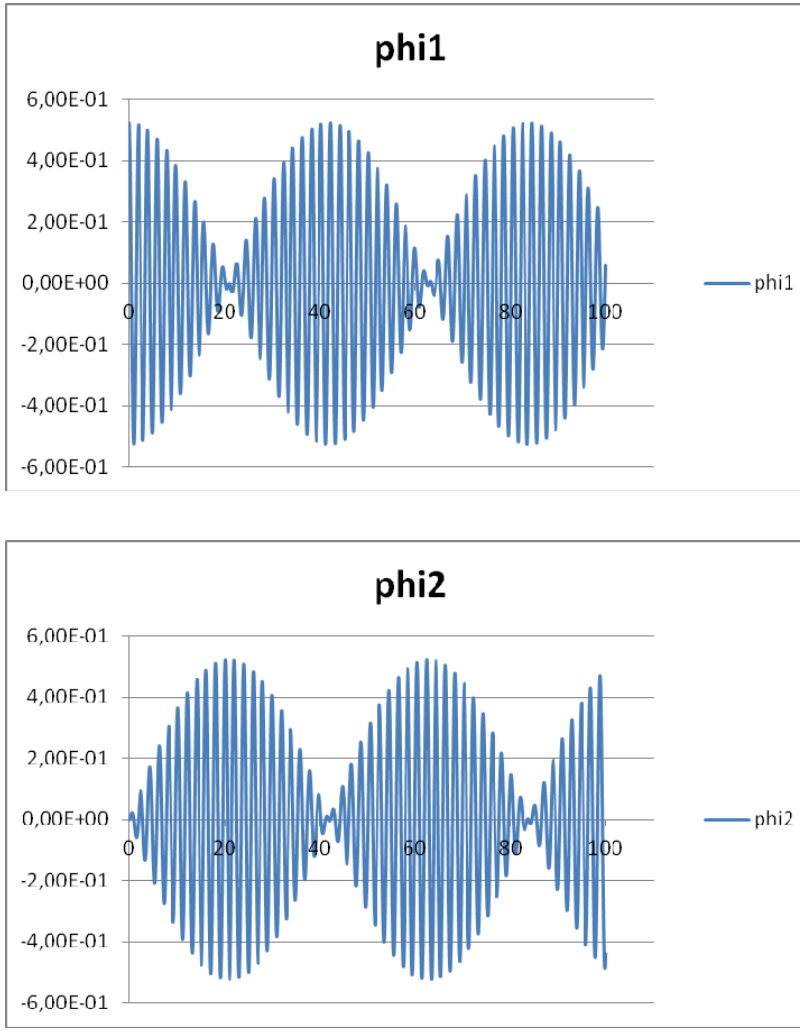


Рис. 4.6

Из представленных графиков (рис. 4.6) видно, что при почти полной остановке первого маятника второй имеет максимальную амплитуду колебаний и наоборот. Это явление называется перекачкой энергии. Теоретическое обоснование этого явления описано, например, в пособии Г.Т. Алдошина [11, 12].

4.5. Модифицированный метод Эйлера

Часто поведение механических систем описывается системами дифференциальных уравнений, которые не могут быть решены аналитически. В таких случаях прибегают к различным численным методам, простейшим из которых является метод Эйлера. Если неизвестная векторная функция обозначена $\vec{Y}(t)$, нормальная система дифференциальных уравнений $\vec{Y}' = F(t, \vec{Y})$ дополнена начальными условиями $\vec{Y}(t_0) = \vec{Y}_0$, то метод Эйлера позволяет в следующей расчетной точке $t_1 = t_0 + h$ приближенно получить значение векторной функции $\vec{Y}_1 = \vec{Y}_0 + h \cdot F(t_0, \vec{Y}_0)$.

Особенность дифференциальных уравнений, описывающих системы материальных точек, заключается в том, что они могут быть представлены в следующем частном виде:

$$\vec{X}'' = F(t, \vec{X}, \vec{X}') \Leftrightarrow \begin{cases} \vec{V}' = F(t, \vec{X}, \vec{V}), \\ \vec{X}' = \vec{V}. \end{cases}$$

Это позволяет модифицировать расчетную схему метода Эйлера:

$$\begin{cases} \vec{V}_1 = \vec{V}_0 + h \cdot F(t_0, \vec{X}_0, \vec{V}_0), \\ \vec{X}_1 = \vec{X}_0 + h \cdot \vec{V}_1 = \vec{X}_0 + h \cdot \vec{V}_0 + h^2 \cdot F(t_0, \vec{X}_0, \vec{V}_0). \end{cases}$$

Предложенные формулы отличаются от стандартных тем, что при вычислении $\vec{X}_1$ используется значение $\vec{V}_1$ вместо $\vec{V}_0$. Поскольку с помощью этих формул решение находится на промежутке $[t_0; t_1]$, то не так принципиально, в какой именно точке этого промежутка брать значение $\vec{V}$. Фактически это приводит к наличию ещё одного слагаемого, пропорционального h^2.

Покажем, что модифицированный метод более удобен для численного решения задач о колебаниях. Для этого рассмотрим простейшую колебательную систему, состоящую из одной материальной точки и одной пружины (рис. 4.7). Свободные колебания этого осциллятора описываются дифференциальным уравнением второго порядка:

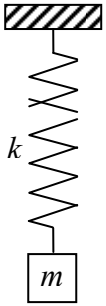


Рис. 4.7

$$mX''+kX=0 \Leftrightarrow \begin{cases} V'=-\dfrac{k}{m}X, \\ X'=V. \end{cases}$$

Согласно методу Эйлера

$$\begin{cases} V_1=V_0-\dfrac{hk}{m}X_0, \\ X_1=X_0+hV_0. \end{cases}$$

Общая энергия системы, равная сумме кинетической и потенциальной энергий, согласно одному из основных законов природы – закону сохранения энергии, должна оставаться постоянной. Проверим это. При $t=t_0$ $E_0=T_0+U_0$, $T_0=\dfrac{mV_0^2}{2}$, $U_0=\dfrac{kX_0^2}{2}$. Найдем общую энергию $E_1=T_1+U_1=\dfrac{mV_1^2}{2}+\dfrac{kX_1^2}{2}$ при $t=t_1$. Согласно методу Эйлера

$$E_1=\frac{m\left(V_0-\dfrac{hk}{m}X_0\right)^2}{2}+\frac{k\left(X_0+hV_0\right)^2}{2}.$$

Упростив, получим $E_1=E_0\left(1+h^2k/m\right)$. Энергия системы увеличивается с каждым шагом расчета по методу Эйлера, что противоречит закону сохранения. Методы Рунге–Кутта являются обобщениями метода Эйлера, который одновременно называется методом Рунге–Кутта первого порядка. Не спасает использование методов более высоких порядков. Например, для метода Рунге–Кутта второго порядка $E_1=E_0\left(1+\dfrac{h^4k^2}{4m^2}\right)$ и энергия системы тоже бес-

предельно увеличивается, хотя и не так быстро. Сделаем аналогичный расчет для модифицированного метода Эйлера

$$\begin{cases} V_1 = V_0 - \dfrac{hk}{m} X_0, \\ X_1 = X_0 + h\left(V_0 - \dfrac{hk}{m} X_0 \right). \end{cases}$$

А полная энергия системы при $t = t_1$ будет равна

$$E_1 = E_0 + \frac{h^2 k}{m}\left(T_0 - U_0 - hk\left(X_0 V_0 - \frac{hU_0}{m} \right) \right).$$

Принципиальным отличием этого результата от вытекающего из метода Эйлера является то, что энергия системы увеличивается пропорционально разности кинетической и потенциальной энергий, а не их сумме. Это особенно актуально при расчете колебательных процессов, поскольку в них кинетическая и потенциальная энергии преобразуются одна в другую, и набегающие погрешности расчета гасят друг друга за цикл. Слагаемое, пропорциональное h^3, тоже периодически меняется за цикл, колеблясь относительно нуля. В итоге, накапливаемая погрешность пропорциональна h^4, как для метода Рунге–Кутта второго порядка, а количество производимых арифметический операций такое же как и в методе Эйлера.

Библиографический список

1. *Буч, Г.* Объектно-ориентированное проектирование с примерами применения / Г. Буч. Киев, 1992.

2. *Марченко, А. И.* Программирование в среде Borland Pascal 7.0 / А.И. Марченко. Киев, 1996.

3. *Страуструп, Б.* Язык программирования С++. Специальное издание: Пер. с англ. / Б. Страуструп. М., 2004.

4. *Практическое* руководство по программированию / Под ред. Б. Мика, П. Хит, Н. Рашби. М., 1986.

5. *Боревич, З.И.* Определители и матрицы / З.И. Боревич. СПб., 2004.

6. *Романовский, И.В.* Дискретный анализ / И.В. Романовский. СПб., 2003.

7. *Самарский, А.А* Численные методы / А.А. Самарский, А.В. Гулин. М., 1989.

8. *Вирт, Н.* Алгоритмы и структуры данных / Н. Вирт. М., 1989.

9. *Вирт, Н.* Программирование на языке Модула-2 / Н. Вирт М., 1987.

10. *Мещерский, И.В*. Сборник задач о теоретической механике / И.В. Мещерский. М., 1986.

11. *Алдошин, Г.Т.* Теория колебаний. Ч.1. Линейные колебания: учебное пособие / Г.Т. Алдошин; Балт. гос. техн. ун-т. СПб., 2006.

12. *Алдошин, Г.Т.* Теория колебаний. Ч.2. Нелинейные колебания: учебное пособие / Г.Т. Алдошин; Балт. гос. техн. ун-т. СПб., 2010.

О Г Л А В Л Е Н И Е

Дмитриев Никита Николаевич, Сахаров Вадим Юрьевич

Моделирование инженерных задач на языке программирования Free Pascal в среде Lazarus

Редактор *Г.В. Никитина*
Корректор *Л.А. Петрова*
Подписано в печать 8.11.2012. Формат 60x84/16. Бумага документная.
Печать трафаретная. Усл. печ. л. 3,25. Тираж 200 экз. Заказ № 177.
Балтийский государственный технический университет
Типография БГТУ
190005, С.-Петербург, 1-я Красноармейская ул., д.1