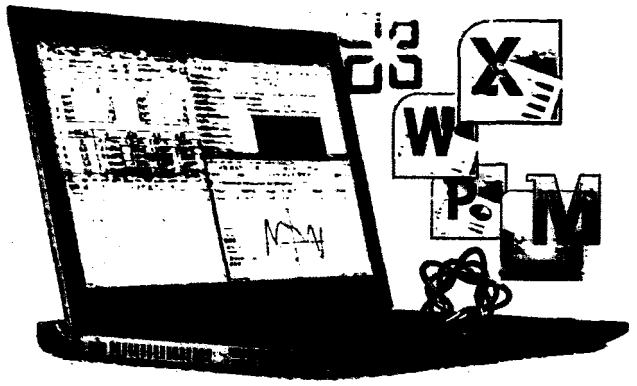
ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по дисциплине «Информатика»

для студентов строительных специальностей

дневной формы обучения

второй семестр



МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ИНФОРМАТИКИ И ПРИКЛАДНОЙ МАТЕМАТИКИ

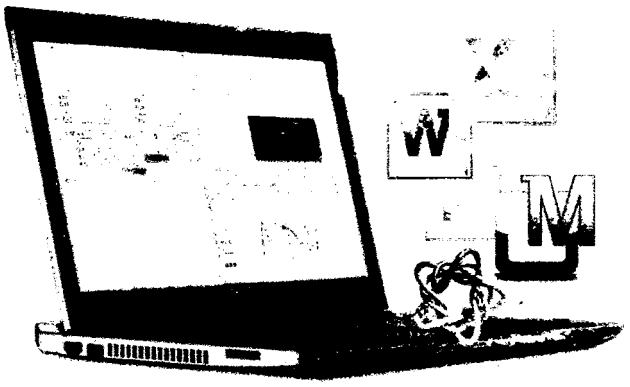
ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по дисциплине «Информатика»

для студентов строительных специальностей

дневной формы обучения

второй семестр



строительный факультет

Группа _____

Фамилия _____

Имя _____

Отчество _____

Вариант _____

м/тел. +375 (___) ____-___-___

Брест 2015

УДК 004

Практикум предназначен для студентов первого курса строительного факультета, изучающих дисциплину «Информатика». В него входят задания для лабораторных работ и задания для их защиты. В теоретической части изложены методические рекомендации по работе с электронными таблицами MS EXCEL с поддержкой VBA и системами компьютерной математики MathCAD и Maple.

Составители: В.А. Кофанов, к.т.н., доцент
Т.Г. Хомицкая, ст. преподаватель
И.В. Тузик, ст. преподаватель

Рецензенты: зав. кафедрой прикладной математики и технологии программирования БрГУ имени А.С. Пушкина, доцент, к.ф.-м.н. О.В. Матысик;
директор Брестского филиала ИООО EPAM Systems, доцент, к.ф.-м.н. С.А. Тузик.

Учреждение образования

Общие указания

Практикум предназначен для организации самостоятельной, практической и лабораторной работы студентов во втором семестре изучения дисциплины «Информатика».

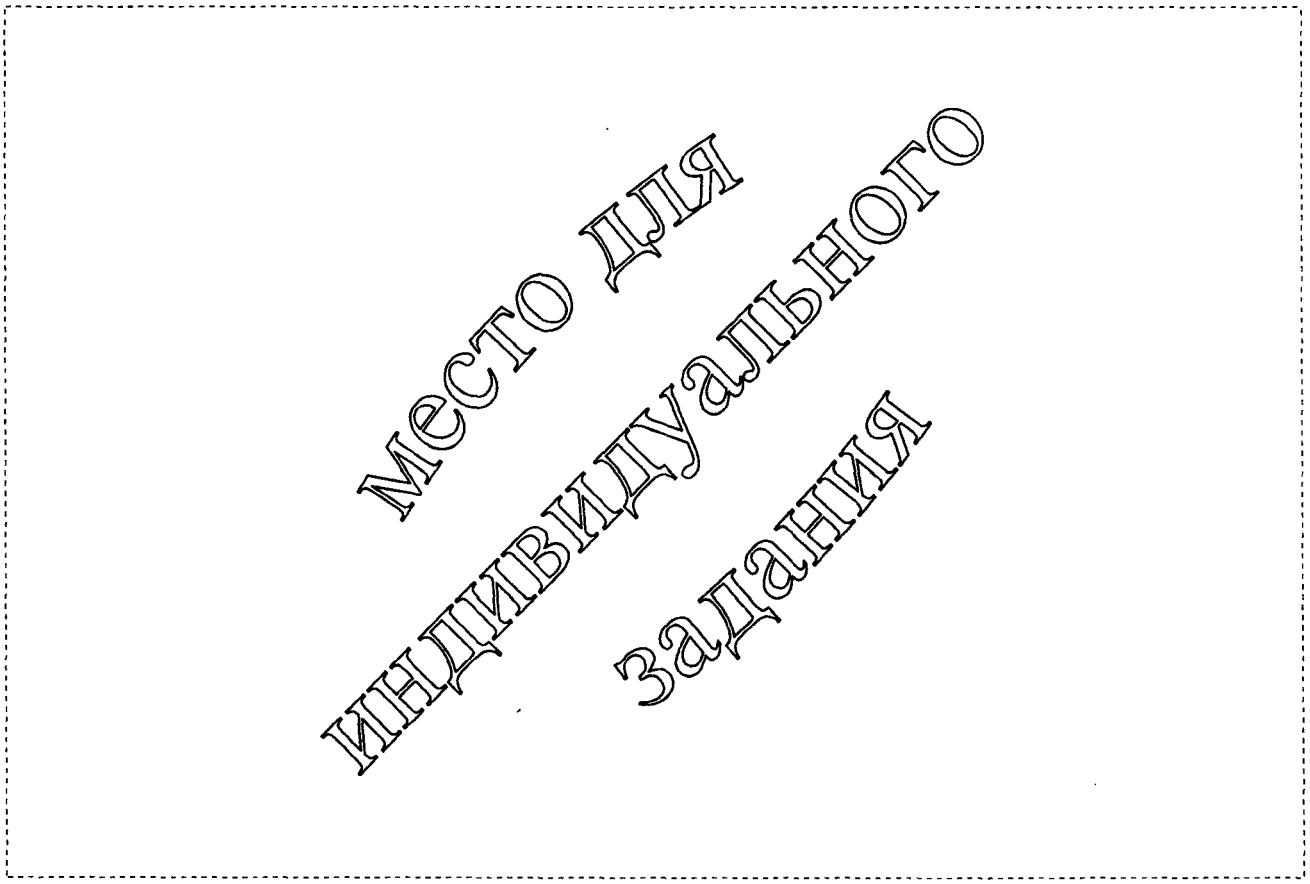
Перед началом работы необходимо привести информацию на титульном листе, то есть должны быть указаны данные ее владельца (Ф.И.О., группа, номер мобильного телефона), а также вариант заданий.

На текущей странице в блок «Индивидуальное задание» необходимо вклеить листок с вариантами заданий, полученный у преподавателя. Специальный блок «Условие» на странице с лабораторной работой заполняется простым переписыванием условия задачи с бланка индивидуального задания.

При выполнении лабораторных работ результаты вычислений заносятся в отчет в том же виде, в котором они отображаются на экране монитора. Ведение записей выполняется четко и разборчиво шариковой ручкой (блок-схемы – карандашом). Неправильные (ошибочные) записи на страницах практикума необходимо исправлять с использованием корректирующих средств (корректирующие ленты, штрих-корректоры и т.п.).

Каждая лабораторная работа считается выполненной только при наличии отметки преподавателя о ее защите, подтвержденной его подписью (личной печатью) на стр. 4. Данные из этого листа служат основанием для допуска к итоговым испытаниям (зачету, экзамену).

Индивидуальное задание



Отметки о защите лабораторных работ

№	Наименование работы	Защита	Примечание
1	Лабораторная работа №1(1,2,3)		
2	Лабораторная работа №2		
3	Лабораторная работа №3(1,2)		
4	Лабораторная работа №4		
5	Лабораторная работа №5(1,2,3)		
6	Лабораторная работа №6(1,2,3)		
7	Лабораторная работа №7(1,2)		

Итог работы в семестре:

Все лабораторные работы выполнены в полном объеме.

_____

(дата, подпись)

Методические указания к выполнению лабораторных работ

ЭЛЕКТРОННАЯ ТАБЛИЦА MICROSOFT EXCEL

Условное форматирование

Условное форматирование позволяет задать определенный формат диапазона ячеек в зависимости от содержимого этого диапазона. Как правило, когда вы задаете условия для форматирования диапазона ячеек (2003 – **Формат/Условное форматирование/Формула**; 2007+ – **Главная/Стили/Условное форматирование/ Использовать формулу для определения форматируемых ячеек**), Excel проверяет каждую ячейку диапазона для определения ее соответствия заданным вами условиям. Затем Excel применяет выбранный вами для данного условия формат во всех ячейках, удовлетворяющих этому условию. Если содержимое ячейки не отвечает любому из заданных условий, форматирование ячейки не меняется.

Предположим, что необходимо отформатировать ячейки массива B7:J17 (рисунок *1*), в которых *число больше (меньше) чисел, содержащихся в соседних восьми ячейках*. Восемь соседних ячеек есть только у ячеек диапазона C8:I16.

	A	B	C	D	E	F	G	H	I	J
6	x \ y	-3	-2.25	-1.5	-0.75	0	0.75	1.5	2.25	3
7	0	-0.275	0.362	0.581	0.265	-0.416	-1.098	-1.414	-1.194	-0.557
8	0.6	0.538	1.084	1.317	0.997	0.148	-0.920	-1.754	-1.5	-1.466
9	1.2	-0.901	0.461	[illegible]	1.383	0.516	-0.536	-1.086	-0.880	-0.198
10	1.8	-1.963	-1.250	0.607	1.354	0.558	-0.111	0.283	0.538	-0.493
11	2.4	-0.304	-1.9[illegible]	-0.717	0.972	0.259	0.206	1.559	0.588	-1.539
12	3	0.474	-0.821	-1.779	0.397	-0.275	0.332	[illegible]	-0.099	0.004
13	3.6	-0.500	[illegible]	-1.[illegible]	-0.170	-0.859	0.275	1.302	0.195	1.537
14	4.2	0.408	0.559	-0.951	-0.563	-1.288	0.120	-0.004	1.368	0.486
15	4.8	1.985	-0.117	0.485	-0.698	-1.[illegible]	-0.013	-1.133	[illegible]	0.204
16	5.4	0.792	0.231	1.562	-0.594	-1.189	-0.025	-1.468	-0.021	1.472
17	6	-0.546	1.406	1.692	-0.351	-0.696	0.118	-0.973	-1.550	0.549

Рисунок 1 – Результат условного форматирования

Выделяем эти ячейки и создаем формулу в диалоговом окне **Условное форматирование** (2003) или **Создание правила форматирования** (2007+) (рисунок *2*).

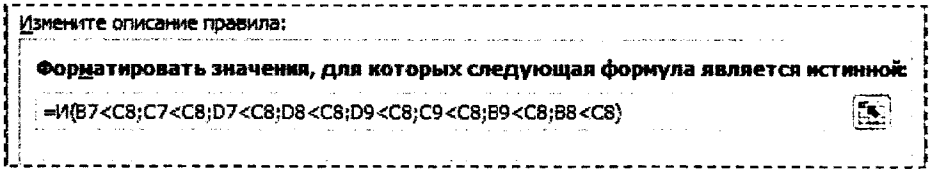


Рисунок 2 – Правило для условного форматирования

Формула составляется относительно левой верхней ячейки выделенного диапазона (C8) и начинается со знака «=». После знака «=» используется встроенная функция И(условие1;условие2;...),которая позволяет проверить одновременное выполнение всех содержащихся в ней условий. Кроме формулы необходимо задать формат ячейки (например, цвет фона ячейки – красный), который будет применен к ней в случае, если формула является истинной. Созданное правило после применения автоматически тиражируется во все ячейки выделенного диапазона. Та как каждая ячейка имеет свои восемь соседних ячеек, то в формуле правила все ссылки должны быть относительными.

На рисунке *2* показано правило для случая, когда необходимо найти ячейки с числом большим *чисел, содержащихся в соседних восьми ячейках* (локальный максимум). Для случая, когда необходимо найти ячейки с числом *меньшим чисел, содержащихся в соседних восьми ячейках* (локальный минимум), в формуле знаки «<» нужно поменять на знаки «>».

Построение поверхностной диаграммы

Чтобы создать поверхностную диаграмму (рисунок *3*), необходимо:

- определить данные, по которым будет построена диаграмма (рисунок *1*);
- выделить диапазон ячеек, содержащий эти данные (B7:J17);
- в Excel 2003 выполнить команду **Вставка/Диаграмма...** и в появившемся мастере диаграмм выбрать следующие настройки:
  - шаг 1. Выбрать тип диаграммы **Поверхность** вид **Поверхность**;
  - шаг 2. На вкладке **Диапазон данных** указать, что ряды данных находятся в столбцах. На вкладке **Ряд** источника данных диаграммы задать диапазон для подписи данных по оси *X* (значения аргумен-

та x – A7:A17) и указать имя каждого ряда (значения аргумента y – для первого ряда – B6, для второго ряда – C6 и т.д.);
  - шаг 3. На вкладке **Заголовки** параметров диаграммы задать название диаграммы «Функция Z(x,y)», а на вкладке **Легенда** убрать флажок **Добавить легенду**;
  - шаг 4. Поместить диаграмму на отдельном листе **График_Z**;
- в Excel 2007+ выполнить команду **Вставка/Диаграммы/Создать диаграмму** (CA)) и выбрать тип диаграммы **Поверхность** вид **Поверхность**. Далее выделить полученный объект:
  - на контекстной вкладке **Конструктор** в группе **Данные** выбрать команду **Выбрать данные**. На форме **Выбор источника данных** изменить **Подписи горизонтальной оси** – A7:A17. Также изменить **Элементы легенды** – имя для первого ряда – B6, для второго ряда – C6 и т.д.;
  - на контекстной вкладке **Макет** в группе **Подписи** выбрать команду **Название диаграммы/Над диаграммой**. Указать название диаграммы – «Функция Z(x,y)»;
  - на контекстной вкладке **Макет** в группе **Подписи** выбрать команду **Легенда/Нет**;

Рисунок 3 – Поверхностная диаграмма

  - на контекстной вкладке **Конструктор** в группе **Расположение** выбрать команду **Переместить диаграмму**. В открывшемся окне **Перемещение диаграммы** указать место размещения диаграммы – на отдельном листе **График_Z**.

Надстройка «Поиск решения» для уточнения локальных экстремумов

С помощью надстройки **поиск решения** уточним, например, один локальный минимум функции Z(x,y), протабулированные значения которой показаны на рисунке *1*. Пусть это будет локальный минимум в координатах X = 4.8, Y = 0. В этом случае точное значение локального минимума будет находиться в диапазоне 4.2..5.4 по оси x и в диапазоне -0.75..075 по оси y. Отмеченные данные занесем в отдельные ячейки рабочего листа (рисунок *4*). В ячейку B9 вставляем функцию Z(x,y), в которой вместо переменных X и Y делаем ссылки на ячейки B7 и B8 соответственно.

В диалоговом окне **Поиск решения** (рисунок *5*) задаем целевую ячейку B9 (в целевой ячейке обязательно должна находиться формула). Поскольку наша цель – минимизировать значение в этой ячейке, устанавливаем переключатель **Равной: минимальному значению**. Затем определяем изменяемые ячейки, от которых зависит результат в целевой ячейке B9. Это ячейки B7 и B8. Далее задаем ограничения по нажатию кнопки **Добавить** (B7≥B2 и B7≤B3, т.е. найденное значение X должно находиться в диапазоне 4.2..5.4; B8≥B4 и B8≤B5, т.е. найденное значение Y должно находиться в диапазоне -0.75..075). Заданные ограничения отобразятся в списке **Ограничения**.

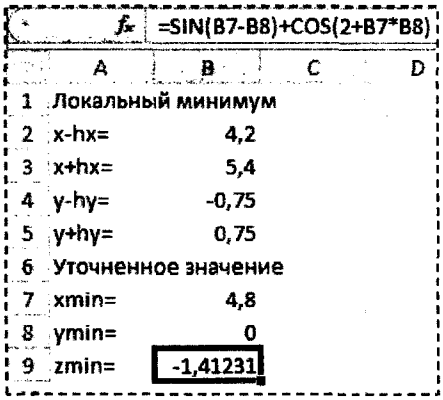


Рисунок 4 – Исходные данные для определения локального минимума

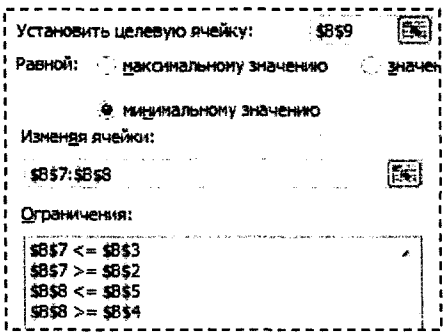


Рисунок 5 – Окно Поиск решения

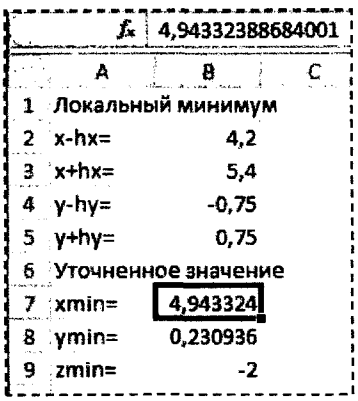


Рисунок 6 – Результат работы процедуры Поиск решения

Чтобы начать процесс решения задачи, необходимо щелкнуть по кнопке **Выполнить**. В строке состояния будет отображаться ход решения задачи. Через некоторое время отобразится информация о том, что решение найдено. В ячейках B7 и B8 появится значение X и Y, при котором функция имеет минимальное значение на участке $x \in [4.2..5.4]$ и $y \in [-0.75..075]$. В ячейке B9 – значение локального минимума функции Z(x,y).

Значение локального максимума определяется аналогично. Все отличие состоит в том, что в отдельные ячейки заносится информация о локальном максимуме и в диалоговом окне **Поиск решения** переключатель устанавливается в положение **Равной: минимальному значению**.

ЯЗЫК ПРОГРАММИРОВАНИЯ VISUAL BASIC FOR APPLICATIONS В MICROSOFT EXCEL

Visual Basic for Applications (VBA) – язык программирования, встроенный в линейку приложений Microsoft Office и предназначенный для их автоматизации. Благодаря его сравнительной лёгкости освоения, офисные приложения могут создавать даже пользователи, не программирующие профессионально.

Обычно программный код VBA организуется в виде совокупности процедур (макросов), каждая из которых предназначена для решения определенной задачи. Наряду с процедурами, в языке VBA также могут быть созданы и собственные функции, которые затем будут использоваться в процедурах.

Безопасность макросов

Чтобы включить макросы в Excel 2003, необходимо выбрать пункт главного меню **Сервис/Макрос/ Безопасность**. В открывшемся окне **Безопасность** установить переключатель **Уровень безопасности** в положение **Средняя** или **Низкая**.

В Excel 2007+ параметры безопасности макросов можно изменять так:

- На вкладке **Разработчик** в группе **Код** нажмите кнопку **Безопасность макросов**. Если вкладка **Разработчик** не отображается, нажмите **кнопку Microsoft Office**, щелкните **Параметры Excel**, а затем в категории **Основные** в разделе **Основные параметры работы с Excel** выберите параметр **Показывать вкладку «Разработчик» на ленте**.
- В категории **Параметры макросов** в разделе **Параметры макросов** выберите вариант **Отключить все макросы с уведомлением** (аналог Excel 2003 – **Средняя**) или **Включить все макросы** (аналог Excel 2003 – **Низкая**).

Если установлена опция **Средняя** (Excel 2003), то при открытии файла в диалоговом окне необходимо нажать кнопку **Не отключать макросы**.

Если установлена опция **Отключить все макросы с уведомлением** (Excel 2007+), то при открытии рабочих книг, содержащих макросы, макросы отключаются и Excel выводит предупреждение системы безопасности о том, что макрос отключен. Если вы хотите включить макросы, щелкните сначала на кнопке **Параметры** в строке предупреждения. Затем в открывшемся диалоговом окне **Параметры безопасности Microsoft Office** установите переключатель **Включить это содержимое** и щелкните на кнопке **ОК**.

Очень важно после этого закрыть и снова открыть данный файл. Без этого действия макросы будут попрежнему отключены. Установленный уровень безопасности распространяется на все файлы Excel.

Открыть и сохранить макросы VBA

Для открытия редактора макросов VBA в Excel 2003 необходимо выбрать пункт меню **Сервис/Макрос/Редактор Visual Basic**, в Excel 2007+ – вкладку **Разработчик** в группе **Код** команду **Visual Basic**. В обеих системах эти действия можно заменить комбинацией клавиш **Alt+F11**.

Рабочие книги Excel 2003, содержащие макросы, сохраняются в файлах с расширением **XLS**, а рабочие книги Excel 2007+ – в файлах с расширением **XLSM**. Если вы попытаетесь сохранить такую рабочую книгу Excel 2007+ в формате **XLSX**, заданном по умолчанию, то Excel сохранит вашу рабочую книгу без макросов. Если вы хотите сохранить макросы, нажмите **кнопку Microsoft Office**, выберите пункт **Сохранить как** и в диалоговом окне **Сохранение документа** выберите тип файла **Книга Excel с поддержкой макросов (*.xlsm)**.

Создание модулей и макросов

Перед тем как вводить код программы, необходимо вставить модуль в рабочую книгу. Если рабочая книга уже имеет лист модуля, его можно использовать для нового макроса.

Чтобы вставить новый модуль, выполните следующие действия.

- Выберите в окне **Project** рабочую книгу, с которой вы работаете в текущий момент.
- Выберите команду **Insert/Module**. В рабочей книге появится новый (пустой) модуль.

Каждый элемент в окне **Project** имеет определенное количество свойств. Для модификации свойств используется окно **Properties**. Чтобы отобразить окно **Properties**, выберите команду **View/Properties Window** или нажмите клавишу **F4**. В окне **Properties** отображается список свойств выбранного элемента.

Для изменения имени модуля необходимо в окне **Project** выбрать нужный модуль, а затем в окне **Properties** в свойстве **Name** указать его новое имя. Чтобы удалить ненужный модуль, нужно щелкнуть по нему правой клавишей мыши в окне **Project** и в контекстном меню выбрать пункт **Remove Module...**. Далее в диалоговом окне нажать на кнопку **Нет**.

VBA-макрос (или процедура) может быть двух типов: подпрограммой и функцией.

Макрос-подпрограмма – это нечто вроде новой команды, которая может быть выполнена либо пользователем, либо другим макросом. В рабочей книге Excel может содержаться произвольное число подпрограмм.

Подпрограммы всегда начинаются с ключевого слова **Sub**, после которого следует имя макроса (у каждого макроса должно быть уникальное имя), а затем – пара круглых скобок. (В этих скобках задаются аргументы, но если у подпрограммы нет аргументов, они остаются пустыми.) Оператор **End Sub** говорит об окончании подпрограммы. Строки, заключенные между этими двумя операторами, составляют тело процедуры или текст макроса.

Вторым типом процедуры VBA является функция. Функция всегда возвращает единственное значение (так же, как и обычная функция рабочей таблицы), которое внутри процедуры присваивается имени функции.

Функции подобны подпрограммам. Обратите внимание на то, что функция начинается ключевым словом **Function** и заканчивается оператором **End Function**.

Таблица 1 – Пример создания процедур в VBA

Тип процедуры	Синтаксис	Пример

Функция	[Public | Private] **Function** *имя* ([*список_арг*]) [As *тип*]
	[*операторы*]
	имя = выражение
	End Function
		`Public Function Fun_F(x)`
		`'Процедура функции`
		
		`Fun_F = 2 * x + 3`
		`End Function`

Подпрограмма	[Public | Private] **Sub** *имя* ([*список_арг*])
	[*операторы*]
	End Sub
		`Public Sub Tab_F()`
		`'Процедура подпрограммы`
		
		`End Sub`

Примеры процедур, показанные в таблице 1, содержат комментарии (например, «'Процедура функции»). Комментарии – это заметки для того, кто составляет код процедуры, VBA их игнорирует. Строка комментариев начинается с апострофа. Комментарий можно поместить после любого оператора. Другими словами, если VBA встречает апостроф, он игнорирует остальной текст этой строки.

Создать процедуру типа **Sub** или **Function** можно вручную либо автоматически.

- Установите курсор в окне **Code**.
- Выберите пункт меню **Insert/Procedure...**
- В появившемся диалоговом окне **Add Procedure** в поле **Name** укажите имя процедуры (без пробелов и скобок), а в поле **Type** укажите тип процедуры **Sub** (рисунок *7*) или **Function** (рисунок *8*).

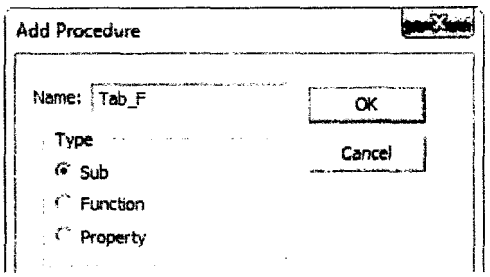


Рисунок 7 – Форма для создания процедуры Sub

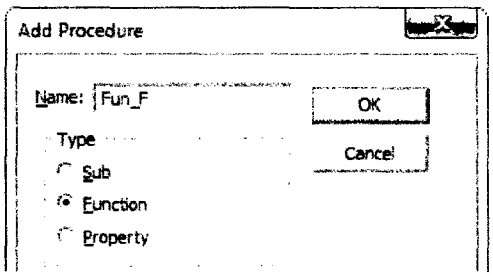


Рисунок 8 – Форма для создания процедуры Function

Запуск макросов

Перед запуском процедуры функции обязательно нужно задать в скобках значения всех аргументов. Так, функцию VBA (Fun_F(x) в таблице 1) можно запустить:

- из другой процедуры VBA (например, y = Fun_F(4));
- из формулы из ячейки рабочего листа точно так же, как и встроенные функции рабочих листов Excel (рисунок *9*);
- из окна **Immediate** (см. ниже **Диалоговое окно Immediate**).

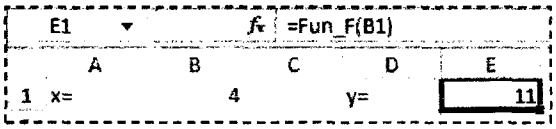


Рисунок 9 – Использование функции Fun_F(x) на рабочем листе

Запуск подпрограмм осуществляется другими способами:

- из рабочего листа Excel. В этом диалоговом окне **Макрос** выберите из списка имя нужной подпрограммы и щелкните на кнопке **Выполнить**. Для открытия диалогового окна **Макрос** выполните команду:
  –Excel 2003: **Макросы...** из пункта главного меню **Сервис/Макрос**.
  –Excel 2007+: **Макросы** на вкладке **Разработчик** в группе **Код**.
- из окна **Code** редактора VBA. Поместите курсор где-нибудь в коде макроса и выполните одно из предложенных действий:
  –выберите пункт главного меню **Run/Run Sub**;
  –нажмите на пиктограмму **Run Sub** панели инструментов **Standard**;
  –нажмите клавишу **F5**;
- из другой процедуры VBA (например, Call Tab_F или просто Tab_F);
- из окна **Immediate** (см. ниже **Диалоговое окно Immediate**).

Пошаговое выполнение макросов

При отладке программы часто возникает необходимость проследить за тем, как выполняются отдельные операторы в отлаживаемой программе или же какие значения принимают те или иные переменные на различных этапах выполнения программы. Для достижения этих целей в редакторе VBA реализован режим прерывания.

Перевести программу в режим прерывания в некоторый, заранее известный момент ее выполнения можно с помощью установки **точки останова**. Чаще всего **точку останова** помещают в строке программного кода непосредственно перед оператором, который, по вашему предположению, является причиной возникновения ошибки. Как только по ходу выполнения программы достигается оператор, которому назначена **точка останова**, работа программы приостановится (шаг 3 таблицы 2). Теперь можно будет проверить текущие значения любых переменных (шаг 6 таблицы 2). Последующее нажатие **F8** позволяет выполнять процедуру по шагам, в порядке естественного выполнения операторов (шаги 4 и 5 таблицы 2). Если очередной оператор является вызовом процедуры типа **Sub** или **Function**, то применение **F8** приведет к открытию текста вызываемой процедуры в окне редактирования кода и позволит пройти эту процедуру по шагам, увидев результаты всех выполняемых в ней действий.

Для размещения или отмены точки останова в программном коде выполните одно из следующих действий (шаг 2 таблицы 2):

- Щелкните на серой полоске, расположенной в левой части окна редактирования кода, напротив требуемой строки программного кода.
- Поместите курсор в нужную строку кода и нажмите **F9**.

Таблица 2 – Элементы пошагового выполнения макроса

Шаг 1 – Создали процедуру My_T() типа Sub	Шаг 2 – Установили **точку останова** с помощью **F9**	Шаг 3 – Запустили процедуру с помощью **F5**
Public Sub My_T() x = 3 y = 8 + x Debug.Print y End Sub	Public Sub My_T() x = 3 y = 8 + x Debug.Print y End Sub	Public Sub My_T() x = 3 y = 8 + x Debug.Print y End Sub
Шаг 4 – Перешли к следующему оператору с помощью F8	**Шаг 5 – Перешли к следующему оператору с помощью F8**	**Шаг 6 – Удерживаем указатель мыши над переменной *y***
Public Sub My_T() x = 3 y = 8 + x Debug.Print y End Sub	Public Sub My_T() x = 3 y = 8 + x Debug.Print y End Sub	Public Sub My_T() x = 3 y = 8 + x y = 11 Debug.Print y End Sub

Количество размещаемых точек останова неограниченно, поэтому в разных строках программы можно разместить столько точек останова, сколько сочтете необходимым. Единственное условие – нельзя размещать точки останова в строках комментариев и строках с операторами, которые VBA на самом деле не выполняет, например, в строках с объявлениями переменных.

В редакторе VBA реализован механизм автоматической подсказки, которая позволяет увидеть текущее значение любой переменной в программе. Для этого, когда программа находится в режиме прерывания, задержите на секунду указатель мыши на имени интересующей вас переменной, и на экране появится окно подсказки – небольшой прямоугольник с именем и текущим значением переменной в нем (шаг 6 таблицы 2).

При выполнении макросов могут возникнуть ошибки, отличные от синтаксических, – ошибки выполнения. Такие ошибки можно обнаружить только в процессе выполнения процедуры. Обычно такие ошибки выполнения приводят к остановке процедур VBA. При этом отображается диалоговое окно с сообщением, в котором указывается номер ошибки и краткое ее описание (рисунок ***10***).

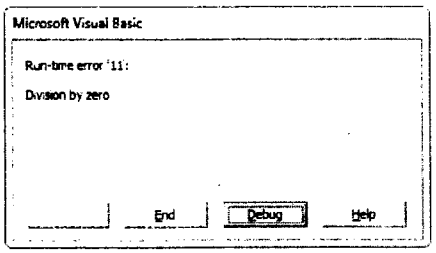


Рисунок 10 – Сообщение об ошибке

В этом окне кнопка **End** прекращает выполнение процедуры. Щелчок на кнопке **Debug** переводит выполняющуюся процедуру в режим прерывания, что позволяет в окне кода редактора VBA внести исправления в тот оператор, где возникла данная ошибка выполнения, а затем продолжить либо завершить (**Run/Reset** или на **Reset** панели инструментов) выполнение программы.

Диалоговое окно Immediate

Помимо средств прерывания и пошагового выполнения, отладчик редактора VBA предоставляет диалоговое окно **Immediate** (Окно немедленного выполнения команд), призванное дополнительно упростить процесс отладки программ и предоставить дополнительные инструменты поиска ошибок.

Чтобы открыть окно **Immediate**, выполните одно из следующих действий:

- выберите команду меню **View/Immediate Window**;
- щелкните на кнопке **Immediate Window** панели инструментов **Debug**;
- нажмите **Ctrl+G**.

Окно **Immediate** редактора VBA позволяет контролировать значения переменных и выражений, проводить вычисления, изменять значения переменных и проверять результаты выполнения функций. Кроме того, в окне **Immediate** можно выполнять отдельные макросы. Так, для запуска процедуры типа **Sub** достаточно указать ее имя и через пробел значения аргументов при их наличии (макрос My_T() из таблицы 3). Для запуска процедуры типа **Function** в окне **Immediate** используется оператор **Print** либо знак «?» (макрос Fun_F(x) из таблицы 3). Введите этот оператор в окне **Immediate**, затем укажите имя функции и в скобках значения аргументов. После завершения ввода нажмите клавишу **Enter**.

Таблица 3 – Результаты выполнения макросов в окне **Immediate**

Макрос	**Function** Fun_F(x) из таблицы 1	**Sub** My_T() из таблицы 2
Результат выполнения в окне **Immediate**	Immediate ?Fun_F(4) 11	Immediate My_T 11

При необходимости можно направить в окно **Immediate** вывод промежуточных значений переменных и выражений (макрос My_T() из таблицы 3) непосредственно при выполнении программы. Для этого разместите в подходящих строках программы операторы, вызывающие метод **Debug.Print** (шаг 1 таблицы 2).

Ввод-вывод данных на рабочий лист

VBA – это объектно-ориентированный язык программирования. Это означает, что основными его элементами являются объекты – рабочие книги (**Workbook**), рабочие листы (**Worksheet**), диапазоны (**Range**, **Cells**). диаграммы (**Chart**) и др. Эти объекты расположены в иерархическом порядке.

Набор однотипных объектов называется коллекцией. Например, коллекция всех объектов рабочих листов (**Worksheet**) называется **Worksheets**. Обратиться к отдельному объекту коллекции можно, используя порядковый номер или ссылки. Например, если в рабочей книге есть три рабочих листа: **Лист1**, **Лист2** и **Лист3**, то к объекту коллекции рабочих листов **Лист2** можно обратиться так – **Worksheets("Лист2")**.

Для выбора объекта, с которым предстоит дальнейшая работа, необходимо использовать метод **Select**. Метод – это действие, примененное к объекту. Так, например, чтобы выбрать рабочий лист **Лист2** (сделать его активным), необходимо записать в макросе следующую команду – **Worksheets("Лист2").Select**.

Для обращения к определенной ячейке рабочего листа необходимо использовать свойства **Range**("*адрес ячейки*") или **Cells**(*номер строки ячейки*, *номер столбца ячейки*). С помощью этих свойств можно передавать значения переменной в ячейку и из ячейки – переменной. Примеры приведены в таблице 4.

Таблица 4 – Примеры использования свойств **Range** и **Cells**

Условие	Присвоить переменной f значение из ячейки **C2** рабочего листа	Вывести на рабочий лист в ячейку **B3** результат выражения 2^2-1
A B C 1 2 4 3 3	f = Range("C2") или f = Cells(2,3)	Range("B3") = 2^2 - 1 или Cells(3,2) = 2^2 - 1
Результат	f = 4	В ячейке B3 число 3

При использовании в выражении без указания координат свойство **Cells** объекта **Worksheet** определяет диапазон, включающий все ячейки данного рабочего листа. Совместное действие **Cells** и метода **Clear** позволяет очистить содержимое всех ячеек активного рабочего листа. Например, **Cells.Clear**.

Диалоговые окна

В VBA есть две функции – **InputBox** и **MsgBox**, которые позволяют отображать простые диалоговые окна. Эти диалоговые окна можно видоизменить несколькими способами, но, конечно, они не могут содержать всех тех опций, которые доступны в созданных пользователем диалоговых окнах.

В краткой форме функция VBA **InputBox**, предназначенная для ввода значений переменным, имеет следующий синтаксис:

InputBox(*сообщение* [,*заголовок*] [,*по_умолчанию*])

Назначение аргументов:

- *сообщение* – текст, отображаемый в окне (обязательный аргумент);
- *заголовок* – текст, который появляется в строке заголовка окна (необязательный аргумент);
- *по_умолчанию* – значение, отображаемое в окне ввода по умолчанию (необязательный аргумент).

В следующем примере функция **InputBox** предлагает пользователю ввести значение переменной *x*.

```
x = InputBox("введите значение x", "ввод значений")
```

При выполнении этого оператора VBA Excel выводит на экран диалоговое окно, показанное на рисунке *11*. Заметьте, что в данном примере использованы только два первых аргумента, параметр *по_умолчанию* не указан. Когда пользователь введет некоторое значение и щелкнет на кнопке OK, это значение будет присвоено переменной *x*. Функция **InputBox** всегда возвращает строку (текст), поэтому может возникнуть необходимость в преобразовании результата в число. Для этого воспользуйтесь функцией **Val**. Функция **Val** – преобразует текст в число.

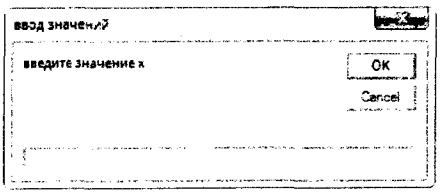


Рисунок 11 – Диалоговое окно функции InputBox

```
x = Val(InputBox("введите значение x", "ввод значений"))
```

Если в качестве параметра *по_умолчанию* задать значение 3.5 (т.е. x=3.5), то код будет выглядеть так:

```
x = Val(InputBox("введите значение x", "ввод значений", "3.5"))
```

Функция VBA **MsgBox** – это весьма удобное средство для того, чтобы отобразить на экране информацию и попросить пользователя сделать выбор, щелкнув на одной из предложенных кнопок. Эта функция используется для того, чтобы вывести значение переменной. Краткий синтаксис функции **MsgBox** выглядит следующим образом:

MsgBox(*сообщение* [,*кнопки*] [,*заголовок*])

Описание аргументов:

- *сообщение* – текст, отображаемый в окне сообщения (обязательный аргумент);
- *кнопки* – коды кнопок, которые отобразятся в окне сообщения (необязательный аргумент);
- *заголовок* – текст, который появляется в строке заголовка окна сообщения (необязательный аргумент).

Функцию **MsgBox** можно вызывать в виде отдельного оператора, не присваивая ее результат какой-нибудь переменной. Если функция вызывается самостоятельно, то аргументы не нужно заключать в круглые скобки. В следующем примере функция **MsgBox** выводит на экран сообщение о переменной *x* = 3.5 (рисунок *12*):

```
MsgBox "значение переменной x = " + Str(x)
```

Здесь в тексте *сообщения* использованы знак «+» для соединения (склеивания) двух частей текста и функция **Str**, которая преобразует числовое значение переменной *x* в текст.

Настраивать окна сообщений можно с помощью соответствующих встроенных констант VBA, которые используются в качестве аргументов **MsgBox**. Встроенные константы VBA, отвечающие за тип пиктограммы в диалоговом окне сообщений, перечислены в таблице 5.

Таблица 5 – Константы в **MsgBox**

Константа	Знак
vbCritical	
vbQuestion	
vbExclamation	
vbInformation	

В следующем примере зададим в операторе **MsgBox** все три аргумента.

```
MsgBox "значение переменной" + vbCr + "x = " + Str(x), vbInformation, "Вывод результата"
```

Результат приведенного оператора отображен на рисунке *13*. В первом аргументе *сообщение* использована встроенная константа **vbCr**, которая позволяет разбить текст на две части и перенести вторую часть на следующую строку.

Если в качестве аргумента используется текст, то он заключается в кавычки.

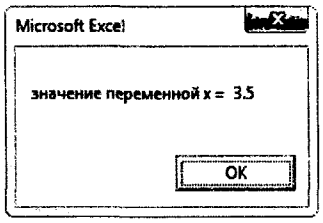


Рисунок 12 – Диалоговое окно оператора MsgBox

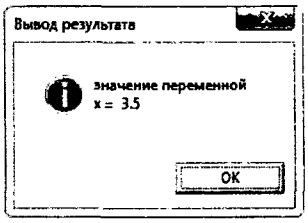


Рисунок 13 – Диалоговое окно оператора MsgBox

Оператор IF

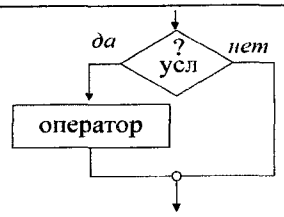
Оператор **If...Then** – один из важнейших управляющих операторов. Он организует выполнение заданного блока операторов, если при вычислении указанного условного выражения будет получено значение ИСТИНА, и не делает ничего, если будет получено значение ЛОЖЬ. Такая конструкция **If...Then** записывается в одной строке, и завершающий оператор **End If** для нее не требуется (таблица 6).

Однако в операторе **If...Then** может выполняться не только один оператор, но и блок последовательно следующих операторов. В этом случае проверяемое условие и выполняемые операторы записываются в отдель-

ных строках. Если условное выражение имеет значение ИСТИНА, выполняемая группа операторов начинается с первого оператора, стоящего после ключевого слова **Then**, и включает все последующие операторы, вплоть до оператора завершения **End If** (таблица 6).

Таблица 6 – Примеры краткой формы разветвляющая структуры

Блок-схема	Оператор	однострочный	многострочный

	Синтаксис	If *условие* Then *оператор*	If *условие* Then
			операторы
			End If

	Пример	If x > 5 Then MsgBox "x больше 5"	If x > 5 Then
			MsgBox "x больше 5"
			End If

Для более сложных ситуаций, когда на основании некоторого условия необходимо выбрать одну из двух различных последовательностей операторов, используется оператор **If...Then...Else**. Как и оператор **If...Then**, конструкция **If...Then...Else** может быть однострочной (таблица 7) и многострочной (таблица 8).

Таблица 7 – Пример полной формы разветвляющая структуры (однострочный оператор **If**)

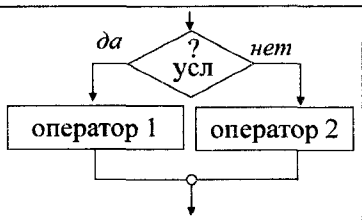
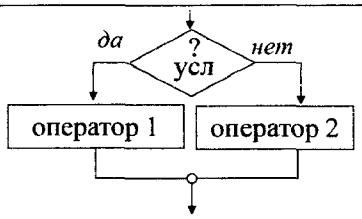
Блок-схема	Оператор	однострочный
	Синтаксис	If *условие* Then *оператор 1* Else *оператор 2*
	Пример	If x > 5 Then MsgBox "x больше 5" Else MsgBox "x не больше 5"

Таблица 8 – Пример полной формы разветвляющая структуры (многострочный оператор **If**)

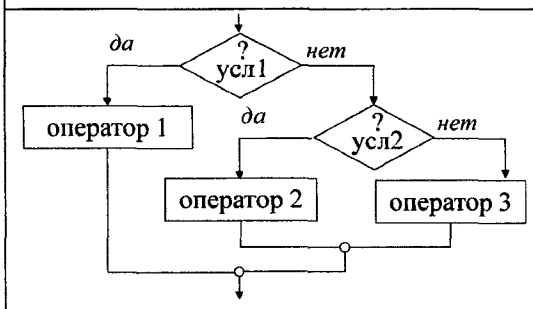
Блок-схема	Оператор	Синтаксис	Пример

	многострочный	If *условие* Then
		оператор 1
		Else
		оператор 2
		End If
			If x > 5 Then
			MsgBox "x больше 5"
			Else
			MsgBox "x не больше 5"
			End If

Часто при написании программ возникают такие ситуации, когда приходится проверять не одно, а два или больше условий. В этом случае можно поместить один оператор **If...Then** внутри другого оператора **If...Then** (или оператор **If...Then...Else** внутри оператора **If...Then...Else**), что называется вложением операторов. Вложенные операторы следует использовать тогда, когда для принятия решения необходимо проверить дополнительное условие, но при этом первое условие должно оказаться истинным. Если же первое условие не выполнено, то будут выполняться операторы, следующие за ключевым словом **Else**.

Таблица 9 – Пример многострочного оператора **If** с вложенным оператором **If**

Блок-схема	Синтаксис	Пример

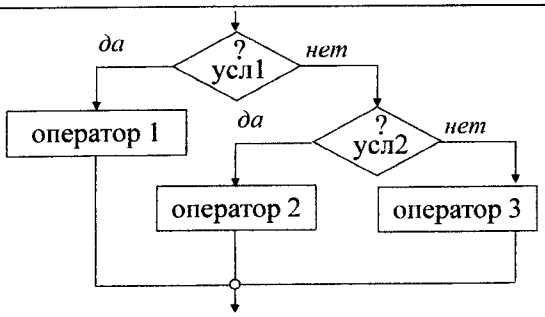
	If *условие 1* Then
	оператор 1
	Else
	If *условие 2* Then
	оператор 2
	[Else
	оператор 3]
	End If
	End If
		If x > 5 Then
		MsgBox "x больше 5"
		Else
		If x < 5 Then
		MsgBox "x меньше 5"
		Else
		MsgBox "x равно 5"
		End If
		End If

Эквивалентом вложенных операторов является оператор **If...Then...ElseIf**, в котором используется ключевое слово **ElseIf**. С помощью ключевого слова **ElseIf** в одном условном операторе можно проверить несколько условий. Синтаксис этого условного оператора выглядит так, как показано в таблице 10.

Ключевое слово **Else** является необязательным, но если оно все же присутствует, то должно быть записано в конструкции оператора последним.

Таблица 10 – Пример многострочного расширенного оператора **If**

Блок-схема	Синтаксис	Пример

	If *условие 1* Then
	оператор 1
	ElseIf *условие 2* Then
	оператор 2
	[Else
	оператор 3]
	End If
		If x > 5 Then
		MsgBox "x больше 5"
		ElseIf x < 5 Then
		MsgBox "x меньше 5"
		Else
		MsgBox "x равно 5"
		End If

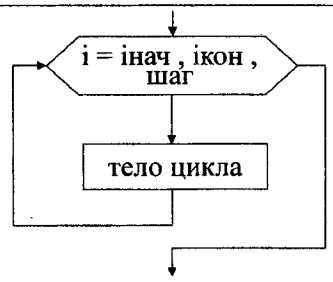
Ключевое слово **Elself** может повторяться в конструкции сколько угодно раз. При этом выполнение заданных условий проверяется строго последовательно, в том порядке, в каком они записаны. И как только истинное условие (ИСТИНА) будет найдено, выполняется соответствующий ему блок операторов, после чего выполнение данного условного оператора прекращается.

Операторы цикла For...Next, Do...Loop и While...Wend

Под циклом мы будем понимать такой оператор или последовательность операторов, которые по ходу выполнения программы могут при необходимости выполняться многократно.

Оператор **For...Next** относится к операторам создания циклов простейшего типа. В программе на языке VBA с его помощью блок операторов выполняется заданное количество раз, известное до начала выполнения цикла. Пример такого оператора приведен в таблице 11.

Таблица 11 – Пример оператора цикла **For...Next**

Блок-схема	Синтаксис	Пример	Результат
	For *i* = *інач* To *ікон* [Step *шаг*] *тело цикла* Next *i*	For i = 0 To 10 Step 2 Debug.Print i ^ 2 Next i	Immediate 0 4 16 36 64 100

В таблице 11 *i* – любая числовая переменная, значение которой меняется в начале каждого цикла (проходе). Параметры *інач* и *ікон* – это числовые выражения, задающие начальное и конечное значения переменной цикла *i*. Числовая переменная *шаг* задает приращение, на которое увеличивается переменная цикла *i* при каждом проходе. Вся фраза с ключевым словом **Step** необязательна. При ее отсутствии интерпретатор VBA (по умолчанию) увеличивает переменную цикла *i* на единицу при каждом выполнении *тела цикла.*

Операторы циклов **Do...Loop** (в отличие от операторов **For...Next**) выполняются не заданное количество раз, а сколь угодно долго – до тех пор, пока не будет выполнено некоторое логическое условие. Язык VBA включает два варианта организации такого цикла: цикл с предусловием и цикл с постусловием.

В цикле с предусловием логическое условие проверяется до начала выполнения цикла. В этом случае, если после оператора цикла **Do** указано ключевое слово **While**, то тело цикла повторяется до тех пор, пока логическое условие, стоящее после **While**, будет истинно (таблица 12). Если указано ключевое слово **Until**, то тело цикла выполняется до тех пор, пока логическое условие будет ложно (таблица 13).

Таблица 12 – Примеры операторов цикла **Do...Loop (While...Wend)** с предусловием ИСТИНА

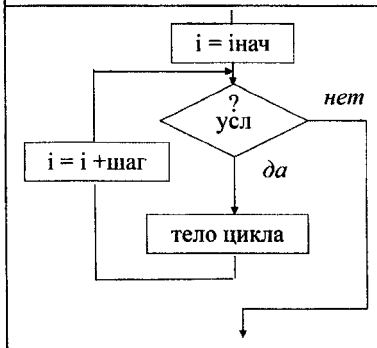
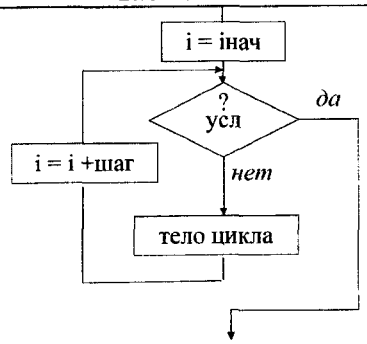
Блок-схема	Синтаксис	Пример	Результат
	*i* = *інач* Do While *условие* *тело цикла* *i* = *i* + *шаг* Loop	i = 0 Do While i <= 10 Debug.Print i ^ 2 i = i + 2 Loop	Immediate 0 4 16 36 64 100
	i = *інач* While *условие* *тело цикла* *i* = *i* + *шаг* Wend	i = 0 While i <= 10 Debug.Print i ^ 2 i = i + 2 Wend	

Таблица 13 – Примеры операторов цикла **Do...Loop** с предусловием ЛОЖЬ

Блок-схема	Синтаксис	Пример	Результат
	*i = інач* Do Until *условие* *тело цикла* *i = i + шаг* Loop	i = 0 Do Until i > 10 Debug.Print i ^ 2 i = i + 2 Loop	Immediate 0 4 16 36 64 100

На практике конструкция оператора **Do While...Loop** наиболее популярна, поэтому в VBA ее часто заменяют оператором цикла **While...Wend**, синтаксис и пример которого приведен в таблице 12.

В цикле с постусловием логическое условие проверяется после выполнения цикла. В этом случае, если после оператора цикла **Loop** указано ключевое слово **While**, то тело цикла повторяется до тех пор, пока логическое условие, стоящее после **While**, будет истинно (таблица 14). Если указано ключевое слово **Until**, то тело цикла выполняется до тех пор, пока логическое условие будет ложно (таблица 15).

Таблица 14 – Примеры операторов цикла **Do...Loop** с постусловием ИСТИНА

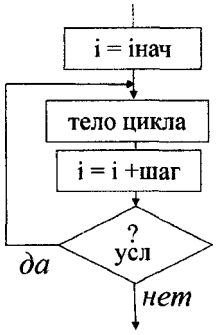
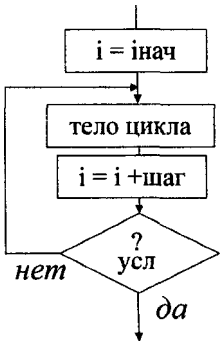
Блок-схема	Синтаксис	Пример	Результат
	*i = інач* Do *тело цикла* *i = i + шаг* Loop While *условие*	i = 0 Do Debug.Print i ^ 2 i = i + 2 Loop While i <= 10	Immediate 0 4 16 36 64 100

Таблица 15 – Примеры операторов цикла **Do...Loop** с постусловием ЛОЖЬ

Блок-схема	Синтаксис	Пример	Результат
	*i = інач* Do *тело цикла* *i = i + шаг* Loop Until *условие*	i = 0 Do Debug.Print i ^ 2 i = i + 2 Loop Until i > 10	Immediate 0 4 16 36 64 100

Создание формы

Пользовательские диалоговые окна создаются в редакторе VBA как экранные формы. Такая экранная форма – форма пользователя – состоит из окна формы (это то самое окно, которое отображается на экране) и нескольких специфических элементов управления, размещенных в этом окне. Все формы, как и любые элементы управления в форме, – это полноценные объекты VBA.

Для создания в проекте VBA новой формы пользователя необходимо активизировать этот проект в окне проектов и либо выбрать команду меню **Insert/UserForm** редактора VBA, либо щелкнуть правой кнопкой мыши в окне **Project** и выбрать команду **Insert/UserForm** в раскрывшемся контекстном меню. При этом новая форма (т.е. заготовка новой формы) появится в специально созданном для нее окне. Рядом с формой будет отображена панель элементов управления **Toolbox** – специальная панель инструментов, содержащая кнопки, с помощью которых соответствующие элементы управления можно разместить в создаваемой форме.

По умолчанию формам пользователя (т.е. соответствующим объектам) автоматически присваиваются имена **UserForml**, **UserForm2** и т.д. Имя формы пользователя можно изменить на более понятное, отражающее назначение формы. Для этого достаточно выбрать требуемую форму в окне проектов, а затем в окне свойств

Properties изменить значение ее свойства **Name**. Если окно свойств не отображено в окне редактора VBA, предварительно нажмите **F4** для вывода этого окна на экран, после чего измените значение свойства **Name** формы.

Для отображения формы на экране используется метод **Show** объекта **UserForm**. Синтаксис вызова метода следующий: **ИмяФормы.Show**. Здесь обозначение **ИмяФормы** представляет имя объекта требуемой формы, содержащееся в свойстве **Name**. Ниже приведен пример процедуры типа **Sub** (она должна находиться в обычном модуле VBA, а не в модуле экранной формы), которая отображает на экране форму **Forml**.

```
Sub Vyvod_form()
   Forml.Show
End Sub
```

Для закрытия окна используется команда **Upload Me** либо **End**, которая обычно записывается в *событие* **Click** элемента управления **CommandButton**.

Событие – это то, что происходит, когда пользователь воздействует на элемент управления экранной формы. Например, щелчок на командной кнопке инициирует событие **Click**, ассоциированное с данной кнопкой. Программное приложение должно иметь процедуры, которые бы выполнялись при наступлении того или иного события. Такие процедуры часто называют *процедурами обработки событий.* Процедуры обработки событий носят имена, в которых название элемента управления объединено с названием события с помощью символа подчеркивания. Например, процедура, которая выполняется после щелчка на кнопке **MyButton,** называется **MyButton_Click**. Если эта кнопка предназначена для закрытия формы **Form1**, то процедура обработки события **Click** для нее будет выглядеть так:

```
Private Sub MyButton_Click()
   Unload Form1
End Sub
```

Для формирования такой процедуры можно выполнить двойной щелчок по кнопке **MyButton**, размещенной на форме **Form1**.

Для перемещения между конструктором формы и процедурами обработки событий элементов управления на панели инструментов окна **Project** слева находятся две кнопки **View Code** и **View Object**

Элементы управления

Элементы управления, которые можно вставить в экранную форму, находятся на панели инструментов **Toolbox**. Чтобы вставить какой-либо элемент управления в экранную форму, щелкните на соответствующем инструменте панели **Toolbox**, затем щелкните на экранной форме. Можно также сначала щелкнуть на нужном инструменте панели **Toolbox**, а затем протащить указатель мыши по экранной форме, указывая размер элемента управления.

Если выделить на форме один из ее элементов управления, то в окне **Properties** будут представлены свойства данного элемента управления. В этом окне слева указываются имена свойств, а справа от каждого имени – его текущее значение.

Таблица 16 – Основные свойства часто используемых элементов управления

Объект	Свойство	Описание
A Label	Caption	Содержит текст, отображаемый в элементе управления
ab| TextBox	Value	Введенный в поле текст
CheckBox	Caption Value	Надпись, отображаемая рядом с данным элементом управления Выбор элемента управления
OptionButton	Caption Value	Надпись, отображаемая рядом с данным элементом управления Выбор элемента управления
Frame	Caption	Надпись, отображаемая вверху элемента управления
CommandButton	Caption	Надпись, отображаемая в центре элемента управления
Image	Picture	Путь к графическому файлу

Свойства элементов управления можно изменять как в процессе разработки экранной формы — вручную, внося изменения в окно свойств элемента управления, так и во время выполнения экранной формы, когда она уже выведена на экран для работы с пользователем, — программно. В последнем случае для изменения свойств элементов управления надо использовать соответствующие операторы VBA. Например:

Имя_объекта.Свойство = Значение

Так команда **TextA.Value=Str(3.5)** поместит в элемент управления **TextBox** с именем **TextA** текст **3.5**.

Скрытая область

Скрытая область – это часть MathCAD-документа, которая присутствует в документе, участвует в расчетах, но не видна на экране. Необходимость в создании такой области возникает в двух случаях: для уменьшения размера документа и для сокрытия информации от посторонних глаз.

Чтобы создать скрытую область, сделайте следующее:

- В главном меню MathCAD выберите пункт **Insert/Area (Вставка/Область)**. На экране появятся две горизонтальные линии с указателями слева. Перетащите их мышью в начало и в конец области, которую надо скрыть.
- Щелкните правой кнопкой мыши в выделенной линиями области. В открывшемся контекстном меню выберите **Collapse (Свернуть)**. Выделенная область исчезнет. Останется лишь одна горизонтальная линия с указателем.

Выражения, спрятанные в скрытой области, продолжают работать в документе, хотя и не видны на экране. Щелкнув правой кнопкой мыши на скрытой области, выберите в контекстном меню пункт **Properties (Свойства)**. В диалоговом окне **Properties (Свойства)** перейдите на вкладку **Area (Область)**. На этой вкладке (рисунок *14*) можно вписать заголовок скрытой области, отметить выделение ее линиями и значком, сделать эту область совсем невидимой.

Чтобы открыть скрытую область, нужно сделать двойной щелчок левой кнопкой мыши на линии с указателем. Скрытая область становится открытой и появляется на экране в обрамлении двух линий с указателями. Чтобы удалить скрытую область, щелкните мышью на одной из линий с указателем, выделив ее, и нажмите клавишу **Del**.

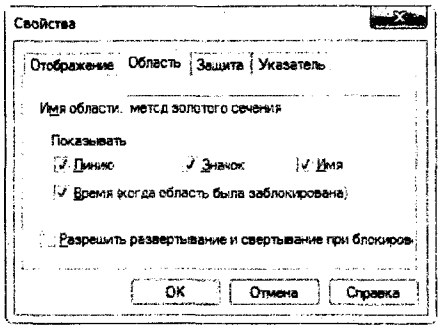


Рисунок 14 – Диалоговое окно Properties (Свойства) для скрытой области

Взаимодействие документов

MathCAD предоставляет возможность объединения нескольких документов в один с помощью ссылки на них. Для этого в основном документе делается ссылка на другой документ. В результате оба документа объединяются в один и работают совместно, хотя второго документа и не видно на экране.

Для создания ссылки надо в главном меню MathCAD выбрать пункт **Insert/Reference... (Вставка/Ссылка...)**. В открывшемся окне, выбрав **Browse... (Обзор...)**, укажите путь к документу, который хотите включить в расчет. Нажмите **OK**. В основном документе появится строка с именем подключенного документа.

Рекомендуется при создании ссылки использовать относительный путь. Указав путь к документу, пометьте флажком пункт **Use relative pass for reference (Использовать для ссылки относительный путь)**. При этом в строке документа, указывающей на ссылку, появится буква (R). В этом случае ссылка останется действующей, даже если вы переместите основной документ в другую папку. Важно лишь, чтобы оба документа находились на одном диске.

Элементы программирования

Элементы программирования в MathCAD служат для создания программных модулей, которые содержат в себе множество строк и фактически являются составными выражениями.

Программный модуль состоит из названия, следующего за ним знака присвоения значения и необходимых выражений в правой части, записанных в столбик и объединенных слева вертикальной чертой.

Для создания программного модуля надо:

- ввести имя программного модуля и, если это необходимо, в скобках указать входные переменные;
- ввести оператор присваивания «:=»;
- щелкнуть на панели **Программирование** по кнопке **Add Line** (или клавишу <]>) столько раз, сколько строк должна содержать программа;
- в появившиеся маркеры ввести нужные операторы, лишние маркеры удалить.

Чтобы создать недостающие маркеры, надо поставить уголок курсора ввода в конец строки, после которой нужно будет вводить новую строку. Клавишей пробела следует выделить полностью всю строку и нажать кнопку **Добавить строку**.

Присваивание значений переменным и константам в программном модуле производится с помощью программного оператора присваивания «←», который вводится с панели **Программирование** нажатием кнопки «←». При создании программы, когда этот знак приходится использовать часто, полезно пользоваться клавишей <{>.

Любой программный модуль представляет собой сочетание обычных математических выражений с операторами условия и цикла.

Условный оператор if

Действие условного оператора **if** состоит из двух частей. Сначала проверяется условие, записанное справа от оператора **if.** Если оно верно, выполняется выражение, стоящее слева от **if,** если не верно, происходит переход к следующей строке программы. Она может содержать новое условие или быть обычным выражением.

Часто встречается условие с двумя вариантами действия: «Если..., то ..., иначе...». Для слова «иначе» на панели **Программирование** имеется оператор **otherwise** (иначе), который вводится так же, как **if.**

На практике условие «если-то-иначе» необходимо вводить в таком порядке:

- в создаваемой программе установить курсор на свободное место ввода, где должен появиться условный оператор;
- на панели **Программирование** щелкнуть на кнопке **if.** В программе появится шаблон оператора с двумя маркерами;
- в правый маркер ввести условие. Пользуйтесь при этом логическими операторами, вводя их с панели **Булевы операторы;**
- слева от оператора **if** ввести выражение, которое должно выполняться, если условие верно. Если при выполнении условия должно выполняться сразу несколько выражений, надо иметь несколько маркеров. Установите курсор ввода на маркер слева от **if** и нажмите **Add Line** столько раз, сколько строк надо ввести. Обратите внимание на то, что при этом изменяется вид условного оператора. Столбик маркеров появится не слева, а под оператором **if;**
- на следующем свободном маркере (на следующей строке) пишут выражение, выполняемое «иначе», если условие не выполняется;
- выделяют курсором выражение так, чтобы уголок курсора ввода был в конце выражения, и на панели **Программирование** нажимают **otherwise.**

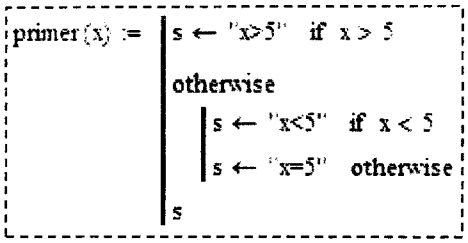


Рисунок 15 – Реализация в MathCAD примера из таблицы 9

Операторы цикла for и while

Панель **Программирование** содержит два оператора цикла, **for** и **while**, работа которых аналогична одноименным операторам VBA.

Чтобы записать цикл **while**, выполните следующие действия:

- Установите курсор на свободное место ввода в программе (справа от вертикальной черты).
- На панели **Программирование** нажмите кнопку **while.** Появится шаблон с двумя местами ввода.
- Справа от слова **while** введите условие выполнения цикла. Обычно это логическое выражение.
- В оставшееся поле ввода (внизу под словом **while)** введите выражение, которое вычисляется в цикле.
- Если в цикле надо вычислять несколько выражений, то вначале установите курсор на место ввода и нажмите кнопку **Add Line** столько раз, сколько строк будет содержать цикл. Затем заполните все места ввода, введя нужные выражения. Удалите лишние места ввода.

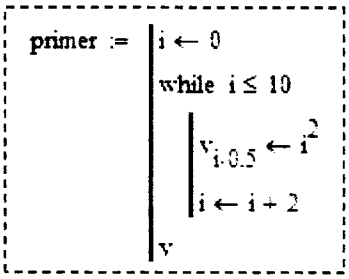


Рисунок 16 – Реализация в MathCAD примера из таблицы 12

В цикле **for** число повторений цикла определяется переменной, задаваемой в начале цикла. Рассмотрим создание такого цикла:

- Установите курсор на свободное место ввода в программе (справа от вертикальной черты).
- На панели **Программирование** нажмите кнопку **for.** Появится шаблон с тремя местами ввода.
- Справа от слова **for** введите имя переменной цикла. После знака ∈ введите диапазон изменения переменной цикла так же, как это делается с помощью дискретной переменной. Переменной цикла может быть ряд чисел, или вектор, или список скаляров, диапазонов, векторов, разделенных запятой.
- В оставшееся поле ввода (внизу, под словом **for**) введите выражение, которое вычисляется в цикле.
- Если в цикле надо вычислять несколько выражений, то вначале установите курсор на место ввода и нажмите кнопку **Add Line** столько раз, сколько строк будет содержать цикл. Затем заполните все места ввода, введя нужные выражения. Удалите лишние места ввода.

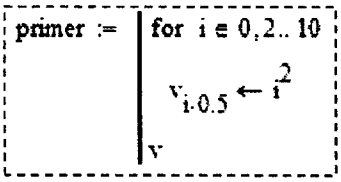


Рисунок 17 – Реализация в MathCAD примера из таблицы 11

СИСТЕМА КОМПЬЮТЕРНОЙ МАТЕМАТИКИ MAPLE

Maple – это система компьютерной математики, содержащая более двух тысяч команд, которые позволяют решать задачи алгебры, геометрии, математического анализа, дифференциальных уравнений, статистики, математической физики. Система Maple предназначена для символьных вычислений, хотя имеет ряд средств и для численного решения дифференциальных уравнений и нахождения интегралов. Обладает развитыми графическими средствами. Имеет собственный язык программирования, напоминающий Паскаль.

Области ввода и вывода

Работа в Maple проходит в режиме сессии – пользователь вводит предложения (команды, выражения, процедуры), которые воспринимаются условно и обрабатываются Maple. Рабочее поле разделяется на три главные части (рисунок 18):

- область ввода – состоит из командных строк. Каждая командная строка начинается с символа «>»;
- область вывода – содержит результаты обработки введенных команд в виде аналитических выражений, графических объектов или сообщений об ошибке;
- область текстовых комментариев – содержит любую текстовую информацию, которая может пояснить выполняемые процедуры.

Текстовые строки не воспринимаются Maple и никак не обрабатываются.

Для того чтобы вставить новую текстовую область, следует на **Панели инструментов** нажать мышью на кнопку [T]. Для вставки новой области ввода, следует нажать мышью на кнопку [>].

Добавление строки в текущую область – **Shift** + **Enter**.

Переопределение области ввода в текстовую область осуществляется с помощью команды **Insert/Text** (Ctrl+T) и только в том случае, когда строка пуста. Обратное действие – **Insert/Maple** Input (Ctrl+M).

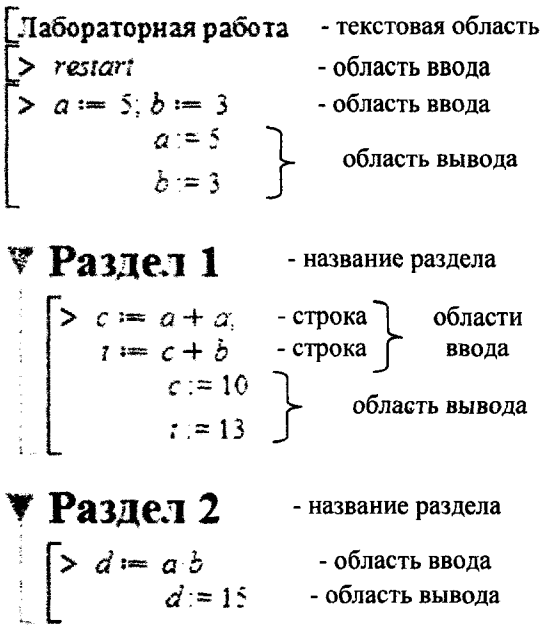


Рисунок 18 – Элементы рабочего поля Maple

Для группировки областей ввода в отдельный раздел необходимо нажать на кнопку [] на **Панели инструментов**. Для разгруппировки – кнопку [].

Удаляется текущая область комбинацией клавиш **Ctrl+Del**.

Основные команды

Maple не пересчитывает документ автоматически. Порядком выполнения команд управляет пользователь. Для запуска последовательного пересчета команд во всем документе необходимо на **Панели инструментов** щелкнуть по кнопке [!!!]. Для последовательного пересчета предварительно выделенных команд используется кнопка [!].

Как правило, в начале документа или перед началом решения новой задачи необходимо давать команду **restart** для очистки оперативной памяти (рисунок 18), т.е. очистить значения всех переменных.

Стандартная команда Maple состоит из имени команды и ее параметров, указанных в круглых скобках: **команда(p1, p2, ...)**. В конце каждой команды может быть знак «;» или «:». Разделитель «;» либо отсутствие его означает, что в области вывода после выполнения этой команды будет сразу виден результат (рисунок 18 и ***19***). Разделитель «:» используется для отмены вывода, то есть когда команда выполняется, но ее результат на экран не выводится (рисунок ***19***).

Символ процента «%» служит для вызова предыдущей команды. Этот символ играет роль краткосрочной замены предыдущей команды с целью сокращения записи (рисунок ***19***).

Для присвоения переменной заданного значения используется знак присвоить «:=» (рисунок 18 и ***19***).

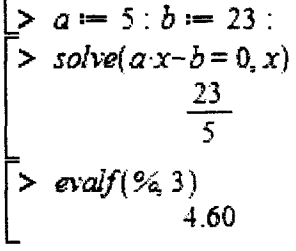


Рисунок 19 – Решение линейного уравнения

Дополнительные команды Maple содержатся в специальных библиотеках подпрограмм, называемых пакетами. Пакеты необходимо подгружать при каждом запуске файла. Загрузить весь пакет можно командой **with(имя_пакета)**, например, **with(plots)**.

Управление результатами вывода

Все вычисления в Maple по умолчанию производятся символьно, то есть результат будет отображаться в виде рациональных дробей и содержать в явном виде иррациональные константы, такие как, e, π и другие. Чтобы получить приближенное значение в виде числа с плавающей точкой, следует использовать команду **evalf(expr,[t])**, где **expr** – выражение, **t** – количество значащих цифр (рисунок ***19***). Также для получения рационального числа в виде числа с плавающей точкой следует дописывать к целой части числа «.0» (рисунок ***20***).

Для подстановки в выражение значений переменных используется функция **eval(expr,[спис_зн])**, где **спис_зн** – список значений переменных (рисунок ***20***).

Команда, над которой будут производиться преобразования, записывается в следующей форме:

имя_переменной:=команда

где **имя_переменной** – произвольное имя выражения;

команда – произвольная команда Maple.

Результат, полученный в области вывода, можно использовать в последующих командах. От вида отображаемого результата будет зависеть форма использующей его команды.

Если результат отображается в виде уравнения, а использовать в дальнейшем нужно только его правую часть, то выделение этой части уравнения осуществляется командой **rhs(выражение)**, выделение левой части – **lhs(выражение)**.

```
> 5·e^3; 5·e^3.0
                    5 e^3
                100.4276846
> ur := y = 25·x^2 − b :
> resh := solve({rhs(ur) = 0}, x)
resh := {x = 1/5 √b}, {x = −1/5 √b}
> x2 := resh[2]
              x2 := {x = −1/5 √b}
> x2 := rhs(resh[2, 1])
                x2 := −1/5 √b
> eval(x2, [b = 25.0])
                -1.000000000
> eval(x2, [b = 30.0])
                -1.095445115
```

Рисунок 20 – Пример управления результатами вывода

Если результат отображается в виде массива уравнений ({ур1},{ур2},{ур3}), а использовать в дальнейшем нужно только правую часть второго уравнения, то перед выделением правой части уравнения необходимо указать в квадратных скобках конкретный элемент массива (рисунок ***20***).

Пользовательская функция

В Maple имеется несколько способов задания пользовательской функции. Рассмотрим два из них.

Первый способ. Определение функции с помощью функционального оператора (рисунок ***21***), шаблон которого находится на левой боковой панели **Expression** рабочего окна Maple. Формат шаблона следующий:

имя_функции:=(аргументы) → выражение

Аргументы перечисляются через запятую.

Второй способ. С помощью команды **unapply** (рисунок ***21***). Формат команды следующий:

имя_функции:=**unapply**(выражение, аргументы)

Обращение к этой функции осуществляется наиболее привычным в математике способом, когда в скобках вместо аргументов функции указываются конкретные значения переменных.

```
> f := (x, y) → 2·x + 5·y
        f := (x, y) → 2x + 5y
> f := unapply(2·x + 5·y, x, y)
        f := (x, y) → 2x + 5y
> resh := solve(z = 2·x + 5·y, z)
              resh := 2x + 5y
> f := unapply(resh, x, y)
        f := (x, y) → 2x + 5y
> f(4, 6)
                38
```

Рисунок 21 – Пример управления результатами вывода

Построение графиков

Для построения графиков функции $f(x)$ одной переменной (в интервале $a \le x \le b$ по оси Ox и в интервале $c \le y \le d$ по оси Oy) используется команда

plot(f(x), x=a..b, y=c..d, параметры),

где **параметры** – параметры управления изображением. Если параметры не указывать, то будут использованы установки по умолчанию.

Основные параметры команды **plot**:

- **title="text"**, где **text** – заголовок рисунка;
- **coords=polar** – установка полярных координат (по умолчанию установлены декартовы);
- **color=n** – установка цвета линии: английское название цвета, например, **yellow** – желтый и т.д.;
- **thickness=n**, где **n**=1,2,3… – толщина линии (по умолчанию **n** = 1);
- **labels=[tx,ty]** – надписи по осям координат: **tx** – по оси *Ox* и **ty** – по оси *Oy*;
- **legend="s"**, где **"s"** – название функционального ряда;
- **axes** – установка типа координатных осей: **axes=normal** – обычные оси; **axes=boxed** – график в рамке со шкалой; **axes=frame** – оси с центром в левом нижнем углу рисунка.

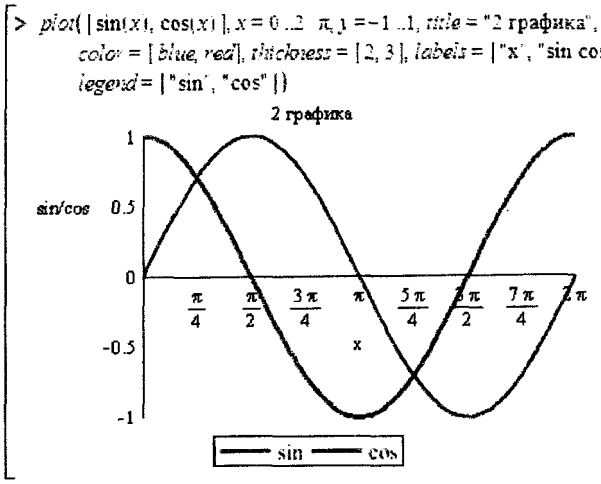


Рисунок 22 – Две функции на одном графике

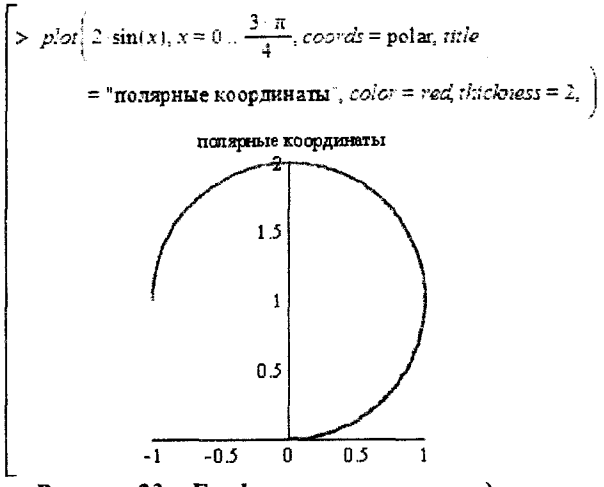


Рисунок 23 – График в полярных координатах

С помощью команды **plot** можно строить помимо графиков функций $f(x)$, заданной явно, также графики функций, заданных параметрически $y=y(t)$, $x=x(t)$:

plot([y=y(t), x=x(t), t=a..b], параметры).

График функции двух переменных $z = f(x, y)$ можно отобразить, используя команду

plot3d(f(x,y), x=x1…x2, y=y1…y2, параметры).

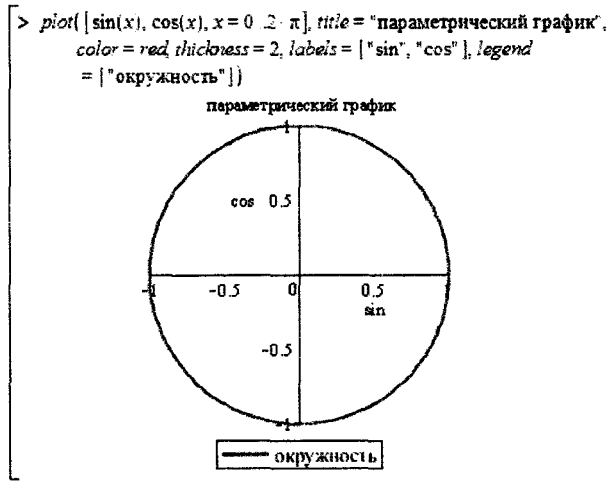


Рисунок 24 – Параметрический график

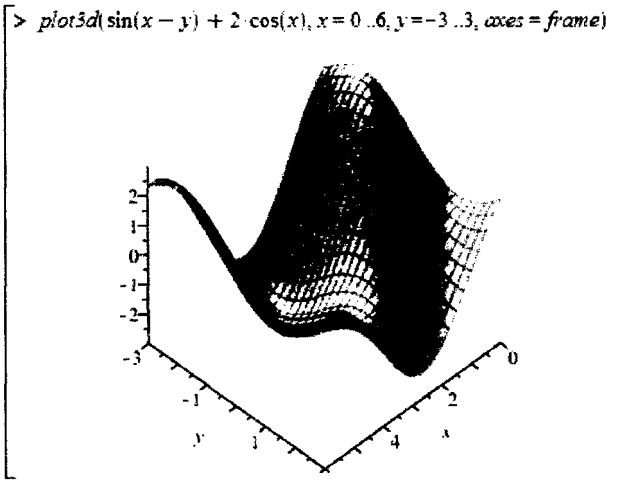


Рисунок 25 – Пространственный график

Функция задана неявно, если она задана уравнением $f(x, y) = 0$. Для построения графика неявной функции используется команда **implicitplot** из графического пакета **plots:**

implicitplot(f(x,y)=0, x=x1..x2, y=y1..y2, параметры).

График линий уровня отображается командой **contourplot**:

contourplot(f(x,y), x=x1..x2, y=y1..y2, параметры).

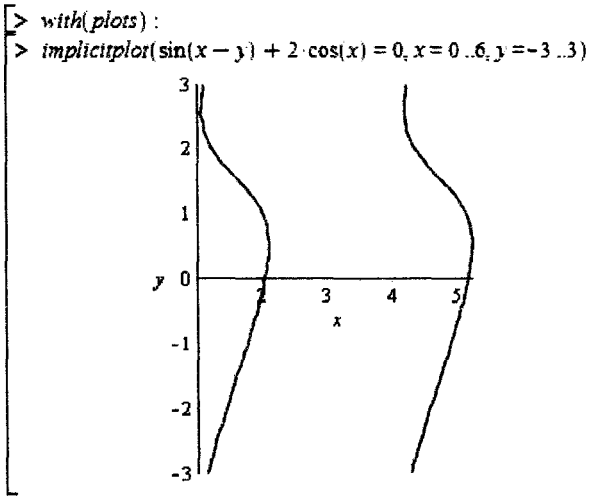


Рисунок 26 – График неявно заданной функции f(x,y) = 0

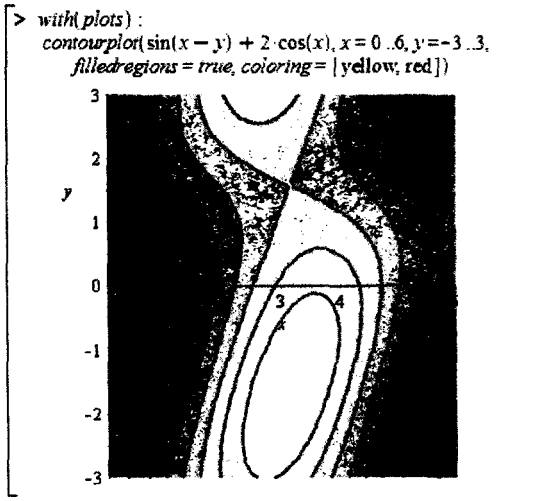


Рисунок 27 – График линий уровня

Дополнительные параметры для команды **contourplot**:

- **filledregions=true** – опция, позволяющая выполнять цветную заливку графика;
- **coloring=n** – установка цветовой гаммы для заливки графика.

Решение СЛАУ

Для решения уравнений в Maple существует универсальная команда **solve(eq, x)**, где **eq** – уравнение, **x** – переменная, относительно которой уравнение надо разрешить (рисунки *19–21*). Системы уравнений решаются с помощью такой же команды **solve({eq1,eq2,...},{x1,x2,...})**, только теперь в параметрах команды следует указывать в первых фигурных скобках через запятую уравнения, а во вторых фигурных скобках перечисляются через запятую переменные, относительно которых требуется решить систему (рисунок *28*).

Для численного решения уравнений, в тех случаях, когда трансцендентные уравнения не имеют аналитических решений, используется специальная команда **fsolve(eq, x)**, параметры которой такие же, как и команды **solve** (рисунок *28*).

```
> solve({2·x + 3·y = 10, 5·x − 2·y = 15}, {x, y})
                    {x = 65/19, y = 20/19}
> fsolve({2·x + 3·y = 10, 5·x − 2·y = 15}, {x, y})
                    {x = 3.421052632, y = 1.052631579}
```

Рисунок 28 – Пример решения уравнений

Дифференцирование и интегрирование

Для вычисления производных в Maple имеются две команды (рисунок *29*):

- прямого исполнения – **diff(f, x)**, где **f** – функция, которую следует продифференцировать, **x** – имя переменной, по которой производится дифференцирование. В качестве аналога команды прямого исполнения можно использовать следующую команду: **f '(x)**.
- отложенного исполнения – **Diff(f, x)**, где параметры команды такие же, как и в предыдущей. Действие этой команды сводится к аналитической записи производной в виде

$$\frac{d}{dx} f(x).$$

Для вычисления производных старших порядков следует в параметрах вместо **x** указать **x$n**, где **n** – порядок производной. В случае **f '(x)** порядок определяется количеством апострофов.

Неопределенный интеграл вычисляется с помощью 2-х команд (рисунок *29*):

- прямого исполнения – **int(f, x)**, где **f** – подынтегральная функция, **x** – переменная интегрирования;
- отложенного исполнения – **Int(f, x)** – где параметры команды такие же, как и в команде прямого исполнения **int**. Команда **Int** выдает на экран интеграл в аналитическом виде математической формулы.

```
> Diff(2·x^3, x) = diff(2·x^3, x)
                    d/dx (2·x^3) = 6·x^2
> Diff(2·x^3, x) = (2·x^3)'
                    d/dx (2·x^3) = 6·x^2
> Diff(2·x^3, x$2) = diff(2·x^3, x$2)
                    d^2/dx^2 (2·x^3) = 12·x
> Int(2·x^3, x) = int(2·x^3, x)
                    ∫ 2·x^3 dx = 1/2·x^4
> Int(2·x^3, x = 1..2) = int(2·x^3, x = 1..2)
                    ∫_1^2 2·x^3 dx = 15/2
```

Рисунок 29 – Команды дифференцирования и интегрирования

Определенный интеграл вычисляется с помощью двух команд:

- прямого исполнения – **int(f, x=a..b)**, где **f** – подынтегральная функция, **x** – переменная интегрирования, **a** и **b** – начало и конец отрезка интегрирования;
- отложенного исполнения – **Int(f, x=a..b)** – где параметры команды такие же, как и в команде прямого исполнения **int**. Команда **Int** выдает на экран интеграл в аналитическом виде математической формулы.

Определение нулей и экстремумов функции

Определить место нахождения нуля и точки экстремума функции в Maple можно с помощью команды **fsolve(eq, x=a..b)**, где **eq** – уравнение, **x** – переменная, относительно которой уравнение надо разрешить, **a** и **b** – диапазон поиска. Фактически поиск характерных точек сводится к решению одного уравнения с одной неизвестной.

В качестве уравнения для определения нуля функции $f(x)$ используется уравнение вида $f(x) = 0$, для определения экстремума – $f'(x) = 0$. Диапазон поиска обязательно должен быть таков, чтобы в него попадал только один нуль функции (в случае поиска нуля функции), или только один экстремум (если определяется экстремум функции).

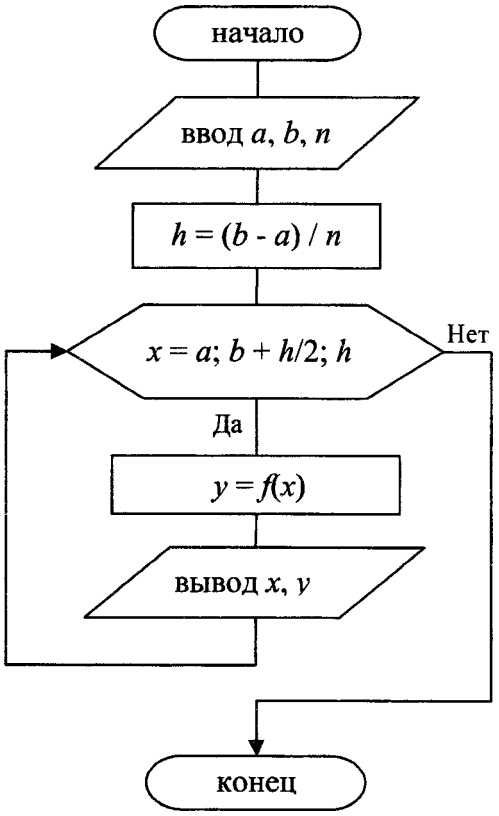
```
> f := x → sin(x) : f1 := x → d/dx f(x) :
> fsolve(f(x) = 0, x = 3·π/4 .. 5·π/4)
                    3.141592654
> fsolve(f1(x) = 0, x = π/4 .. 3·π/4)
                    1.570796327
```

Рисунок 30 – Определение нуля и экстремума функции из рисунка 22

Табулирование функции

Табулирование функции – это вычисление значений функции при изменении аргумента от некоторого начального значения до некоторого конечного значения с определенным шагом. Именно так составляются таблицы значений функций, отсюда и название – табулирование. Необходимость в табулировании возникает при решении достаточно широкого круга задач. Например, путем табулирования можно отделить (локализовать) корни уравнения, т.е. найти такие отрезки, на концах которых функция имеет разные знаки. Необходимость в табулировании возникает также при построении графиков функции на экране компьютера и т.д.

Блок-схема алгоритма табулирования:



Реализация табулирования $y(x) = \cos^2(x) - \sin(x^2 + 2x - 3)$ в MathCAD с помощью программного модуля

$$\text{Tab\_F}(a,b,n) := \left| \begin{array}{l} T_{0,0} \leftarrow \text{"x"} \quad T_{0,1} \leftarrow \text{"y(x)"} \\ h \leftarrow \dfrac{b-a}{n} \\ i \leftarrow 1 \\ \text{for } x \in a, a+h .. b + \dfrac{h}{2} \\ \quad \left| \begin{array}{l} T_{i,0} \leftarrow x \quad T_{i,1} \leftarrow y(x) \\ i \leftarrow i + 1 \end{array} \right. \\ T \end{array} \right.$$

Tab_F(0,3,10) =

	0	1
0	"x"	"y(x)"
1	0	1.14112
2	0.3	1.65167
3	0.6	1.67264
4	0.9	0.76659
5	1.2	-0.61334
6	1.5	-0.77307
7	1.8	0.69462
8	2.1	0.87835
9	2.4	-0.41335
10	2.7	1.07947
11	3	1.51666

Программная реализация в VBA:

```
Sub Tab_f()
 'функция y(x) определена в процедуре-функции Fun_y()
 a = 0
 b = 3
 n = 10

 h = (b - a) / n
 Range("A1") = "a = ": Range("B1") = a
 Range("A2") = "b = ": Range("B2") = b
 Range("A3") = "n = ": Range("B3") = n
 Range("A4") = "h = ": Range("B4") = h

 Range("A6") = "x": Range("B6") = "y"
 rw = 7
 For x = a To b + h / 2 Step h
        y = Fun_y(x)
        Cells(rw, 1) = x: Cells(rw, 2) = y
        rw = rw + 1
 Next x
End Sub
```

Результат выполнения в Excel:

	A	B
1	a =	0
2	b =	3
3	n =	10
4	h =	0,3
5		
6	x	y
7	0	1,14112
8	0,3	1,651673
9	0,6	1,672637
10	0,9	0,766587
11	1,2	-0,61334
12	1,5	-0,77307
13	1,8	0,69462
14	2,1	0,878349
15	2,4	-0,41335
16	2,7	1,07947
17	3	1,516658

Метод дихотомии (уточнение нуля функции)

Полагаем, что отделение корней произведено и на интервале $[a, b]$ расположен один корень функции $f(x)$, который необходимо уточнить с погрешностью ε. Одним из методов уточнения корня на отрезке является метод половинного деления (метод дихотомии). Суть метода заключается в последовательном делении выбранного отрезка на две равные части до тех пор, пока одновременно не выполнятся оба условия:

$$|f(c)| \le \varepsilon \text{ и } |b-a| \le \varepsilon,$$

где c – середина отрезка $[a, b]$, $c = (b - a) / 2$;

ε – точность вычисления (задается пользователем).

На каждом этапе деления та часть, на которой нет нуля функции, отбрасывается. Графически такую ситуацию можно представить так:

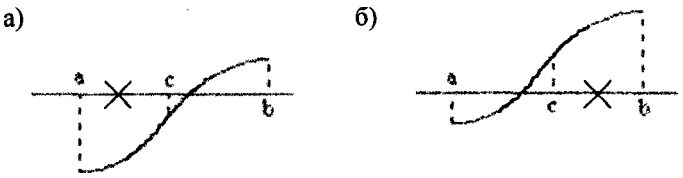
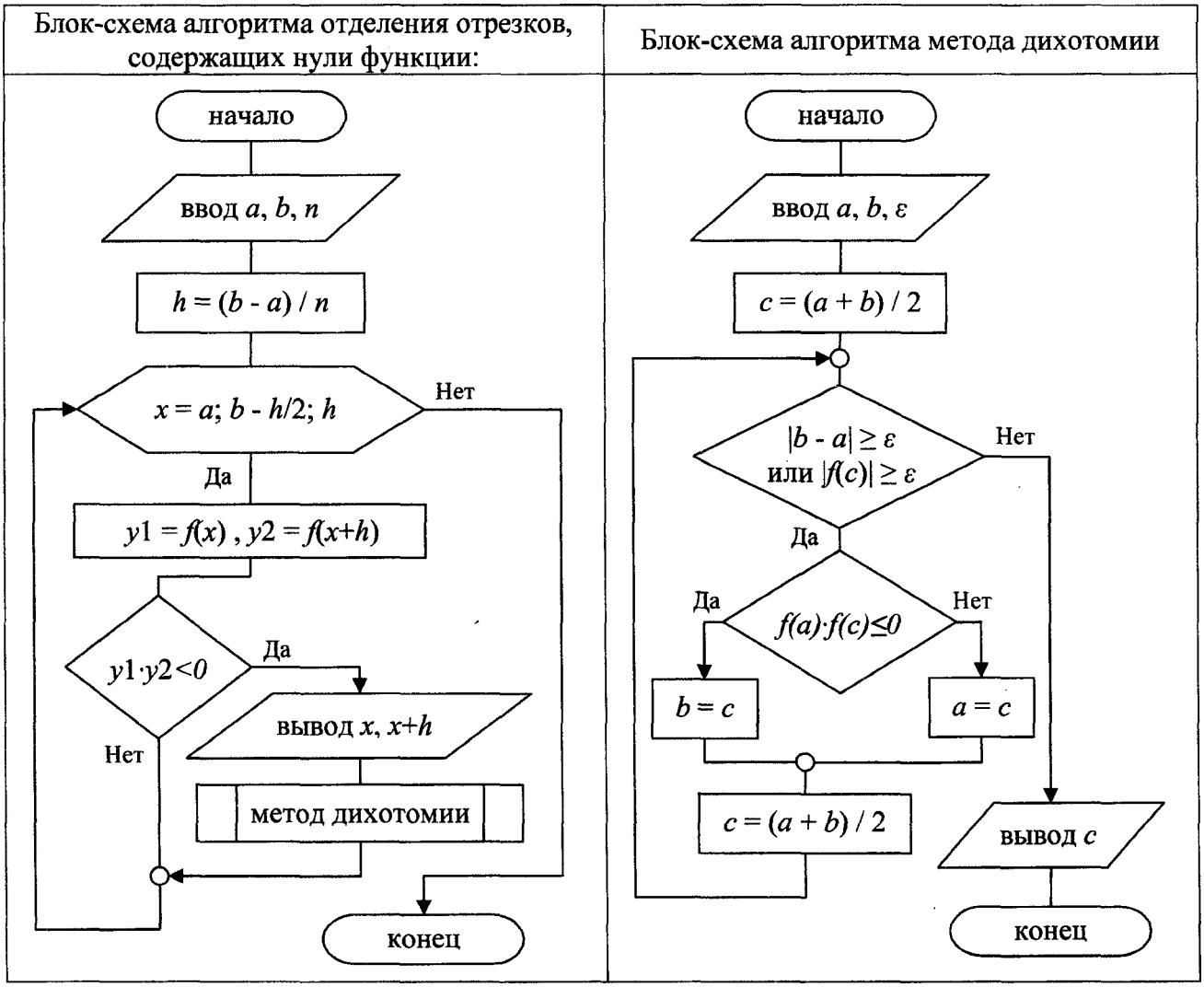


Рисунок 31 – Графическая интерпретация метода дихотомии

Аналитически, процедура «отбрасывания» происходит следующим образом:

$$\begin{cases} a = c, \text{ если } f(a)\cdot f(c) > 0 \text{ - рисунок 31(а)} \\ b = c, \text{ если } f(a)\cdot f(c) \le 0 \text{ - рисунок 31(б)} \end{cases}$$

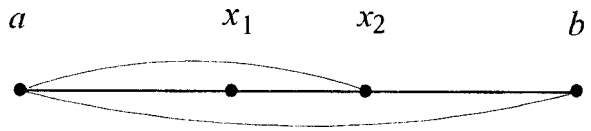


Метод золотого сечения

Метод золотого сечения – метод поиска экстремума функции одной переменной на локализованном отрезке. В основе метода лежит принцип деления отрезка в пропорциях **золотого сечения**: деление отрезка на две части так, чтобы отношение длины всего отрезка к длине большей части равнялось отношению длины большей части к длине меньшей части.

$$\frac{b-a}{b-x}=\frac{b-x}{x-a}=t \quad \text{или} \quad \frac{b-a}{x-a}=\frac{x-a}{b-x}=t$$

Для того чтобы найти определённое значение функции на заданном отрезке, отвечающее критерию поиска (пусть это будет минимум), рассматриваемый отрезок делится в пропорции золотого сечения в обоих направлениях, то есть выбираются две симметрично расположенные точки x_1 и x_2 такие, что:



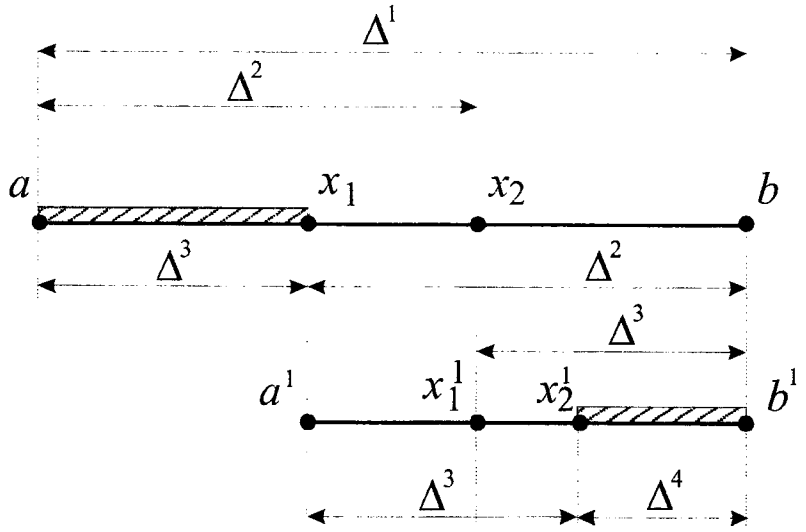
т.е. $\frac{b-a}{b-x_1}=\frac{b-x_1}{x_1-a}=t$ и $\frac{b-a}{x_2-a}=\frac{x_2-a}{b-x_2}=t$

Примечательно, что точка x_1 в свою очередь производит золотое сечение отрезка $[a, x_2]$, т.е.

$$\frac{x_2-a}{x_1-a}=\frac{x_1-a}{x_2-x_1}=t.$$

Аналогично, точка x_2 производит золотое сечение отрезка $[x_1, b]$.

Проиллюстрируем на рисунке (отбрасываемый интервал на рисунке будет заштрихован):



где $\Delta_1 = b-a$ – исходный интервал;

Δ_2 – интервал, полученный после уменьшения интервала Δ_1 отбрасыванием его левого или правого подынтервала.

Итак, метод золотого сечения состоит в том, что длины последовательных интервалов берутся в фиксированном отношении:

$$\frac{\Delta^1}{\Delta^2}=\frac{\Delta^2}{\Delta^3}=t.$$

Поскольку $\Delta^1=\Delta^2+\Delta^3$, то получаем

$$t=\frac{\Delta^1}{\Delta^2}=\frac{\Delta^2+\Delta^3}{\Delta^2}=1+\frac{\Delta^3}{\Delta^2}=1+\frac{1}{t}\Rightarrow t^2-t-1=0.$$

Корнем этого уравнения является золотое сечение:

$$t = \frac{\sqrt{5}+1}{2} \approx 1.618033988\,,\ \frac{1}{t} \approx 0.618$$

Можно записать формулы для точек x_1 и x_2, производящих золотое сечение на интервале $[a, b]$:

$$x_1 = b - \frac{b-a}{t} = b - (b-a)\cdot 0.618\ ;\ x_2 = a + \frac{b-a}{t} = a + (b-a)\cdot 0.618\ .$$

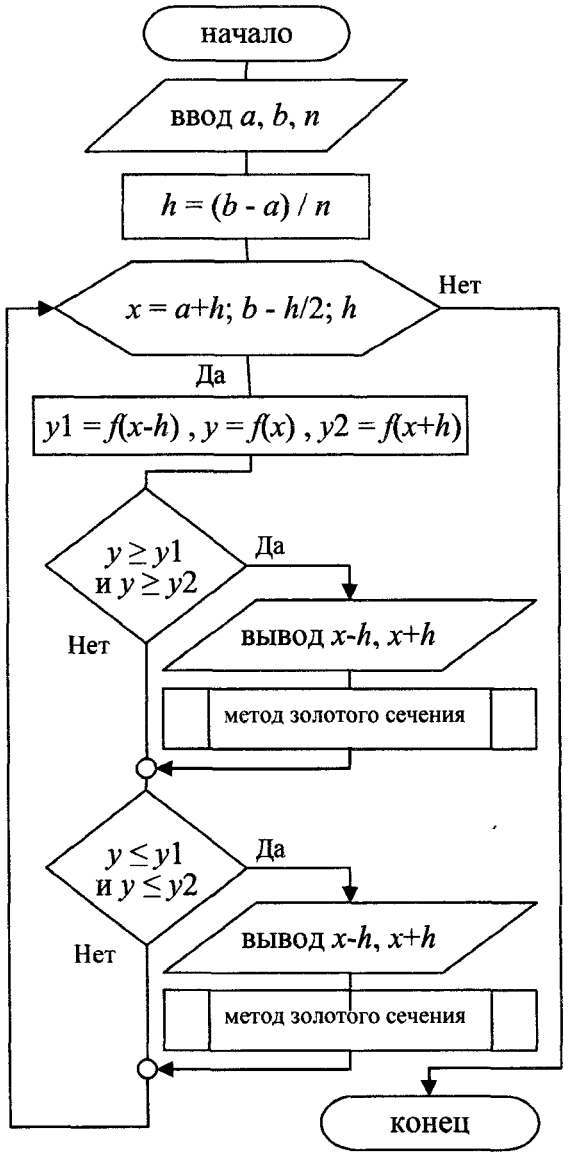
Алгоритм метода золотого сечения следующий.

На первой итерации заданный отрезок делится двумя симметричными относительно его центра точками x_1 и x_2; рассчитываются значения в этих точках. После чего тот из концов отрезка, к которому среди двух точек ближе оказалась та, значение в которой больше (для случая поиска минимума), отбрасывают.

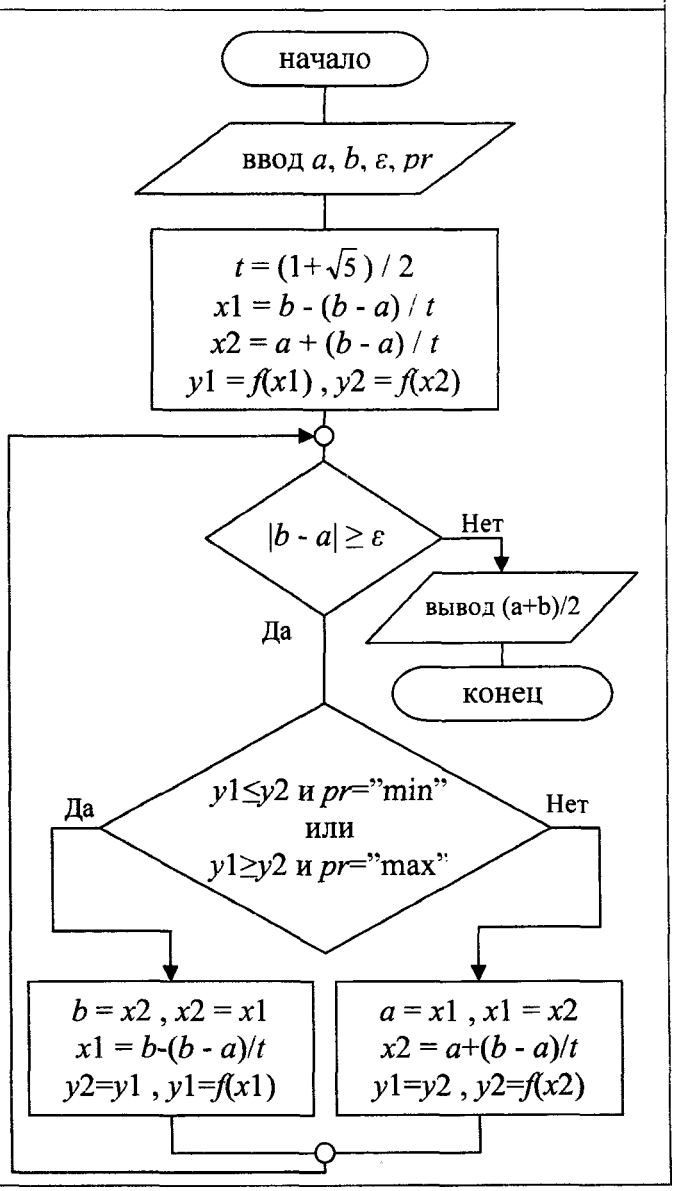
На следующей итерации в силу показанного выше свойства золотого сечения уже надо искать всего одну новую точку.

Процедура продолжается до тех пор, пока не будет достигнута заданная точность.

Блок-схема алгоритма отделения отрезков, содержащих экстремумы



Блок-схема алгоритма метода золотого сечения



Лабораторная работа № 1(1,2,3)

ТЕМА. Основы работы в VBA. Создание пользовательских функций и макросов (с линей - структурой) в Excel + VBA и в MathCAD.

Условие: Функция: $f(x)$ =

Выражение F1: Выражение F2:

Задача по геометрии: _____

_____

_____

Задание 1. Разработать линейную программу для вычисления значений функции $f(x)$.

Public Function Fun_f( )

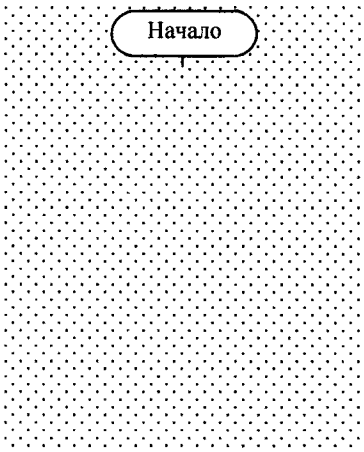
_ _

_ _

_ _

_ _

End Function



Результаты пошагового выполнения программы

	x =	x =	x =
переменная			
переменная			
переменная			
функция Fun_f			

Задание 2. Вычислить значения выражений F1 и F2.

	A	B	C	D	E
1	Параметры:			Выражение:	
2	x=			F1=	
3	y=			F2=	
4	z=				

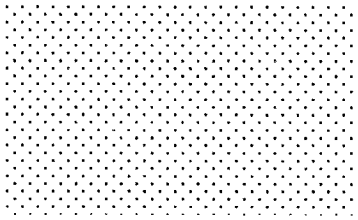
Ячейка E2: _____

_____

Ячейка E3: _____

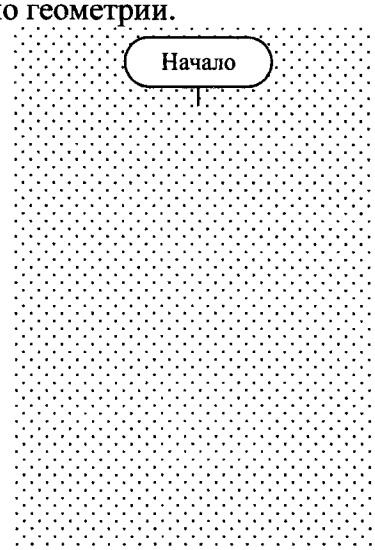
_____

Задание 3. Разработать линейную программу для решения задачи по геометрии.



Рисунок

№ п/п	Дано	Результат	
		Excel	MathCAD
1			
2			



Строка кода в VBA с функциями InputBox и MsgBox:

– Ввод первого параметра _____

_____

_____

– Вывод результата _____

_____

_____

Работа выполнена верно: _____ _____

(дата) (подпись)

Задания на защиту лабораторной работы № 1(1,2,3)

Задание 1. Разработать линейную программу для вычисления значений функции *f*(*x*):

Public Function Fun_f2(   )

\- -

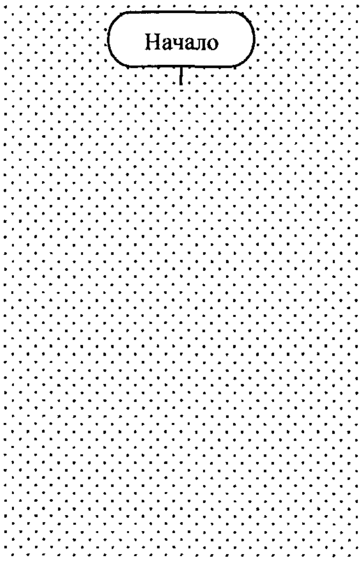
\- -

\- -

\- -

\- -

End Function



Результаты пошагового выполнения программы

	x =	*x* =
переменная		
переменная		
переменная		
переменная		
функция Fun_f2		

выдано: __________ ________________
(дата) (подпись)

Задание 2. Вычислить значение выражения F3 =

	A	B	C	D	E
1	Параметры:			Выражение:	
2	x=			F3=	
3	y=				
4	z=				

Public Function Fun_f3(   )

\- -

\- -

\- -

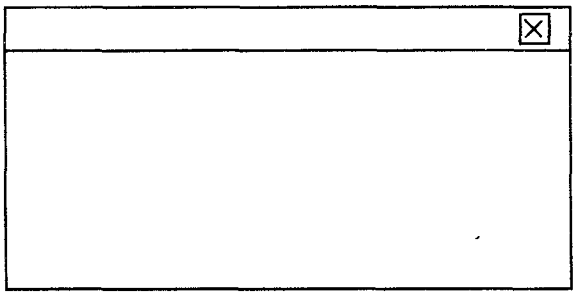
\- -

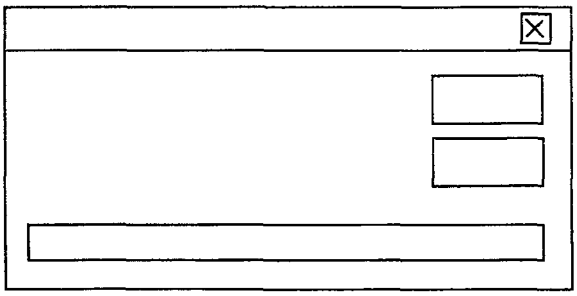
\- -

End Function

выдано: __________ ________________
(дата) (подпись)

Задание 3. Сформировать диалоговое окно на основе рисунка.





Строка кода в VBA:

– Рисунок слева____

____

– Рисунок справа____

____

выдано: __________ ________________
(дата) (подпись)

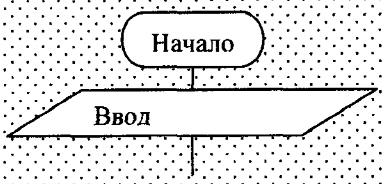
Лабораторная работа № 2

ТЕМА. Основы работы в VBA. Создание пользовательских функций (с разветвляющейся структурой) в Excel + VBA.

Условие:

функция q(a,b,x) =

Задание 1. Составить блок-схему алгоритма, реализующего вычисление заданной функции.



Задание 2. Разработать программу в VBA. (без использования команды Debug.Print).

```
Public Function Fun_q(                )
________________________________
________________________________
________________________________
________________________________
________________________________
________________________________
________________________________
________________________________
________________________________
________________________________
________________________________
________________________________
________________________________
________________________________
________________________________
End Function
```

Задание 3. Вычислить значение функции.

а)

a=	b=	x=

q = ________________(ветвь __)

б)

a=	b=	x=

q = ________________(ветвь __)

в)

a=	b=	x=

q = ________________(ветвь __)

г)

a=	b=	x=

q = ________________(ветвь __)

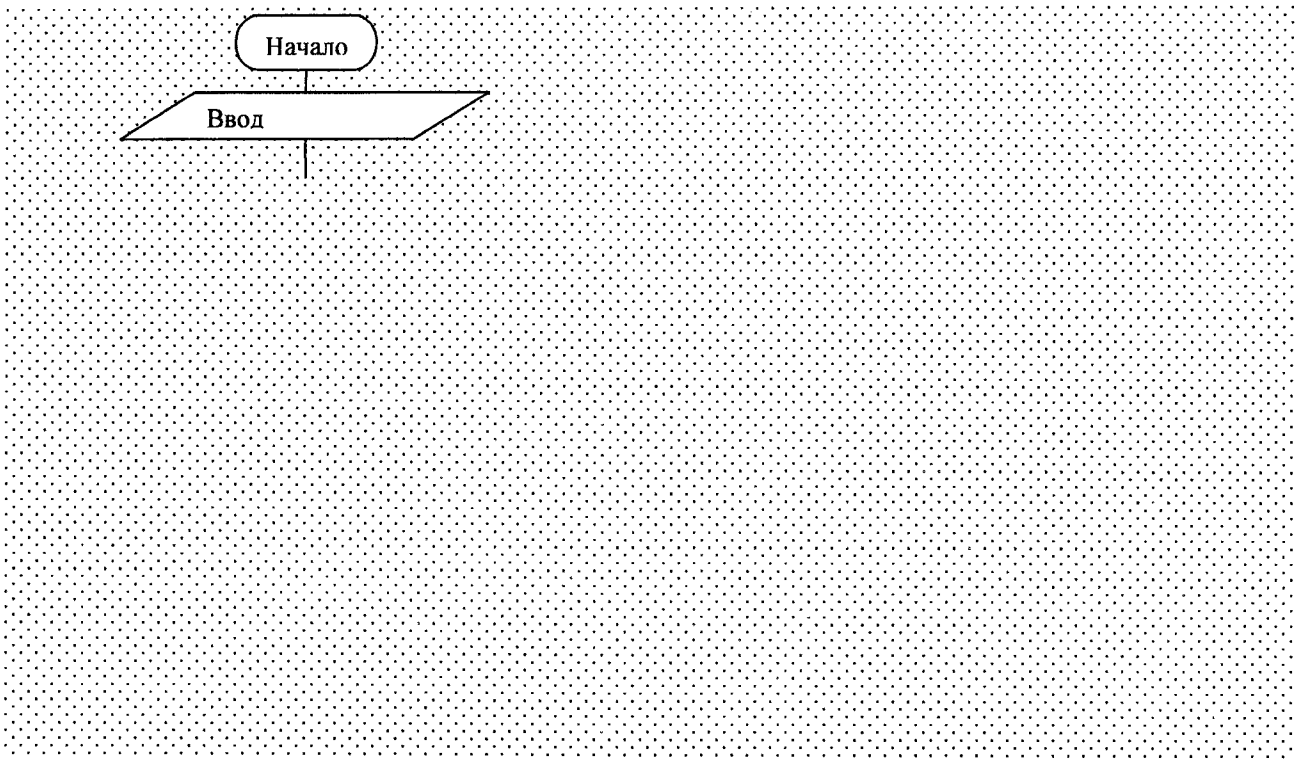
Работа выполнена верно: ____________ ________________

(дата) (подпись)

Задания на защиту лабораторной работы № 2

Задание 1. Составить блок-схему алгоритма, реализующего вычисление функции:

q( ) =



выдано: _____ _____
(дата) (подпись)

Задание 2. Разработать программу в VBA для функции:

q(a,b,x) =

а)	a =	b =	x =	q =	ветвь –
б)	a =	b =	x =	q =	ветвь –

выдано: _____ _____
(дата) (подпись)

Задание 3. Разработать программу в VBA для функции:

q(a,b,x) =

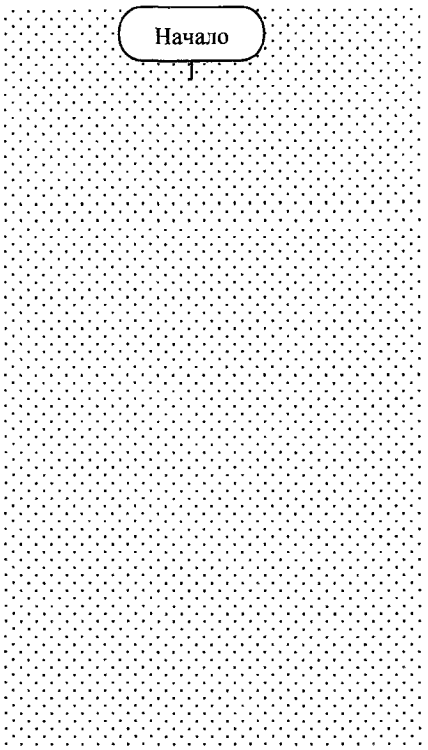
а)	a =	b =	x =	q =	ветвь –
б)	a =	b =	x =	q =	ветвь –

выдано: _____ _____
(дата) (подпись)

Лабораторная работа № 3(1,2)

ТЕМА. Основы работы в VBA. Создание пользовательских функций (с циклической структурой) в Excel + VBA и MathCAD.

Задание 1. Разработать циклическую программу для вычисления суммы S =



Public Function Summa_S(n)

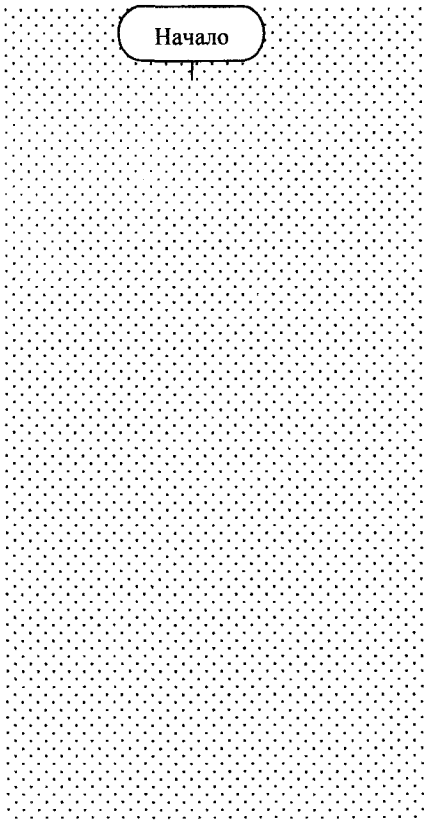
_____
_____
_____
_____
_____
_____
_____
_____
_____

End Function

Результаты выполнения программы

	A	B	C	D
1	n=	0	s=	
2	n=	1	s=	
3	n=	2	s=	
4	n=		s=	

Задание 2. Разработать программу для вычисления суммы S = с заданной точностью ε.



Public Function Summa(eps)

_____
_____
_____
_____
_____
_____
_____
_____
_____

End Function

Результаты выполнения программы

	A	B	C	D
	eps =		S =	
	eps =		S =	

Результат последнего шага при ε = _____

	A	B	C	D
3	n	элемент	условие	сумма

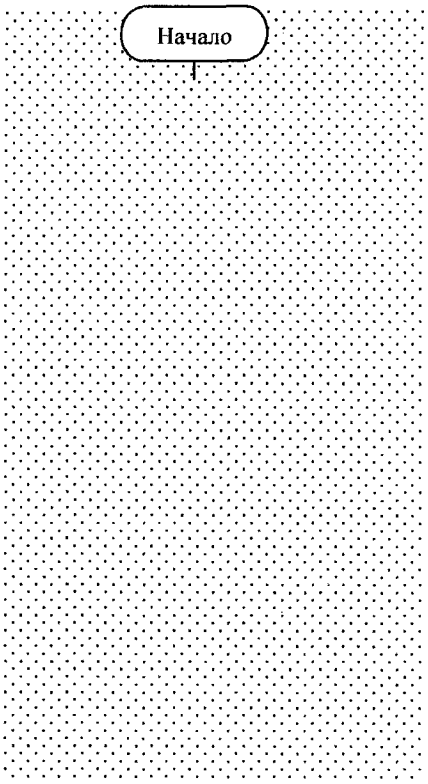
Формула в ячейке B__ = _____

Отрезок, содержащий значение суммы с заданной точностью, [_____;_____].

Работа выполнена верно: _____ _____
(дата) (подпись)

Задания на защиту лабораторной работы № 3(1,2)

Задание 1. Разработать циклическую программу для вычисления суммы:



с помощью оператора цикла: □ For...Next, □ While...Wend, □ Do { □ While / □ Until}...Loop { □ While / □ Until}

Public Function Summa_S(n)

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

End Function

Результаты выполнения программы

	A	B	C	D
1	n=		s=	

выдано: _____ _____
(дата) (подпись)

Задание 2. Определить, какое □ минимальное / □ максимальное количество чисел последовательности из задания 1 лабораторной работы необходимо взять, чтобы ее элемент ____ превысил значение _____. Ответ: ____ значений.

выдано: _____ _____
(дата) (подпись)

Задание 3. Определить, сколько чисел последовательности из задания 2 лабораторной работы нужно взять, в случае когда точность вычисления ее суммы □ увеличится / □ уменьшится в ___ раз(а). Ответ: ____ значений.

выдано: _____ _____
(дата) (подпись)

Задание 4. Определить отрезок, содержащий значение суммы последовательности из задания 2 лабораторной работы, вычисленной с точностью ε = _____.
Ответ: [_____;_____].

выдано: _____ _____
(дата) (подпись)

Задание 5. Определить отрезок, содержащий значение суммы последовательности из задания 2 лабораторной работы, вычисленной с точностью ε = _____.
Ответ: [_____;_____].

выдано: _____ _____
(дата) (подпись)

Лабораторная работа № 4

ТЕМА. Знакомство с СКМ Maple. Простейшие расчеты.

Условие:

арифметическое выражение

система уравнений

$$\begin{cases} \\ \\ \end{cases}$$

при: a = b = c =

Задание 1. Вычислить значение арифметического выражения.

Результат равен _____.

Задание 2. Решить систему уравнений, используя встроенные функции.

Переменная	Функция solve()	Функция fsolve()	Функция solve() (без ур. __ и ур. __)
x1			
x2			
x3			
x4			

Задание 3. Построить на одном графике две дополнительные функции на отрезке [-π, π].

```
> plot(_____
     _____
```

Задание 4. Построить в параметрической системе координат _____.

```
> plot(_____
     _____
```

Задание 5. Построить график функции, заданной в полярной системе координат.

```
> plot(_____
     _____
```

Работа выполнена верно: _____ _____
(дата) (подпись)

Задания на защиту лабораторной работы № 4

Задание 1. Вычислить значение арифметического выражения.

Результат:

_____

при: a = b = c =

выдано: _____ _____
(дата) (подпись)

Задание 2. Решить систему уравнений, используя встроенные функции.

система уравнений: $\begin{cases} \\ \\ \end{cases}$

x = _____

y = _____

z = _____

выдано: _____ _____
(дата) (подпись)

Задание 3. Решить систему уравнений, используя встроенные функции.

система уравнений: $\begin{cases} \\ \\ \end{cases}$

x = _____

y = _____

z = _____

выдано: _____ _____
(дата) (подпись)

Задание 4. Построить на графике (□ одну □ две) функции _____,
_____ с параметрами: □ название _____,
1-я ф. □ цвет ___, □ толщина ___, 2-я ф. □ цвет ___, □ толщина ____, легенда – □ да, □ нет.

```
> plot(_____
     _____
```

выдано: _____ _____
(дата) (подпись)

Задание 5. Построить на графике (□ одну □ две) функции _____,
_____ с параметрами: □ название _____,
1-я ф. □ цвет ___, □ толщина ___, 2-я ф. □ цвет ___, □ толщина ____, легенда – □ да, □ нет.

```
> plot(_____
     _____
```

выдано: _____ _____
(дата) (подпись)

Лабораторная работа № 5(1)

ТЕМА. Исследование функции одной переменной на отрезке в Maple.

Условие:

$y(x) =$_____ a =____ b =____

Задание 1. Определить функцию $y(x)$ и её первую производную. Построить обе функции на одном графике на отрезке $[a, b]$.

Maple:
отобразить легенду;
ориентировочный размер графика – 8×16 см

Задание 2. Используя встроенные возможности Maple, найти:

– нули функции;

α	β	x	$y(x)$

– локальные экстремумы функции.

α	β	x	$y(x)$	$y'(x)$	тип

Задание 3. Вычислить площадь криволинейной фигуры, ограниченной функцией $y(x)$.

Фигура показана на графике. Площадь фигуры: _____.

Задание 4. Найти координаты точки пересечения касательных k1(x) и k2(x), проведенных к первому и последнему нулю функции $y(x)$.

xt =_____; k1(xt) =_____; k2(xt) =_____.

Касательные k1(x) и k2(x) показаны на графике.

Работа выполнена верно: _____ _____
(дата) (подпись)

Задания на защиту лабораторной работы № 5(1)

Задание 1. Используя встроенные возможности Maple, построить график функции y(x) = _____ на участке [a =_____; b = _____] и определить все □ нули, □ локальные минимумы и □ локальные максимумы функции.

– нули / – локальные экстремумы функции

α	β	x	$y(x)$	$y'(x)$	тип

выдано: _____ _____
(дата) (подпись)

Задание 2. Вычислить площадь криволинейной фигуры, ограниченной функцией $y(x)$ = _____ на участке [a =_____; b =_____], выбрав в качестве левой границы отрезка __ □ нуль (□ экстремум) функции, а в качестве правой границы отрезка __ □ нуль (□ экстремум) функции.

Площадь фигуры: _____.

выдано: _____ _____
(дата) (подпись)

Задание 3. Вычислить площадь треугольника, ограниченного осью OX и касательными, проведенными к функции $y(x)$ из лабораторной работы в точках с координатами x1 = _____ и x2 = _____. Площадь фигуры: _____ .

выдано: _____ _____
(дата) (подпись)

Задание 4. Вычислить площадь треугольника, ограниченного осью OX и касательными, проведенными к функции $y(x)$ из лабораторной работы в точках с координатами x1 = _____ и x2 = _____. Площадь фигуры: _____ .

выдано: _____ _____
(дата) (подпись)

Задание 5. Найти координаты точки пересечения касательных k3(x), проведенной к ____ □ нулю (□ экстремуму), и k4(x), проведенной к ____ □ нулю (□ экстремуму) функции $y(x)$ = _____ на участке [a = _____; b =_____].

xt = _____; k1(xt) = _____; k2(xt) = _____.

выдано: _____ _____
(дата) (подпись)

Задание 6. Найти координаты точки пересечения касательных k3(x), проведенной к ____ □ нулю (□ экстремуму), и k4(x), проведенной к ____ □ нулю (□ экстремуму) функции $y(x)$ = _____ на участке [a = _____; b =_____].

xt = _____; k1(xt) = _____; k2(xt) = _____.

выдано: _____ _____
(дата) (подпись)

Лабораторная работа № 5(2,3)

ТЕМА Исследование функции одной переменной на отрезке в Excel + VBA и MathCAD с использованием подпрограмм (блоков).

Условие:

$y(x) =$_____ a =____ b =____ n =____

Задание 1. В VBA и MathCAD разработать процедуру (программный модуль) для построения таблицы значений на участке [a,b] функции $y(x)$:

```
Public Function Fun_y(x)
_____________________________________________
End Function
```

Задание 2. В VBA и MathCAD разработать процедуру (программный модуль) для отделения и уточнения нулей функции $y(x)$ на участке [a,b].

Результаты первых трех шагов метода дихотомии для первого нуля функции

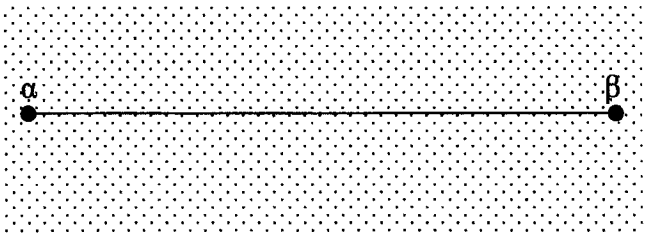
шаг 1	$\alpha =$	$c =$	$\beta =$
	$y(\alpha) =$	$y(c) =$	$y(\beta) =$
шаг 2	$\alpha =$	$c =$	$\beta =$
	$y(\alpha) =$	$y(c) =$	$y(\beta) =$
шаг 3	$\alpha =$	$c =$	$\beta =$
	$y(\alpha) =$	$y(c) =$	$y(\beta) =$

Задание 3. В VBA и MathCAD разработать процедуру (программный модуль) для отделения и уточнения локальных экстремумов функции $y(x)$ на участке [a,b].

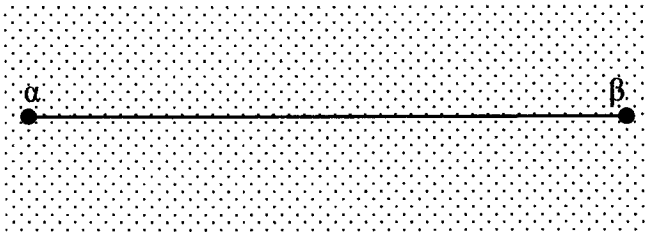
Результаты первых трех шагов метода золотого сечения для первого локального экстремума

шаг 1	$\alpha =$	$x1 =$	$x2 =$	$\beta =$
		$y(x1) =$	$y(x2) =$	
шаг 2	$\alpha =$	$x1 =$	$x2 =$	$\beta =$
		$y(x1) =$	$y(x2) =$	
шаг 3	$\alpha =$	$x1 =$	$x2 =$	$\beta =$
		$y(x1) =$	$y(x2) =$	

Задание 4. Проиллюстрировать результаты, полученные в задании 2 и 3.



метод дихотомии



метод золотого сечения

Работа выполнена верно: _____ _____

(дата) (подпись)

Задания на защиту лабораторной работы № 5(2,3)

Задание 1. Получить результаты первых трех шагов выполнения программы, реализующей уточнение ___ нуля или ___ локального экстремума функции при n =_____.

шаг 1	α =			β =
шаг 2	α =			β =
шаг 3	α =			β =

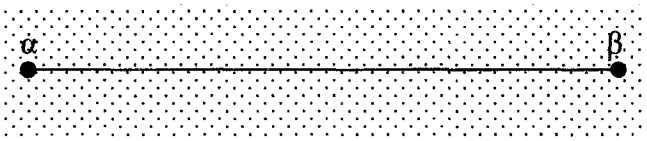
выдано: _____ _____
(дата) (подпись)

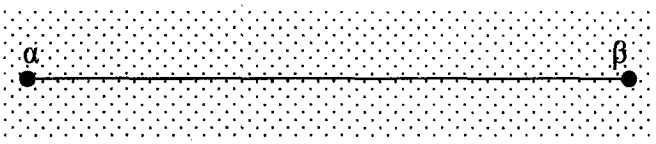
Задание 2. Получить результаты первых трех шагов выполнения программы, реализующей уточнение ___ нуля или ___ локального экстремума функции при n =_____.

шаг 1	α =			β =
шаг 2	α =			β =
шаг 3	α =			β =

выдано: _____ _____
(дата) (подпись)

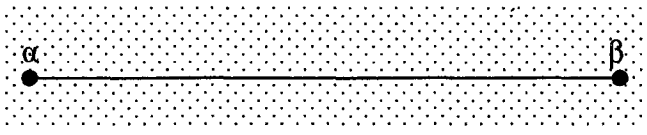
Задание 3. Проиллюстрировать результаты первых трех шагов выполнения программы, реализующей уточнение ___ нуля или ___ локального экстремума функции при n =_____, либо результат четвертого шага для уточнения □ нуля или □ локального экстремума функции из лабораторной работы.

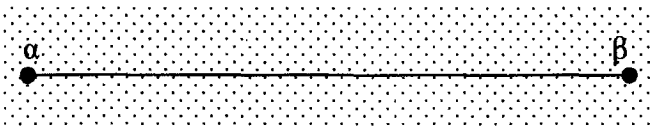




выдано: _____ _____
(дата) (подпись)

выдано: _____ _____
(дата) (подпись)





выдано: _____ _____
(дата) (подпись)

выдано: _____ _____
(дата) (подпись)

Лабораторная работа № 6(1,2,3)

ТЕМА. Исследование функции двух переменных на прямоугольнике.

Условие:

$Z(x,y) =$

ax =____ bx =____ nx =____

ay =____ by =____ ny =____

Задание 1. В *Excel+VBA* и MathCAD разработать процедуры для построения таблицы значений функции $z = Z(x,y)$.

```
Public Function Fun_Z(x, y)
Fun_Z = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
End Function
```

Z(ax,ay) = ____________; *Z*(bx,by) = ____________.

Задание 2. Построить график функции *Z*.

Maple: [> ____

Задание 3. Найти глобальные экстремумы для заданной области (MathCAD).

Нач. прибл.	x	y	Z
мин			
макс			

Уточненное	x	y	Z
мин			
макс			

Задание 4. Определить области, содержащие локальные экстремумы (Excel).

мин – {____$\le x \le$____, ____$\le y \le$____}, макс – {____$\le x \le$____, ____$\le y \le$____}

Ут. значения: x_{min} = ____________; y_{min} = ____________; z_{min} = ____________;

x_{max} = ____________; y_{max} = ____________; z_{max} = ____________ .

Задание 5. Построить график линий уровня и показать все лок. экстремумы (Maple + MathCAD).

График линий уровня (Maple).
Размер: 9 X 9 см.
Отметить: точки локальных минимумов символом – ●;
точки локальных максимумов символом – ■.

	x	y	Z(x,y)
Минимум			
Максимум			

Работа выполнена верно: ____________ ______________

(дата) (подпись)

Задания на защиту лабораторной работы № 6(1,2,3)

Приклеить распечатку с графиками (Maple): график сечения Z=0, графики сечений при фиксированном значении x и фиксированном значении y

Задание 1. Уточнить значение локального экстремума (указано на графике).

Система	Минимум			Максимум		
	x	y	Z(x,y)	x	y	Z(x,y)
Excel						
MathCAD						

выдано: _____ _____
(дата) (подпись)

Задание 2. Уточнить значение локального экстремума (указано на графике).

Система	Минимум			Максимум		
	x	y	Z(x,y)	x	y	Z(x,y)
Excel						
MathCAD						

выдано: _____ _____
(дата) (подпись)

Задание 3. С помощью инструмента «условное форматирование» отметить цветом _____ и _____ места локальных экстремумов функции на участке [ax, bx] = = [____,____] и [ay, by] = [____,____] при количестве разбиений по оси x nx = ____ и по оси y ny = ____. Выписать по одному результату на каждый вид экстремума.
Ответ: область _____.

Минимум			

Максимум			

выдано: _____ _____
(дата) (подпись)

Задание 4. С помощью инструмента «условное форматирование» отметить цветом _____ и _____ места локальных экстремумов функции на участке [ax, bx] = = [____,____] и [ay, by] = [____,____] при количестве разбиений по оси x nx=____ и по оси y ny = ____. Выписать по одному результату на каждый вид экстремума.
Ответ: область _____.

Минимум			

Максимум			

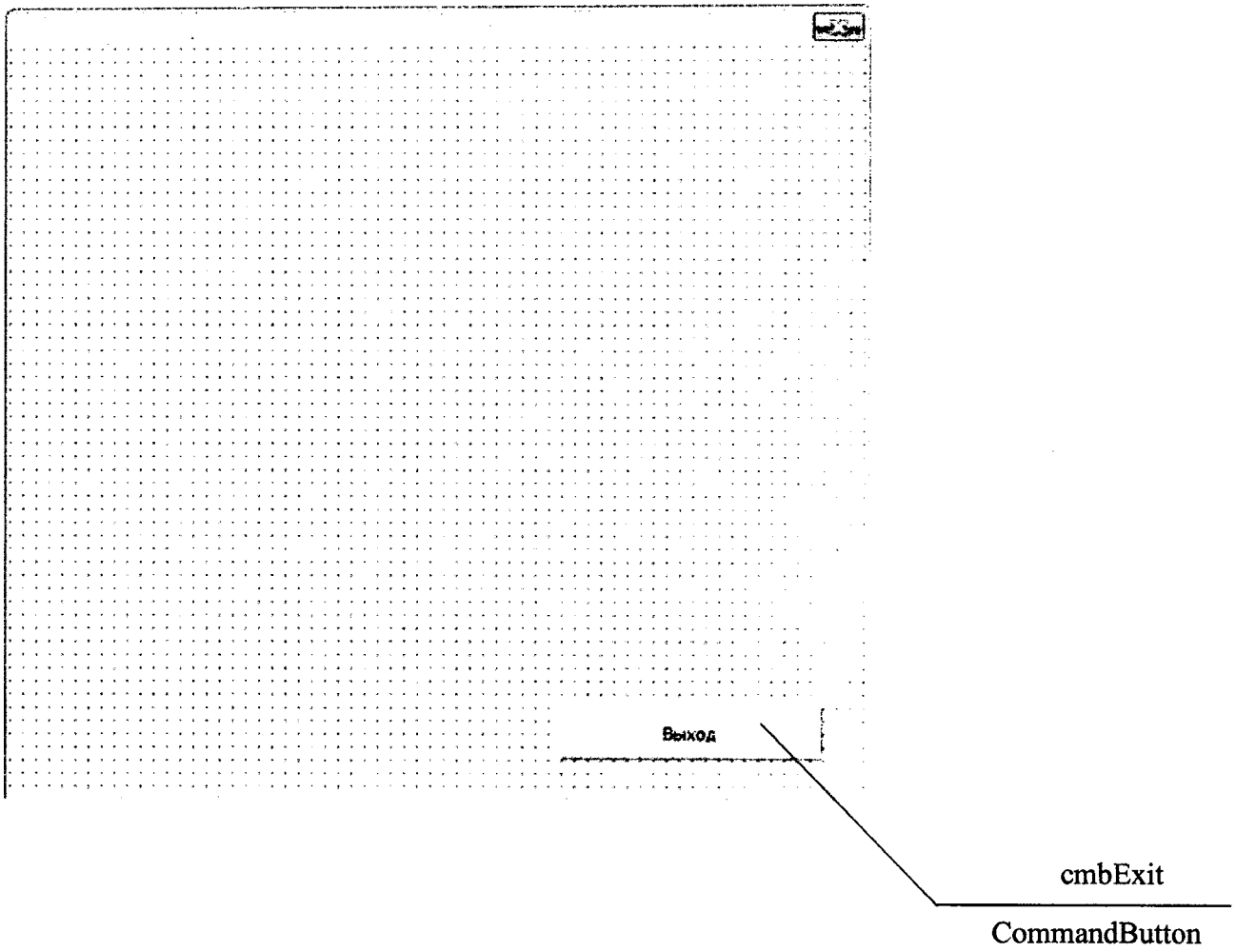
выдано: _____ _____
(дата) (подпись)

Лабораторная работа № 7

ТЕМА. Разработка пользовательских форм.

Условие указано в лабораторной работе №1.

Задание 1. Разработать в VBA пользовательскую форму для решения геометрической задачи.



Задание 2. Организовать вывод результата задачи на рабочий лист Excel.

Мера угла	◉ градусная ○ радианная	○ градусная ◉ радианная
Исходные данные		
Результат (Excel)	_____ = _____ ячейка значение	_____ = _____ ячейка значение

Задание 3. Реализовать в работе дополнительное задание (не обязательное).

_____

_____

выдано: _____ _____
(дата) (подпись)

Задание 4. Выписать значения измененных свойств созданных объектов (кроме размеров объектов).

Объект/Имя	Свойство (кроме имени)	Значение

Работа выполнена верно: ____________ ________________

(дата) (подпись)

Список дополнительной литературы

1. Бондаренко, М. Microsoft Word 2003 в теории и на практике / М. Бондаренко, С. Бондаренко. – М.: Новое знание, 2004. – 336 с.
2. Макаров, Е.Г. Mathcad: учебный курс (+CD) / Е.Г. Макаров. – СПб.: Питер, 2009. – 384 с.
3. Савотченко, С.Е. Методы решения математических задач в Maple: учебное пособие / С.Е. Савотченко, Т.Г. Кузьмичева. – Белгород: Изд. Белаудит, 2001. – 116 с.
4. Слепцова, Л.Д. Программирование на VBA в Microsoft Office 2010 / Л.Д. Слепцова. – М.: ООО «И.Д. Вильямс», 2010. – 432 с.

Оглавление

УЧЕБНОЕ ИЗДАНИЕ

Составители:
Кофанов Валерий Анатольевич
Хомицкая Татьяна Георгиевна
Тузик Ирина Владимировна

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по дисциплине «Информатика»
для студентов строительных специальностей
дневной формы обучения
второй семестр

Ответственный за выпуск:Кофанов В.А.
Редактор:Боровикова Е.А.
Компьютерная верстка:Кармаш Е.Л.
Корректор:Никитчик Е.В.

Подписано к печати 31.12.2015 г. Бумага «Снегурочка». Формат 60x84 $^1/_8$.
Гарнитура Times New Roman. Усл. печ. л. 5,58. Уч. изд. л. 6,0.
Заказ № 1347. Тираж 205 экз. Отпечатано на ризографе учреждения образования
«Брестский государственный технический университет»
224017, г. Брест, ул. Московская, 267.