№ 747

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
ИНСТИТУТ СТАЛИ и СПЛАВОВ
Технологический университет
МИСиС

Кафедра обработки металлов давлением

Галкин С.П., Михайлов В.К.

Одобрено Методическим
советом института

ПРИКЛАДНЫЕ КОМПЬЮТЕРНЫЕ СИСТЕМЫ

Раздел: *Решение задач численными методами в среде электронных таблиц Excel*

Учебное пособие
для практических занятий
студентов специальностей 1106, 1204, 1703

МОСКВА 2001

УДК 004.42

АННОТАЦИЯ

Учебное пособие предназначено для практических занятий по курсам «Информатика», «Прикладные компьютерные системы», «Моделирование и оптимизация технологических систем».

Пособие содержит теоретическое описание численных методов, используемых при решении инженерных задач, алгоритмы их реализации, программное обеспечение этих методов на алгоритмическом языке VBA (Visual Basic for Application) для Excel 97, а также примеры и многовариантные задания.

Пособие состоит из восьми тематических частей по основам численных методов и дополнительных сведений, посвященных программированию в VBA. Овладение этим материалом позволит научиться редактировать и самостоятельно составлять программный код макросов на языке VBA для всех приложений Office 97 (Excel, Word, Access и др.), а также AutoCad 14, 15, а в дальнейшем перейти непосредственно к визуальному, объектно-ориентированному программированию на алгоритмическом языке Visual Basic версий 5 и 6.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Активное распространение программных продуктов компании Microsoft, работающих в операционной системе **Windows**, позволяет решать многочисленные практические задачи. Большая часть этих задач может быть решена без применения программирования. Однако для решения нестандартных научных и инженерных задач, а также для автоматизации работы приложений без программирования не обойтись.

В качестве языка программирования используется один из самых динамичных и современных макроязыков – Visual Basic for Application (VBA) в среде **Excel 97**, который имеет следующие преимущества:

- существенно расширяет, как возможности **Excel 97**, так любого другого приложения, поддерживаемого этим макроязыком (в настоящее время их более 50, в том числе **Word–97**, **Access–97**, **Power Point 97**, **AutoCad 14**, **AutoCad 15**);
- позволяет создавать «мегапрограммы», объединяющие и координирующие усилия мощных прикладных программ, автоматизирующие и облегчающие работу пользователя;
- позволяет относительно легко освоить как численные методы, так и основы самого макроязыка.

Последнее преимущество языка может быть описано по следующей схеме: создание макросов макрорекордером → правка программного кода макросов и освоение основ VBA → самостоятельное создание программ «вручную» с использованием, по мере освоения VBA, все большего числа его возможностей. Для облегчения самостоятельного изучения VBA в пособии изложены основы работы с VBA и его справочной системой (см. раздел 1). Более полное описание языка VBA можно найти в литературных источниках [7] – [9].

При написании материала настоящего пособия в основу положены литературные источники [1]–[6], в которых приемы и методы программирования изложены более основательно, но на базе других алгоритмических языков.

В восьми разделах пособия (со 2-го по 9-й) описаны численные методы, используемые для:

– решения систем линейных уравнений;
– решения задач интерполяции;
– решения задач аппроксимации;
– решения алгебраических и трансцендентных уравнений;
– определения экстремумов функций;
– решения систем нелинейных уравнений;
– вычисления определенных интегралов;
– численного интегрирования дифференциальных уравнений.

В каждом разделе рассмотрено от 3 до 7 различных численных методов. Порядок рассмотрения практически любого метода следующий:
– теоретическое изложение метода;
– описание алгоритма этого метода;
– программный код, реализующий алгоритм;
– пример.

В каждом разделе приведены задания, которые предлагается решить тем или иным методом, и контрольные вопросы.

1. ОСНОВЫ РАБОТЫ С VBA В MICROSOFT EXCEL 97

Компанией Microsoft разработана технология, которую называют ActiveX Automation. Эта технология позволяет обращаться к объектам приложения средствами любого макроязыка, если приложение и макроязык поддерживают эту технологию. С помощью ActiveX Automation можно автоматизировать работу за компьютером, разрабатывая макросы и специальные приложения. Для этого можно использовать современные стандартные дружественные к пользователю макроязыки типа: Visual Basic, Microsoft Visual C++, Delphi и др. Кроме перечисленных выше, к дружественным макроязыкам относится Visual Basic for Application (VBA), главная особенность которого заключается в том, что компоненты Microsoft Office 97 включают этот язык, и нет необходимости приобретать отдельно любой из перечисленных, к тому же дорогостоящих, макроязыков. На сегодняшний момент VBA является единым языком, который используют уже более 50 приложений, и их число неуклонно продолжает расти. К этим приложениям относятся: **Word–97**, **Access–97**, **Excel–97**, **Power Point 97**, **AutoCad 14**, **AutoCad 15** и др.

Макроязык VBA является современным языком визуального и объектно-ориентированного программирования. Он является объектным языком по двум причинам. Во-первых, он позволяет создавать собственные классы объектов, а, во-вторых, что более важно, позволяет работать с огромным числом объектов, содержащихся в библиотеках. Все приложения, входящие в **Office 97**: **Excel**, **Word**, **Access** и другие, представляют собой совокупность объектов со своими свойствами, методами и событиями.

Рассмотрим основы работы с VBA в **Excel-97**.

1.1. Переменные VBA

Подобно ячейкам электронных таблиц переменные VBA могут сохранять текст, даты и другие типы данных. Для корректной и быстрой работы программ необходимо объявлять переменные и тип их данных заранее.

Самый простой способ – использование оператора `Dim`, например, для описания обычной переменной целого типа –

```
Dim x As Integer,
```

где `x` – переменная, а `Integer` – ее тип;

для описания переменной одномерного массива –

```
Dim m(25) As Integer;
```

для динамически объявляемого массива:

```
Dim k As Integer
Dim z() As Integer
k = 125
ReDim z(k)
```

В VBA используются 13 типов данных, которые вместе с размерами, требуемыми для сохранения значений, и диапазонами допустимых значений перечислены в таблице.

Характеристики типов данных

Название	Размер, байт	Диапазон значений
Byte (байт)	1	от 0 до 255
Boolean (логический)	2	true или False
Integer (целое)	2	от –32768 до 32767
Long (длинное целое)	4	от –2147483648 до 2147483647
Single (с плавающей точкой обычной точности)	4	от –3,402823 E+38 до –1,401298 E–45 для отрицательных значений; от 1,401298 E-45 до 3,402823 E+38 для положительных значений
Double (с плавающей точкой двойной точности)	8	от –1,79769313486232 E+308 до –4,94065645841247 E-324 для отрицательных значений; от 4,94065645841247 E–324 до 1,79769313486232 E+308 для положительных значений

Продолжение таблицы

Название	Размер, байт	Диапазон значений
Currency (денежный)	8	от –922337203685477,5808 до 922337203685477,5807
Decimal (масштабируемое целое)	14	+/–79228162514264337593543950335 без дробной части; +/–7,9228162514264337593543950335 с 28 знаками справа от запятой; минимальное ненулевое значение: +/–0,0000000000000000000000000001
Date (даты и время)	8	от 1 января 100 г. до 31 декабря 9999 г.
Object (объект)	4	любой указатель объекта.
String (строка переменной длины)	10 + длина строки	от 0 до приблизительно 2 миллиардов.
String (строка постоянной длины)	Длина строки	от 1 до приблизительно 65400.
Variant (числовые подтипы)	16	любое числовое значение вплоть до границ диапазона для типа Double.
Variant (строковые подтипы)	22 + длина строки	как для строки (String) переменной длины.
Тип данных, определяемый пользователем (с помощью ключевого слова Type)	Объем определяется элементами	диапазон каждого элемента определяется его типом данных.

Под массив любого типа данных необходимо выделить оперативную память, исходя из суммы следующих слагаемых:
– 20 байт на любой массив;
– 4 байта на каждую размерность массива;
– число байт для хранения данных.

Объем памяти, требуемый для сохранения данных, рассчитывается как произведение числа элементов на размер элемента. Например, данные в одномерном массиве, который содержит четыре элемента типа Integer, требующих по 2 байта на элемент, занимают 8 байт. Вместе с 20 байтами на массив и 4 байтами на размерность общий требуемый объем составляет 32 байта.

Значение типа Variant, содержащее массив, требует 12 байт в дополнение к объему, требуемому массивом.

1.2. Функции, процедуры, макросы

В алгоритмическом языке VBA в отличие от классического языка Visual Basic нет такого понятия, как основная программа, которая может использовать процедуры и функции. В VBA роль основной программы играет *макрос*, который может по необходимости обращаться к *процедурам* и *функциям* или к другому макросу.

1.2.1. Макрос

Макрос – это набор инструкций, указывающих последовательность действий, которые **Microsoft Excel 97** должен выполнить вместо вас. Макросы, по сути, являются компьютерными программами, которые не выполняются независимо, а запускаются и работают только внутри **Excel**. Они используются для автоматизации трудоемких и часто повторяющихся задач.

Макросу при вызове не могут быть переданы никакие параметры, однако во время выполнения он может находить или получать нужные ему значения из файлов данных, из рабочих листов и с помощью специальных диалоговых окон. Первая и последняя строки программного кода макроса являются начальной и конечной точками макроса. Эти строки должны начинаться инструкцией `Sub`, содержащей через пробел имя макроса и пустой список параметров, обозначаемый двумя, подряд идущими, открывающей и закрывающей скобками, а заканчиваться инструкцией `End Sub`:

```
Sub Макрос1()
   текст макроса
End Sub
```

Наличие пустого списка параметров позволяет VBA классифицировать эту программную единицу как макрос и представлять его в диалоговом окне Макрос (меню Сервис\Макрос\Макросы…).

В тексте программы макрос может быть вызван двумя способами:

```
1. Call Макрос1()
```

или

```
2. Макрос1
```

1.2.2. Процедура

Программная единица, называемая *процедурой*, имеет список передаваемых параметров. По этому признаку можно определить, является программная единица макросом или процедурой, взглянув только на первую строку программного кода:

```
Sub Фигура(Длина As Single, Ширина As Single,_
 Площадь As Single)
   'Вычисление площади прямоугольника
    Площадь = Длина * Ширина
End Sub
```

В данном примере за инструкцией `Sub` идет имя процедуры «Фигура», затем параметры: Длина, Ширина, Площадь с описанием типа этих параметров.

В тексте программы процедура может быть вызвана двумя способами:

```
1. Call Фигура(a, b, s)
```

или

```
2. Фигура a, b, s
```

Параметры при вызове: *a*, *b*, *s* должны иметь тип, соответствующий параметрам передачи: Длина, Ширина, Площадь.

Следует отметить, что процедуру нельзя запустить из диалогового окна Макрос, она может вызываться только в строках программного кода VBA.

1.2.3. Функция

Программная единица функция возвращает при вызове единственное значение. Этим она отличается от процедур, которые могут возвращать одно и более значений. Функции могут быть встроенными в Excel или созданы пользователем.

Следующая пользовательская функция принимает три аргумента, а возвращает результат, являющийся произведением первых двух параметров и суммой третьего:

```
Function ТриАргумента(x, y, z)
   ТриАргумента = x * y + z
End Function
```

Вызов такой функции с помощью программного кода происходит следующим образом:

```
Результат = ТриАргумента(x, y, z),
```

где x, y, z – числа или переменные, имеющие числовые значения.

В общем случае первая строка программного кода имеет вид:

```
Function ТриАргумента(x As Тип1, y As Тип2,
z As Тип3) As Тип4,
```

где ТипN, N = 1, 2, 3 – это типы переменных и самой функции.

При совместной работе VBA и **Excel** типы, как правило, не описывают. Они по умолчанию принимаются как Variant. Это связано с тем, что в VBA в качестве разделителя целой и дробной части числа используется точка, а в **Excel** – запятая.

В программном коде можно использовать функцию **Excel** AVERAGE следующим образом:

```
СреднееЗначение = Application.Average(12, 24, 8).
```

1.3. Способы создания макросов

Существует два способа создания макросов:

– автоматический, при котором записывают последовательность своих действий с использованием макрорекордера;

– ручной, при котором вводят инструкции в особом листе **Excel**, называемом модулем.

Для ввода инструкций в модуле используется язык программирования VBA.

Автоматический способ создания макросов

Процесс записи макроса с помощью макрорекордера состоит из трех шагов:

– активизировать режим записи макроса и присвоить ему имя;
– выполнить действия, которые требуется записать, например: выделение ячеек, ввод и форматирование таблиц;
– остановить запись макроса.

Активизация режима работы макроса происходит следующим образом: в меню Сервис (Tools) выбрать команду Макрос (Macro)\Начать запись (Record New Macro). Excel выведет на экран окно диалога Запись макроса (Record Macro). В этом окне можно указать четыре опции.

1. Назначить имя макросу или принять предложение **Excel** –Макрос1 – в поле Имя макроса.
2. Назначить макросу комбинацию клавиш, по которой он будет запускаться, введя в поле Сочетание клавиш (Shortcut Key) букву или, удерживая клавишу [Shift], нажать букву.
3. Сохранить макрос в активной книге, для чего в поле Сохранить (Store Macro In) вывести Эта книга (This Workbook).
4. Ввести краткий комментарий к макросу в поле Описание (Description).

Чтобы начать запись макроса, нужно нажать кнопку ОК. **Excel** выведет в строке состояния сообщение Запись (Recording), и на экране появится панель инструментов Остановка записи (Stop Recording) с кнопкой Остановить запись (Stop Recording Macro). Далее выполняются действия, которые следует записать. После их завершения нужно нажать кнопку Остановить запись (Stop Recording Macro) на панели инструментов Остановка записи (Stop Recording) или выбрать в меню Сервис (Tools) команду Макрос (Macro)\Остановить запись (Stop Recording). Этот шаг необходим, иначе **Excel** будет продолжать запись действий бесконечно.

Ручной способ создания макросов

Процесс создания макроса вручную состоит из трех шагов.

1. Активизировать редактор VBA путем выполнения серии действий с меню: Сервис\Макрос\Редактор Visual Basic.
2. Вставить модульный лист в меню Вставка, выбрать опцию Модуль, или щелкнуть мышкой на нужном модуле в окне проекта.
3. Начать работу по набору текста макроса в модульном листе.

Чтобы проверить работу нового макроса, нужно очистить рабочий лист и нажать сочетание клавиш [Ctrl]+[буква] или [Ctrl]+[Shift]+[буква] в зависимости от того, что было указано в поле Сочетание клавиш окна Запись макроса. **Excel** запустит макрос и выполнит произведенные действия в той же последовательности, в которой они были записаны.

Запуск макроса можно выполнить и другим способом. В меню Сервис выбрать команду Макрос\Макросы (Macros), чтобы открыть окно диалога Макрос. Далее выбрать имя макроса и нажать кнопку Выполнить (Run).

Как упоминалось выше, запись макроса производит макрорекордер. Его работа начинается после нажатия кнопки ОК в окне диалога Запись макроса и заканчивается нажатием кнопки Остановить запись. В начале своей работы макрорекордер вставляет в текущую книгу так называемый модуль (module). В этот модуль помещается программный код на языке Visual Basic, содержащий инструкции всех действий, производимых пользователем во время работы макрорекордера. Модуль не появляется в книге вместе с другими листами, увидеть его можно тремя способами.

Первый способ: в меню Сервис необходимо выбрать команду Макрос\Макросы. После выделения в окне диалога выбранного макроса и нажатия кнопки Изменить запустится редактор Visual Basic, и появится модуль, содержащий текст с инструкциями макроса.

Второй способ вывода на экран программного кода макроса заключается в выполнении цепочки команд Сервис\Макрос\Редактор Visual Basic и выборе в раскрывающемся списке поля Описания имени процедуры, совпадающей с именем макроса.

Третий, самый быстрый *способ* вызова редактора Visual Basic – использование комбинации клавиш [Alt]+[F11] (повторное нажатие [Alt]+[F11] приведет к возвращению в текущий лист).

Модуль совсем не похож на рабочий лист. Вместо сетки строк и столбцов – окно, которое можно встретить в текстовом процессоре. Это окно содержит меню, команды которого позволяют редактировать, отлаживать и запускать программы Visual Basic. В модуле можно вводить, копировать, вставлять, перемещать и удалять инструкции Visual Basic и комментарии, используя приемы, знакомые по текстовым процессорам (например, **Word**).

1.4. Обмен информацией между ячейками рабочего листа и переменными VBA

В **Excel**, с одной стороны, имеются рабочие листы, в которых можно работать с диапазонами, ячейками, столбцами, строками, формулами и данными. С другой стороны, имеются макросы, процедуры и функции, где выполняются вычисления. При решении практически любой задачи возникает необходимость обмениваться информацией. Существует большое количество способов это сделать. Рассмотрим самые простые из них – передача и считывание данных.

Ссылка на ячейку рабочего листа с использованием метода `Cells` выглядит так: `Cells(7, 2)`, где 7 – номер строки, а 2 – номер столбца. Приведем пример передачи данных в ячейки листа:

```
Dim rws As Integer, cols As Integer
For rws=1 To 10
   For cols=1 To 10
      Cells(rws, cols).Value = rws*cols
   Next cols
Next rws
```

Этот программный код VBA заполняет блок ячеек размером 10×10 таблицей умножения. Используя вложенный цикл, он по очереди проходит каждую ячейку в блоке. Для того чтобы понять, как происходит работа, рассмотрим четвертую строку:

`Cells(rws, cols).Value = rws*cols`, которую можно записать и короче:

```
Cells(rws, cols) = rws*cols.
```

Эта строка берет две переменные – `rws` и `cols`, и использует их для определения ячейки. Если переменная `rws` в настоящий момент равна 2 и `cols` равна 3, то `Cells(rws, cols)` выбирает ячейку в строке 2 и столбце 3, что является ячейкой «B3». Значение этой ячейки становится равным `rws*cols` или в данном случае – 6. Таким образом, всякий раз при выполнении средней строки в соответствующую ячейку помещается произведение двух переменных.

По аналогии с предыдущим примером приведем пример считывания данных из ячеек листа:

```
Dim rws As Integer, cols As Integer, a() As Double
ReDim a(10, 10)
For rws=1 To 10
   For cols=1 To 10
      a(rws, cols) = Cells(rws, cols).Value
   Next cols
Next rws
```

В пятой строке происходит считывание данных из текущей ячейки `Cells(rws, cols)` и передача в двухмерный массив `a(rws, cols)`.

Если при форматировании ячеек рабочего листа из VBA необходимо ссылаться сразу на несколько ячеек, то для этого используется метод Range. Его можно применять для выделения прямоугольного блока или нескольких отдельных блоков. Блок как минимум должен содержать хотя бы одну ячейку. Прямоугольные блоки ячеек задаются с использованием ячейки начала и ячейки конца, отделяемых друг от друга двоеточием:

– выделение блока из 12 ячеек –

```
Range("B4:E6").Select
```

– выделение различных одиночных ячеек –

```
Range("A1,B2,C1,D2").Select
```

– выделение двух разных блоков ячеек –

```
Range("A2:C4,D8:F10").Select
```

1.5. Диалоговые окна

Для передачи данных в макросы, процедуры и функции или вывода сообщений пользователю используются *диалоговые окна*. Самыми простыми диалоговыми окнами являются *окна сообщений*. Это диалоговые окна, которые выдают пользователю сообщения и снабжаются одной или более кнопками для выбора. В VBA они создаются с использованием функции MsgBox. В своей самой простой форме MsgBox используется как оператор с одним аргументом – сообщением, которое должно отображаться. Например, следующая строка кода создает окно сообщения с выводом информации «Это окно сообщения»:

```
MsgBox "Это окно сообщения".
```

MsgBox можно использовать и для отображения числового значения переменной:

```
Dim x As Integer
x = 10
MsgBox x
```

Полное описание этой функции можно найти в справочной системе VBA.

Для задания значений переменным программного модуля используется *окно ввода*. Окно ввода вызывает функция InputBox. Функция InputBox имеет следующий синтаксис:

```
InputBox(сообщение[, заголовок]),
```

где `сообщение` – обязательный аргумент, который содержит текстовое сообщение пользователю;

`заголовок` – аргумент, являющийся необязательным и, если он опущен, используется заголовок по умолчанию.

Возвращаемым значением функции InputBox является любая информация, вводимая пользователем в текстовое поле отображаемого окна ввода.

Еще один вид окон VBA – это отображение обычных диалоговых окон **Excel**. Например, код для отображения диалогового окна Открытие документа следующий:

```
Application.Dialogs(xlDialogOpen).Show.
```

Если этот код запустить, появится диалоговое окно и будет работать так, как если бы пользователь ранее выбрал Открыть из меню Файл. Для отображения другого диалогового окна, необходимо заменить константу `xlDialogOpen` на константу требуемого диалога. Список всех констант можно найти в Справочной системе.

1.6. Отладка и редактирование кода

В режим отладки VBA переходит, когда во время выполнения макроса возникает необрабатываемая программным путем ошибка выполнения. При этом выдается сообщение об ошибке с возможностью выбора одного из вариантов: завершение выполнения или переход в режим отладки. Большое преимущество режима отладки заключается в том, что выполнение программного кода приостанавливается в месте возникновения ошибки. Другим важным моментом является то, что при этом сохраняются значения всех текущих переменных.

VBA предоставляет еще одну возможность переключения приложения в режим отладки. Это возможно благодаря *точке останова* (*BreakPoint*). *Точка останова* – это выделенная строка программного кода, на которой автоматически останавливается выполнение. По достижении этой строки VBA также переходит в режим отладки. В VBA имеется новая возможность установки и удаления *точек останова* с помощью полоски индикатора. Если щелкнуть слева в окне кода левой кнопкой мыши, то в этом месте появится коричневая точка и находящаяся напротив строка закрашивается коричневым цветом. Удаление *точки останова* производится повторным щелчком мыши по тому же месту. Кроме этого, установить и удалить *точки останова* можно также с помощью контекстного меню, или кнопки *Точка останова* панели инструментов, или нажатием клавиши [F9].

Точка останова может располагаться в любой строке кода, включая заголовки макросов, процедур и функций и строку с кодом

`End`. *Точки останова* нельзя устанавливать только в строках комментария или пустых строках.

В режиме отладки особым образом выделяется строка, которая должна выполняться следующей. Строка выделяется желтым цветом, а на полосе индикатора рядом с ней появляется желтая стрелка. Если выполнение программного кода прерывается в *точке останова*, то оба выделения комбинируются. При этом важно, чтобы строка выделялась и как следующий выполненный оператор, т. е. эта строка еще не выполнялась, а только подлежит выполнению. Чтобы продолжить выполнение с текущей строки необходимо нажать клавишу [F5], а для продолжения с любой другой строки – перетащить мышью желтую стрелку полосы индикатора на нужную строку. Если попытаться установить желтую стрелку на строку, которая не может быть выполнена, курсор мыши примет вид, указывающий на невозможность переноса. После отпускания кнопки мыши данная строка не выделяется.

Особый интерес представляет возможность выделения следующей выполняемой строки при пошаговом выполнении. В этом случае можно точно проследить очередность выполнения операторов, что важно, например, в операторах ветвления, когда необходимо точно установить, какая ветвь программного кода выполняется. Пошаговое выполнение является важным средством поиска ошибок и отладки программного кода. Существует несколько различных команд пошагового выполнения: Шаг с заходом, Шаг с обходом, Шаг с выходом и Выполнить до текущей позиции.

Шаг с заходом позволяет не только выполнить текущий оператор, но если это оператор вызова макроса, процедуры или функции, дает возможность перейти в эту программную единицу.

Шаг с обходом подобен шагу с заходом, но эта команда выполняет вызов процедуры как единичный оператор, т. е. без захода.

На практике для эффективного использования этих двух команд должны выполняться описанные ниже действия. Командой Шаг с обходом начать поиск ошибки для ее локализации. После нахождения ошибки в вызываемом программном модуле проверить ее с помощью команды Шаг с заходом.

Команда Шаг с выходом осуществляет выход из программного модуля без последующего пошагового выполнения всех его операторов. Эту команду целесообразно использовать, если при выполнении ошибочно произошел вход в вызываемый модуль, или стало