**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ**

**ГОСУДАРСТВЕННОЕ ВЫСШЕЕ УЧЕБНОЕ ЗАВЕДЕНИЕ**

**УКРАИНСКИЙ ГОСУДАРСТВЕННЫЙ**

**ХИМИКО-ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ**

**КАФЕДРА ИТК**

**КУРСОВАЯ РАБОТА**

**ПО КУРСУ**

**«ОБЪЕКТНО-ОРИЕНТИРОВАНОЕ ПРОГРАММИРОВАНИЕ»**

**ВЫПОЛНИЛ:**

студент группы 3ИC27 Куделя С.В.

**ПРОВЕРИЛ:**

ассистент Ильхман Яна Викторовна

Днепропетровск 2009

**СОДЕРЖАНИЕ**

1. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

2. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ (ПО)

2.1 Назначение программного продукта

2.2 Основные задачи

2.3 Входные и выходные данные

3. ПРОЕКТИРОВАНИЕ

3.1 Выделение основных объектов ПО

3.2 Описание полей и методов

3.3 Иерархия классов на основе выделенных объектов

4. ОСНОВНЫЕ ФОРМЫ И КОМПОНЕНТЫ, ИСПОЛЬЗУЕМЫЕ ДЛЯ РЕАЛИЗАЦИИ ПРОГРАММЫ. ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ РЕШЕНИЯ РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

4.1 Метод Рунге-Кутта

4.2 Описание программы ” РЕШЕНИЕ ОДУ “

4.3 Назначение элементов графического окна программы

4.4 Реакция программы при возникновении ошибок

4.5 Перечень компонент DELPHI использованных в программе

5. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ И ТРЕБОВАНИЯ К ПО

6. ТЕКСТ ПРОГРАММЫ

7. РЕШЕНИЕ ДИФФЕРЕНЦИАЛЬНОГО УРАВНЕНИЯ Y = Y−2X/Y МЕТОДОМ РУНГЕ – КУТТА В СРЕДЕ EXCEL

ВЫВОД

**1. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ**

Delphi является объектно-ориентированной средой программирования. В качестве языка программирования используется язык Object Pascal.

Исторически сложилось так, что программирование возникло и развивалось как процедурное программирование, которое предполагает, что основой программы является алгоритм, процедура обработки данных.

Объектно-ориентированное программирование (ООП) — это методика разработки программ, в основе которой лежит понятие объект. Объект — это некоторая структура, соответствующая объекту реального мира, его поведению. Задача, решаемая с использованием методики ООП, описывается в терминах объектов и операций над ними, а программа при таком подходе представляет собой набор объектов и связей между ними. Объектно-ориентированное программирование позволяет конструировать новые и производные (дочерние) классы на основе существующих классов.

По сравнению с традиционными способами программирования ООП обладает рядом преимуществ. Главное из них заключается в том, что эта концепция в наибольшей степени соответствует внутренней логике функционирования операционной системы (ОС) Windows. Программа, состоящая из отдельных объектов, отлично приспособлена к реагированию на события, происходящие в ОС. К другим преимуществам ООП можно отнести большую надежность кода и возможность повторного использования отработанных объектов.

Основные понятия ООП в языке Delphi: объект, класс, компонент;

Основные механизмы ООП: инкапсуляция, наследование и полиморфизм;

**Класс** — помимо описания данных, включает описание процедур и функций, которые могут быть выполнены над представителем класса — объектом. Данные класса называются полями, процедуры и функции — методами. Переменные в зависимости от предназначения именуются полями или свойствами.

При создании объекта он наследует структуру (переменные) и поведение (методы) своего класса.

Класс, называемый потомком, производным или дочерним классом (подклассом), также может быть создан на основе другого родительского класса (предка) и при этом наследует его структуру и поведение.

Методы — это процедуры и функции, описанные внутри класса и предназначенные для операций над его полями. В состав класса входит указатель на специальную таблицу, где содержится вся информация, нужная для вызова методов. От обычных процедур и функций методы отличаются тем, что им при вызове передается указатель на тот объект, который их вызвал. Поэтому обрабатываться будут поля именно того объекта, который вызвал метод. Внутри метода указатель на вызвавший его объект доступен под зарезервированным именем self.

Понятие свойства можно определить его как поле, доступное для чтения и записи не напрямую, а через соответствующие методы.

Классы могут быть описаны либо в секции интерфейса модуля, либо на верхнем уровне вложенности секции реализации. Не допускается описание классов внутри процедур и других блоков кода.

Любой **компонент** (элемент управления) или объект в Delphi всегда является экземпляром класса.

Программно объект представляет собой переменную **объектного** типа.

Для каждого компонента Delphi существует свой класс, наследуемый от TComponent. Предком всех объектов, включая компоненты, является класс TObject.

**Инкапсуляция** - это создание защищенных объектов, доступ к свойствам и методам которых разрешен только через определенные разработчиком <точки входа>. Иначе говоря, инкапсуляция - это предоставление разработчику конкретного набора свойств и методов для управления поведением и свойствами объекта, определяемыми внутри класса.

Инкапсуляция обеспечивает скрытие полей объекта с целью обеспечения доступа к ним только посредством методов класса.

В языке Delphi ограничение доступа к полям объекта реализуется при помощи свойств объекта. Свойство объекта характеризуется полем, сохраняющим значение свойства, и двумя методами, обеспечивающими доступ к полю свойства. Метод установки значения свойства называется методом записи свойства (write), а метод получения значения свойства — методом чтения свойства (read).

В описании класса перед именем свойства записывают слово property (свойство). Ключевое слово property обозначает свойство объекта, которое в отличие от обычных полей (переменных) класса имеет спецификаторы доступа обеспечивающие контроль доступа к свойствам объекта.

После имени свойства указывается его тип, затем — имена методов, обеспечивающих доступ к значению свойства. После слова read указывается имя метода, обеспечивающего чтение свойства, после слова write — имя метода, отвечающего за запись свойства.

Пример описания класса TPerson, содержащего два свойства: Name и Address:

**type**

TName = **string**[15]; TAddress = string[35];

TPerson = class // класс

**private**

FName: TName; // значение свойства Name

FAddress: TAddress; // значение свойства Address

**Constructor** Create(Name:Tname);

**Procedure** Show;

**Function** GetName: TName;

**Function** GetAddress: TAddress;

**Procedure** SetAddress(NewAddress:TAddress);

**public**

**Property** Name: Tname // свойство Name

**read** GetName; // доступно только для чтения

**Property** Address: TAddress // свойство Address

**read** GetAddress // доступно для чтения

**write** SetAddress; // и записи

**end;**

В программе для установки значения свойства записать обычную инструкцию присваивания значения свойству. Например, чтобы присвоить значение свойству Address объекта student, достаточно записать

student.Address := 'Гвардейский, ул.Зенитная 1, кв.10';

Внешне применение свойств ничем не отличается от использования полей объекта. Однако между свойством и полем объекта существует принципиальное отличие: при присвоении и чтении значения свойства автоматически вызывается процедура, которая выполняет некоторую работу.

**Наследование** позволяет определять новые классы в терминах существующих классов.

Новые классы возможно определять посредством добавления полей, свойств и методов к уже существующим классам. Такой механизм получения новых классов называется порождением. При этом новый, порожденный класс (потомок) наследует свойства и методы своего базового, родительского класса.

В объявлении класса-потомка указывается класс родителя. Например, класс TEmployee (сотрудник) может быть порожден от рассмотренного выше класса TPerson путем добавления поля FDepartment (отдел). Объявление класса TEmplioyee в этом случае может выглядеть так:

TEmployee = **class**(TPerson)

FDepartment: integer; // номер отдела

**constructor** Create(Name:TName; Dep:integer);

**end;**

Заключенное в скобки имя класса TPerson показывает, что класс TEmployee является производным от класса TPerson. В свою очередь, класс TPerson является базовым для класса TEmployee.

Класс TEmpioyee должен иметь свой собственный конструктор, обеспечивающий инициализацию класса-родителя и своих полей.

Пример реализации конструктора класса TEmployee:

**constructor** TEmpioyee.Create(Name:Tname;Dep:integer);

**begin**

**inherited** Create(Name);

FDepartment:=Dep;

**end;**

В приведенном примере директивой inherited вызывается конструктор родительского класса. После этого присваивается значение полю класса-потомка.

После создания объекта производного класса в программе можно использовать поля и методы родительского класса. Ниже приведен фрагмент программы, демонстрирующий эту возможность.

engineer := TEmployee.Create('Сидоров',413);

engineer.address := 'ул.Блохина, д.8, кв.10';

Первая инструкция создает объект типа TEmployee, вторая — устанавливает значение свойства, которое относится к родительскому классу.

**Полиморфизм** - это возможность различных объектов реагировать по-разному на одни и те же события, иначе говоря, возможность использовать одинаковые имена для методов, входящих в различные классы. Концепция полиморфизма обеспечивает в случае применения метода к объекту использование именно того метода, который соответствует классу объекта. Синтаксис языка поддерживает общепринятую для объектно-ориентированного программирования нотацию: **имя\_объекта: свойство** для ссылки на свойство объекта или имя\_объекта: метод для вызова метода объекта. При этом имя\_объекта может содержать как простую, так и квалифицированную ссылку. Квалифицированная ссылка содержит отделенные точкой имена родительских объектов.

Пусть определены три класса, один из которых является базовым для двух других:

**tуре**

// базовый класс TPerson = **class**

fname: **string;** // имя

**constructor** Create(name**:string);**

**function info: string;**

**virtual;**

**end;**

// производный от TPerson TStud = class(TPerson)

fgr:integer; // номер учебной труппы

**constructor** Create(name:**string;**gr:integer);

**function** info: **string; override; end;**

// производный от TPerson TProf = class(TPerson)

**fdep:string;** // название кафедры

**constructor** Create(name**:string;dep:string);**

**function info: string;**

**override;**

**end;**

В каждом из этих классов определен метод info. В базовом классе при помощи директивы virtual метод info объявлен виртуальным. Объявление метода виртуальным дает возможность дочернему классу произвести замену виртуального метода своим собственным. В каждом дочернем классе определен свой метод info, который замещает соответствующий метод родительского класса (метод порожденного класса, замещающий виртуальный метод родительского класса, помечается директивой override).

Определение метода info для каждого класса:

**function** TPerson.info:**string;**

**begin**

result := '';

**end;**

**function** TStud.info:**string;**

**begin**

result := fname + ' гp.' + IntTostr(fgr);

**end;**

**function** TProf.info:string;

**begin**

result := fname + ' каф.' + fdep;

**end;**

Так как оба класса порождены от одного и того же базового, объявить список студентов и преподавателей можно так (следует помнить, что объект — это указатель):

list: **array**[l**.**.SZL] of TPerson;

Объявить подобным образом список можно потому, что язык Delphi позволяет указателю на родительский класс присвоить значение указателя на дочерний класс. Поэтому элементами массива list могут быть как объекты класса TStud, так и объекты класса TProf.

Вывести список студентов и преподавателей можно применением метода info к элементам массива. Например, так:

st := '';

**for** i:=l to SZL do // SZL - размер массива-списка

**if** list[i] о NIL

**then** st := st + list[i].Info

+ #13; ShowMessage (st);

Во время работы программы каждый элемент массива может содержать как объект типа xstud, так и объект типа TProf. Концепция полиморфизма обеспечивает применение к объекту именно того метода, который соответствует типу объекта.

Перегрузка методов

Есть еще одна, совершенно особенная разновидность методов — перегружаемые.

Перегрузка методов нужна, чтобы произвести одинаковые или похожие действия с разнотипными данными.

Пример, иллюстрирующий статические методы:

**type**

TlstObj = **class**

FExtData : Extended;

**procedure** SetData(AValue: Extended);

**end;**

T2ndObj = class(TlstObj)

FIntData : Integer;

**procedure** SetData(AValue: Integer); **end;**

**var** T1: TlstObj;

T2 : T2ndObj;

В этом случае попытка вызова из объекта Т2 методов

Т2.SetData (1.0);

Т2.SetData(1);

вызовет ошибку компиляции на первой из двух строк. Для компилятора внутри Т2 статический метод с параметром типа extended перекрыт, и он его "не признает".

Для выхода из сложившегося положения можно переименовать один из методов, например, создать SetlntegerData и SetExtendedData, но если методов не два, а, например, сто, моментально возникнет путаница. Сделать методы виртуальными нельзя, поскольку тип и количество параметров в одноименных виртуальных методах должны в точности совпадать. Для разрешения этой ситуации существуют перегружаемые методы, объявляемые при помощи директивы overload:

**type**

TlstObj = **class**

FExtData : Extended;

procedure SetData(AValue: Extended);**overload;**

**end;**

T2ndObj = class(TlstObj)

FIntData : Integer;

**procedure** SetData(AValue: Integer); **overload;**

**end;**

Объявив метод SetData перегружаемым, в программе можно использовать обе его реализации одновременно. Это возможно потому, что компилятор определяет тип передаваемого параметра (целый или с плавающей точкой) и в зависимости от этого подставит вызов соответствующего метода: для целочисленных данных — метод объекта T2ndobj, для данных с плавающей точкой — метод объекта Tistobj.

Можно перегрузить и виртуальный (динамический) метод. Надо только в этом случае добавить директиву reintroduce:

**type**

TlstObj = **class**

FExtData : Extended;

**procedure** SetData(AValue: Extended); **overload; virtual;**

**end**;

T2ndObj = class(TlstObj)

FIntData : Integer;

**procedure** SetData(AValue: Integer); **reintroduce; overload**;

**end;**

На перегрузку методов накладывается ограничение — нельзя перегружать методы, находящиеся в области видимости published, т. е. те, которые будут использоваться в Инспекторе объектов.

**Области видимости**

В модели объектов языка Delphi существует механизм доступа к составным частям объекта, определяющий области, где ими можно пользоваться (области видимости). Поля и методы могут относиться к четырем группам (секциям), отличающимся областями видимости. Методы и свойства могут быть общими (секция **public**), личными (секция **private**), защищенными (секция **protected**) и опубликованными (секция **published**). Есть еще и пятая группа, **automated**, она ранее использовалась для создания объектов СОМ; теперь она присутствует в языке только для обратной совместимости с программами на Delphi версий 3—5.

Области видимости, определяемые первыми тремя директивами, таковы.

* Поля, свойства и методы секции **public** не имеют ограничений на видимость. Они доступны из других функций и методов объектов, как в данном модуле, так и во всех прочих, ссылающихся на него.
* Поля, свойства и методы, находящиеся в секции **private**, доступны только в методах класса и в функциях, содержащихся в том же модуле, что и описываемый класс. Такая директива позволяет полностью скрыть детали внутренней реализации класса. Свойства и методы из секции **private** можно изменять, и это не будет сказываться на программах, работающих с объектами этого класса. Единственный способ для кого-то другого обратиться к ним — переписать заново созданный вами модуль (если, конечно, доступны исходные тексты).
* Поля, свойства и методы секции **protected** также доступны только внутри модуля с описываемым классом. Но — и это главное — они доступны в классах, являющихся потомками данного класса, в том числе и в других модулях. Такие элементы особенно необходимы для разработчиков новых компонентов — потомков уже существующих. Оставляя свободу модернизации класса, они все же скрывают детали реализации от того, кто только пользуется объектами этого класса.

**СОЗДАНИЕ НОВОГО КЛАССА**

**Объявление типа**

Для того чтобы создать новый класс, в interface-секции кода модуля следует записать:

**type**

TNewClass = class(ParentClass)

**end;**

Каждая форма в проекте, разрабатываемом в Delphi, описывается отдельным модулем (создаваемым автоматически при создании новой формы). Этот модуль описывает новый класс для компонента Form. Первоначально по умолчанию создается класс TForml, наследуемый от класса TForm из VCL-библиотеки. Это автоматически записывается в модуле следующим образом:

**type** (Объявление класса}

TForml = class(TForm)

**private**

[Объявление private переменных и методов}

**public**

{Объявление общедоступных переменных и методов}

**end;**

**var**

Forml: TForml; {Создание экземпляра класса}

**implementation**

{Секция реализации методов)

**end.**

**Объявление переменных и методов класса**

Переменные класса указываются после модификаторов доступа **(public, private**, **protected, published, automated**), определяющих их область видимости. Свойства, указанные после модификатора доступа published, являются общедоступными и отображаются в инспекторе объектов.

После имени переменной или списка имен, разделенных через запятую, указывается символ : и тип переменной. Типом может быть как базовый тип Delphi (например. Integer, Boolean), так и производный тип, в том числе реализуемый, как некоторый класс. Такой тип иногда называется объектным типом.

При объявлении методов класса перед именем метода указывается ключевое слово function или procedure. Для функций также после имени функции через символ указывается тип возвращаемого значения.

Например:

**type**

TNewClass = class(ParentClass]

{Модификатор доступа public)

**public**

Varl: Integer;

Var2, Var3: TVarTypeClass;

**procedure** P1;

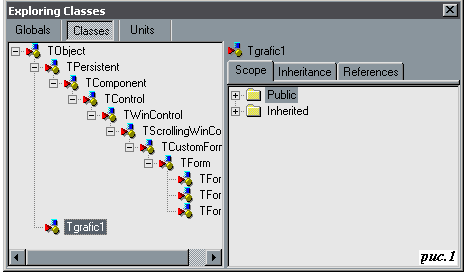
**function** F1: Integer;

**end;**

Объявление класса содержит только объявление переменных и методов. Реализация методов - функций и процедур - записывается в implementation-секции модуля.

Каждый модуль, создаваемый на основе разрабатываемой формы, представляет собой описание класса. Как правило, производного от класса TForm. Любой компонент, располагаемый в форме, также является экземпляром некоторого класса.

Классы в Delphi образуют иерархическое дерево. Будем называть классы из VCL-библиотеки Delphi базовыми классами. Иерархическое дерево для некоторого класса любого компонента имеет корневым элементом класс TObject. Просмотреть иерархию классов-потомков можно в окне Exploring. Для того чтобы перейти в него, достаточно выполнить команду меню **View|Browser** или нажать клавиши **Shift+CtrL+B.**

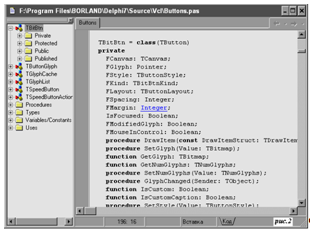


На рис.1 представлена страница Classes окна **Exploring Classes.** На ней отображено иерархическое дерево наследования для класса TForm 1. В правой части окна расположена панель, содержащая три страницы - **Scope,** Inheritance, **References.** Страница **Scope** содержит древовидную диаграмму всех объектов, переменных и методов выделенного на левой панели класса. При этом ветвь **Inherited** содержит имена класса-предка и класса-потомка. Страница **Inheritance** содержит поддерево иерархии классов, начиная с класса-предка для выделенного на левой панели класса.

На странице **References** можно узнать имена всех модулей и номера строк, в которых встречается имя данного класса.

Для того чтобы добавить в проект собственные описания производных классов, наиболее целесообразно создать отдельный модуль и в interface-секции модуля записать все объявления классов.

Все классы VCL-библиотеки Delphi разбиты на группы, которые расположены в каталоге **Delphi7\Source\VCL**. Для того чтобы просмотреть файл библиотеки, достаточно выполнить **File | Open** и выбрать каталог и имя файла. Справа в окне кода программы (рис.2) будет показан код модуля, а слева — список всех объявленных в нем классов.



**СВОЙСТВА / МЕТОДЫ И ОБРАБОТЧИКИ СОБЫТИЙ**

Каждый объект обладает набором свойств. Свойства могут быть как наследуемые от родительского класса, так и добавленные индивидуально для создаваемого объекта. Список всех свойств объекта и их значений отображается в диалоговом окне Object Inspector.

Ссылка на свойство в программном модуле записывается как

Имя\_объекта.Свойство;

Метод - это процедура или функция, ассоциируемая с некоторым объектом.

Ссылка на методов программном модуле записывается как

Имя\_0бъекта. Метод;

Delphi-приложение выполняется в среде Windows, и как любое Windows-приложение, получает сообщения о возникающих для него событиях. Управление приложением фактически сводится к обработке получаемых сообщений.

Методы, в которых содержится код обработки события, называются обработчиками событий (Event Handler). Delphi автоматически генерирует процедуры обработки событий – обработчики событий для любого компонента. При этом имя обработчика событий формируется из имени компонента и названия события (например, EditlClick). Имя обработчика события автоматически квалифицируется именем класса формы.

Например: TForml.ButtonlClick(Sender: TObject);.

Для каждого компонента предусмотрено одно стандартное событие. Например, для командной кнопки, флажка, списка, поля ввода - это событие Click, а для формы - событие FormCreate.

Для того чтобы автоматически добавить в модуль объявление и описание обработчика стандартного события, достаточно выполнить на компоненте формы или самой форме двойной щелчок мышью. Объявление события добавляется в interface-секцию модуля, а пустое описание события - в implementation-секцию модуля. Далее в редакторе кода внутри уже имеющегося блока begin end; следует только ввести код обработчика события.

Например:

**procedure** TForml.ButtonlClick(Sender: TObject);

**begin**

{место для введения кода}

**end;**

Для того чтобы добавить обработчик любого события, можно выбрать в инспекторе объектов страницу **Events** и выполнить двойной щелчок мышью в поле, расположенном справа от имени события. Это поле представляет собой комбинированное окно списка - в него можно вводить новое значение имени обработчика события двойным щелчком мыши или выбирать имя уже существующей процедуры. Это позволяет при необходимости определять одну процедуру обработки событий одновременно для нескольких событий.

**2. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ (ПО)**

**2.1 Назначение программного продукта**

Программа предназначена для решения заданных программно обыкновенных дифференциальных уравнений первого порядка методом Рунге – Кутта, вывода результата решения ОДУ на экран в виде графика в декартовой системе координат.

**2.2 Основные задачи**

Программа обеспечивает решение следующих задач:

− ввод исходных данных;

− решение ОДУ и вывод результата решения в численном и аналитическом виде;

− обнуление результатов решения ОДУ;

− контроль корректности ввода исходных данных и вывод на экран сообщение о содержании

ошибки с рекомендацией по её устранению;

− контроль возникновения в процессе вычислений ошибки «деление на ‘0’» и вывод на экран

соответствующего сообщения о содержании ошибки с рекомендацией по её устранению.

**2.3 Входные и выходные данные**

Входными данными для программы являются:

− начальное условие решения ОДУ − у'(х0) = у0.,

− начальное и конечное значения отрезка, в пределах которого находится решение ОДУ;

−величина шага дифференцирования,

−формула образцовой функции.

Выходными данными программы являются:

− массив (х1 ,у1 ; х2 ,у2;…; хi ,уi) − решений выбранного дифференциального уравнения на заданном

интервале;

− график функции, которая, будучи подставленной, в исходное образцовое уравнение, обращает его в

тождество и одновременно удовлетворяет начальному условию.

**3. ПРОЕКТИРОВАНИЕ**

**3.1 Выделение основных объектов ПО**

− Объект класса TRKutta (Form1) – главная окно программы.

− Объект класса TRngeKutta (Form2) – окно вывода графика функции-решения ДУ.

− Объект класса TSpravka (Form3) – окно «О программе».

− Объект класса TRungeKutta – координатная плоскость и график функции;

**3.2 Описание полей и методов**

**Класс TRKutta** **(**родительский класс **TForm)**

**Поля класса:**

x1 - значение x1(начало отрезка);

x2 - значение x2(конец отрезка);

yc - начальные значения (Y0) для передачи в графический модуль;

xc - начальные значения (х0)для передачи в графический модуль;

y - значение Y(x1);

h – зачение величины шага вычислений;

f - значение функции при начальных условиях

zx - массив значений аргумента;

zy - массив значений функции;

line\_arr - размерность массивa;

**Методы класса:**

procedure Button1Click – вычисление значений функции – решения ОДУ;

procedure Button2Click – очистка полей ввода/вывода данных;

procedure Button3Click - вывод окна ГРАФИК;

procedure Button4Click - выход из программы;

procedure RadioGroup1Click - выбор образцовой функции;

procedure Button5Click - активация ввода образцовых функций;

procedure Button6Click - деактивация ввода образцовых функций;

procedure N7Click - вывод информации о программе;

**Класс TRungeKutta (**родительский класс **TObject)**

**Поля класса**

x0,y0 - координаты начала координатных осей;

dx,dy - шаг координатной сетки (в пикселях)

y1,y2,xP,xL - засечки по оси Y и X

dlx,dly - шаг меток (оцифровки) линий сетки по X и Y

cross - счетчик неоцифрованных линий сетки

dcross - количество неоцифрованных линий между оцифрованными

razm – размер массивов;

uzx - Динамический массив координат-X

uzy - Динамический массив координат-Y

uxc,uyc - Оцифровка по осям

mx, my - масштабы по осям X и Y;

BaseMasht\_X,BaseMasht\_Y - МАХ значения элементов в массивах

**Методы класса**

function MaxAbsElementArray – определяет MAX элемент массива по модулю;

procedure UstanovkaParametrov – вычисляет исходные параметры необходимые для построения декартовой плоскости;

procedure KoordPloskost – вычерчивает координатную плоскость;

**Конструктор**

constructor TRungeKutta.CreateGr - создание объекта (график функции, координатная сетка, координатные оси)

**Класс Form2 (**родительский класс **TForm)**

**Поле класса (private)**

RungeKutta - переменная-объект класса **TRungeKutta ;**

**Методы класса**

procedure N4Click - вывод графика функции в окне 'График';

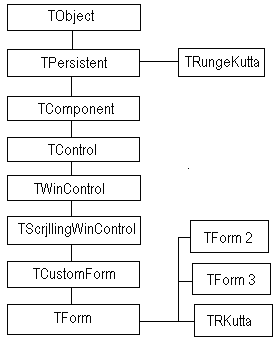
procedure N5Click - закрытие окна 'График';

**Класс Form3 (**родительский класс **TForm)**

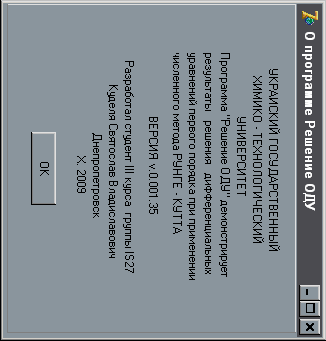
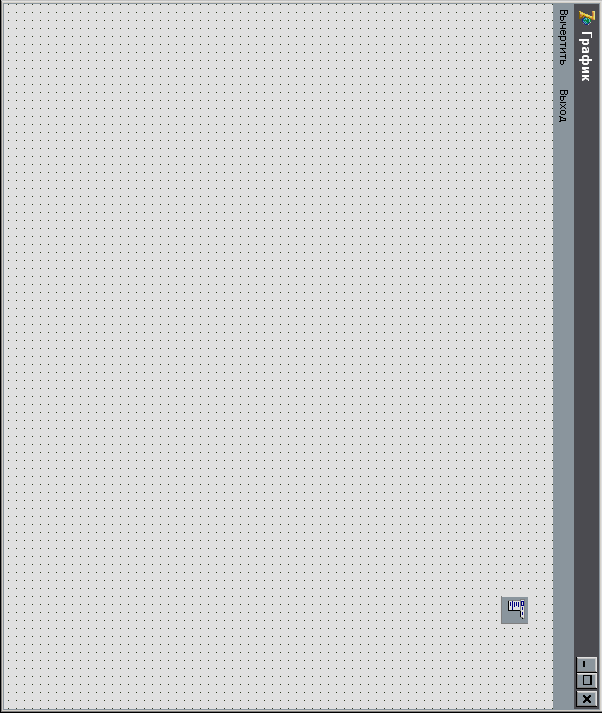
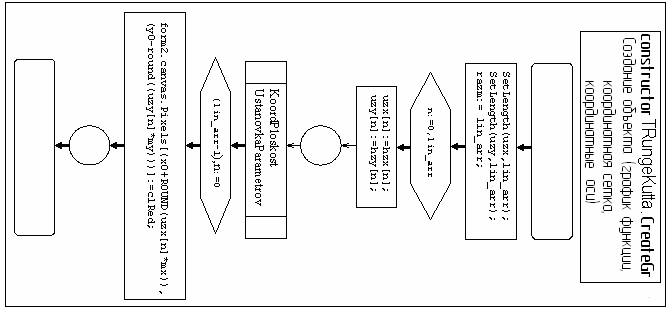
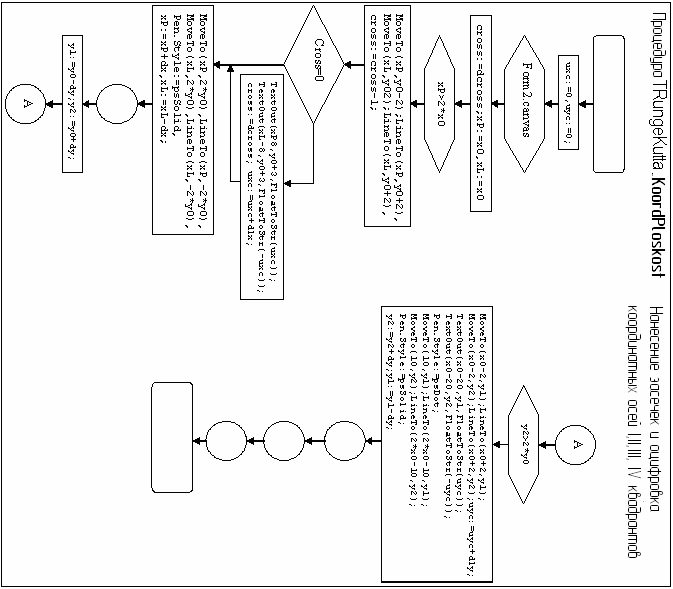
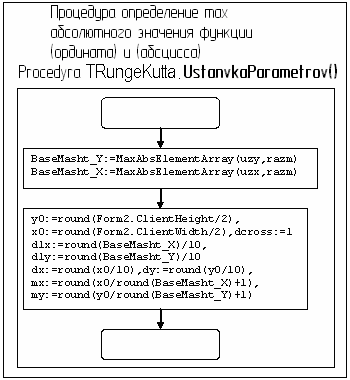
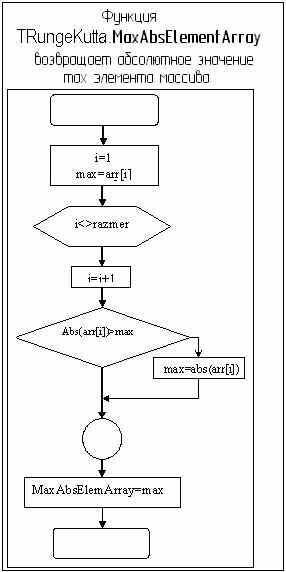
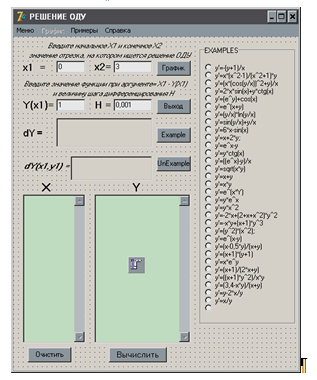
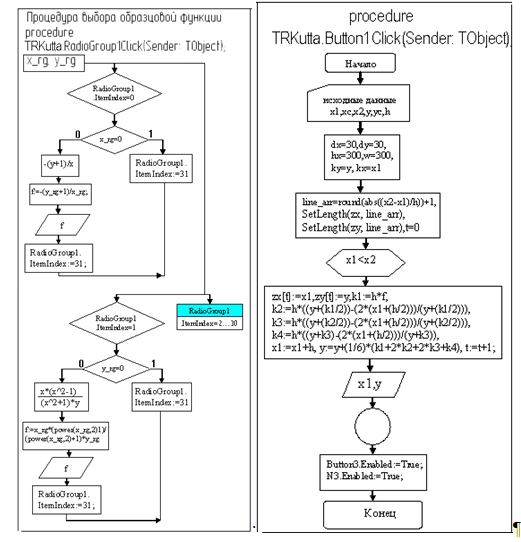
**Метод класса**

procedure Button1Click – процедура вывода информации о программе

**3.3 Иерархия классов на основе выделенных объектов**



**4. ОСНОВНЫЕ ФОРМЫ И КОМПОНЕНТЫ, ИСПОЛЬЗУЕМЫЕ ДЛЯ РЕАЛИЗАЦИИ ПРОГРАММЫ. ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ РЕШЕНИЯ РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ**



**РУКОВОДСТВО**

**Теоретический материал**

**1. Решение дифференциальных уравнений**

Пусть дано дифференциальное уравнение первого порядка:

у' = f(x)

и начальное условие его решения:

у'(х0) = у0..

Тогда решить уравнение — это значит найти такую функцию у — φ(х), которая, будучи подставленной, в исходное уравнение, обратит его в тождество и одновременно будет удовлетворено начальное условие. Задача отыскания функции у = φ (х) называется в математике задачей Коши. При решении дифференциального уравнения порядка n задача Коши формулируется следующим образом.

Дано дифференциальное уравнение порядка n:

у(n) = f(x, y, у'’,…,yn-1)

Необходимо найти такую функцию у = φ (х), которая, будучи подставленной в исходное уравнение, обратит его в тождество и одновременно будут удовлетворены следующие п начальных условий:

у(х0) = у0

у'(х0) = у'0

. . .

уn-1(х0) = уn-10

**4.1 Метод Рунге-Кутта**

Метод Рунге-Кутта обладает более высокой точностью, чем методы Эйлера за счет снижения методических ошибок. Идея метода состоит в следующем.

По методу Эйлера решение дифференциального уравнения первого порядка определяется из соотношения:

yi+1 = yi + Δyi;

где Δyi = hf (хi, yi) = hу' (хi, yi).

Тогда приращение **Δyi**, может быть найдено путем интегрирования:



Или окончательно



Вычислим теперь интеграл по методу прямоугольников:

yi+1 = yi + (xi+1 - xi)f(xi, yi) = yi + hf(xi, yi).

Из полученного выражения видно, что вычисление интеграла по методу прямоугольников приводит к формуле Эйлера.

Вычислим интеграл по формуле трапеций:

yi+1 = yi +0,5h(f(xi, yi)+ f(xi+1, yi+1))

Из выражения видно, что оно совпадает с расчетной формулой усовершенствованного метода Эйлера-Коши. Для получения более точного решения дифференциального уравнения следует воспользоваться более точными методами вычисления интеграла. В методе Рунге-Кутта искомый интеграл представляется в виде следующей конечной суммы:



где Pi — некоторые числа, зависящие от q ; Ki(h) — функции, зависящие от вида подынтегральной функции f(x,y) и шага интегрирования h, вычисляемые по следующим формулам:

K1(h) = hf(x, y);

K2(h) = hf(x + a2h, y + β21K1(h));

K3(h) = hf(x + a3h, y + β 31K1(h) + β 32K2(h));

Kn(h) = hf(x + aqh, , y + β q1K1(h) + ... + β q,q-1Kq-1(h)).

Значения p, α, β получают из соображений высокой точности вычислений. Формулы Рунге-Кутта третьего порядка (q= 3) имеют следующий вид:



K1=hf(xi, yi);

K2=hf(xi + 0,5h, yi+0,5 K1);

K3=hf(xi+h, yi+K1+2K2).

Наиболее часто используется метод Рунге-Кутта четвертого порядка, для которого расчетные формулы имеют следующий вид:



K1=hf(xi, yi);

K2=hf(xi + 0,5h, yi+0,5 K1);

K3=hf(xi+0,5h, yi+0,5K2).

K3=hf(xi+h, yi+K3).

Формулы Рунге-Кутта имеют погрешности порядка h q+1 . Погрешность метода Рунге-Кутта четвертого порядка имеет порядок h5

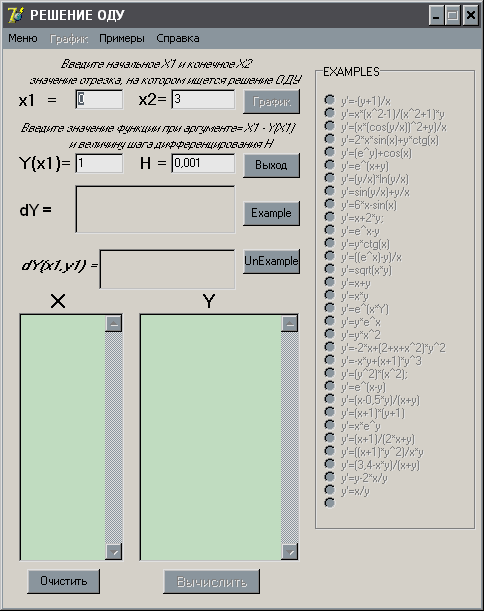
**4.2 Описание программы ” РЕШЕНИЕ ОДУ “**

Программа ”Решение ОДУ“ достаточно проста в использовании.

При запуске программы открывается главное окно программы (**рис. 4**), с установленными по умолчанию начальными условиями в полях ввода.

**Назначение элементов ввода данных.**

1. Поля **X1, X2, Y(x1), H** предназначены для ввода начального и конечного значений отрезка, на котором ищется решение дифференциального уравнения, значения функции при аргументе равном **Х1** и величины шага дифференцирования;
2. В поле **dY** выводится формула дифференциального уравнения 1-й степени, выбранная для решения;
3. В поле **dY(x1,y1)** выводится значение производной в исходной точке.

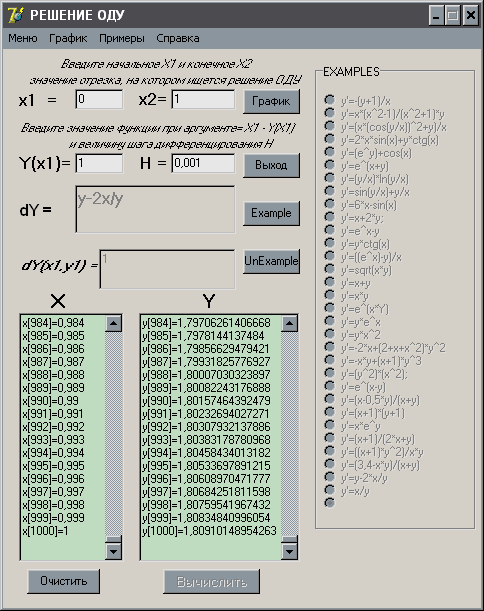


**Рис.4**

**Назначение элементов управления и контроля.**

1. При нажатии кнопки **EXAMPLE** активируются “радиокнопки” выбора уравнений;
2. Щелчком “радиокнопки” выбирается соответствующее ей уравнение, вид формулы контролируется по её отображению в поле **dY** ;
3. Щелчком по кнопке **ВЫЧИСЛИТЬ** находятся приближенные решения выбранного дифференциального уравнения на заданном интервале;
4. Решения дифференциального уравнения в виде пар значений **X - Y** выводятся в поля **X** и **Y; (рис. 5.)**

По окончании вычислений активируются кнопка **ГРАФИК** и пункт меню **ГРАФИК** главного окна системы.

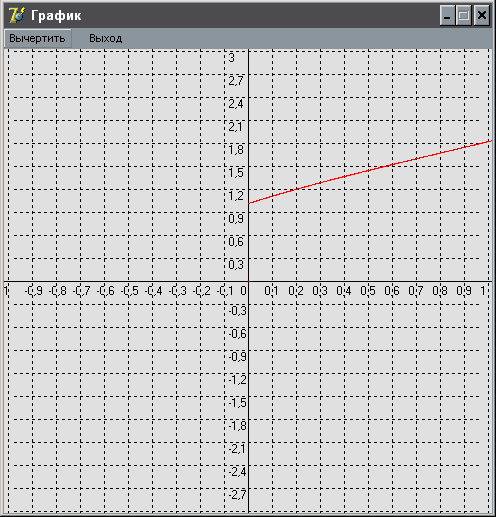


**Рис.5**

**4.3 Назначение элементов графического окна программы**

Вход в графическое окно осуществляется с помощью кнопок **ГРАФИК** наглавной формеили пункт меню **ГРАФИК (рис. 6).**

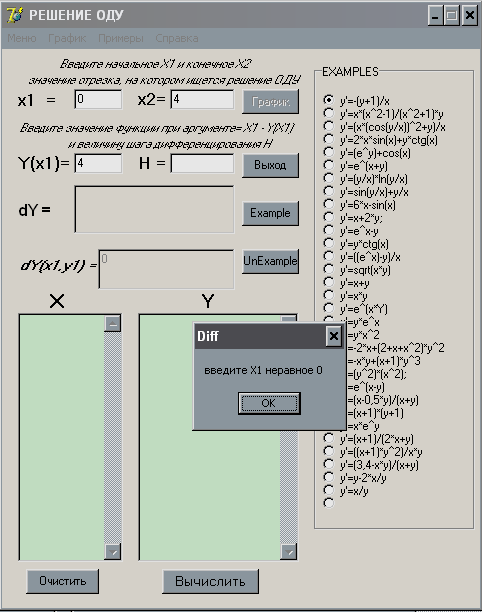
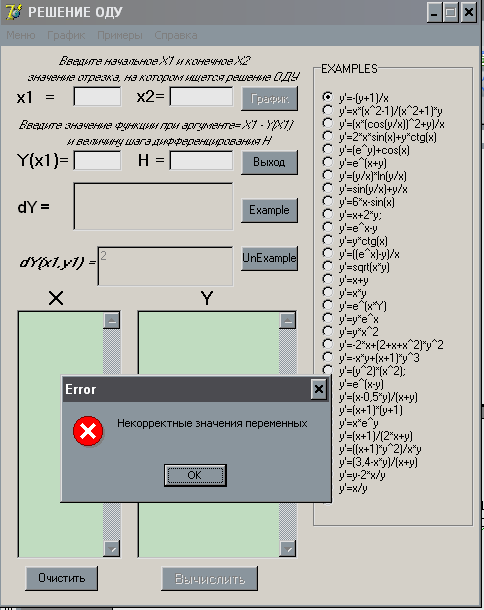
С помощью кнопки **ВЫЧЕРТИТЬ** на координатную плоскость выводится график функции – решение дифференциального уравнения на заданном интервале.



**рис.6**

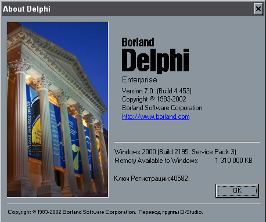
**4.4 Реакция программы при возникновении ошибок**

При вводе пользователем ошибочных данных (отсутствии начальных условий, некорректных значений переменных) программа выдает сообщение об ошибке (рис**.7** а**,** б) рис**.7**а**.** рис**.7**б**.**



**рис 7а Рис.7б**

**Версия DELPHI**



**4.5 Перечень компонент DELPHI использованных в** **программе**

В **Form1** использованы компоненты:

- Edit1.text, Edit2.text, Edit3.text, Edit4.text – для ввода начальных условий дифференциального

уравнения

- Memo4.TMemo – для вывода формулы уравнения;

- Memo1.TMemo, Memo2.TMemo - для вывода результатов вычислений;

- Memo3.TMemo – для вывода значения производной в точке (Х0,Y0)

- ScrollBars ssVertical в свойствах Memo1.TMemo, Memo2.TMemo;

- Button1 “Вычислить”, Button2 “Очистить”, Button3 “График”, Button4 “Выход”,

Button5 “Example”, Button6 “UnExample”;

- Label1.TLabel - Label9.TLabel – для отображения назначения компонентов Memo и Edit;

- RadioGroup – для выбора вида уравнения;

- MainMenu ;

В **Form2** использованы компоненты:

- MainMenu - для построения графика;

В **Form3** использованы компоненты:

- Panel1.T Panel – для размещения информации о программе;

- Label1.TLabel – Label14.TLabel – для отображения информации о программе;

- Button1.T Button “OK” – для выхода из окна

**5. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ И ТРЕБОВАНИЯ К ПО**

**Технические характеристики**

Программа работает в среде операционных систем Windows 9х, NT.

**Требования к ПО**

Минимальные системные требования

a процессор Intel 486 с рабочей частотой 66 MHz и выше;

b) операционная система Windows 95, 98, NT 4.0, 2000, XP;

с) 16 Мбайт оперативной памяти (или более);

d) 3 Мбайт свободного пространства на жёстком диске.

**6. ТЕКСТ ПРОГРАММЫ**

**Код программы**

**unit RKt;**

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,

StdCtrls, CheckLst, ComCtrls, ExtCtrls,math, Menus;

type

TRKutta = class(TForm)

Label2: TLabel;

Label3: TLabel;

Label5: TLabel;

Label6: TLabel;

Edit1: TEdit;

Edit2: TEdit;

Edit3: TEdit;

Edit4: TEdit;

Memo1: TMemo;

Memo2: TMemo;

Button1: TButton;

Button2: TButton;

Label4: TLabel;

Label7: TLabel;

Button3: TButton;

Button4: TButton;

Label9: TLabel;

RadioGroup1: TRadioGroup;

Button5: TButton;

Memo3: TMemo;

Button6: TButton;

MainMenu1: TMainMenu;

N1: TMenuItem;

N3: TMenuItem;

N4: TMenuItem;

Example1: TMenuItem;

UnExample1: TMenuItem;

N5: TMenuItem;

N7: TMenuItem;

N2: TMenuItem;

N9: TMenuItem;

N8: TMenuItem;

Label1: TLabel;

Memo4: TMemo;

Label8: TLabel;

Label10: TLabel;

Label11: TLabel;

Label12: TLabel;

procedure Button1Click(Sender: TObject);

procedure Button2Click(Sender: TObject);

procedure Button3Click(Sender: TObject);

procedure Button4Click(Sender: TObject);

procedure RadioGroup1Click(Sender: TObject);

procedure Button5Click(Sender: TObject);

procedure Button6Click(Sender: TObject);

procedure N7Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

RKutta: TRKutta;

x1,x2,yc,xc,y,h:extended;

line\_arr:integer; // размерность массивa

f:real; // значение функции при начальных условиях

zx:array of real;

zy:array of real;

implementation

uses RungeKutta, Spravka;

{$R \*.DFM}

procedure TRKutta.Button1Click(Sender: TObject);

var k1,k2,k3,k4:extended;

t:integer;

begin

Memo1.Clear;

Memo2.Clear;

memo1.Enabled:=true;

memo2.Enabled:=true;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=False;

//Проверка возможности ввода начальных условий и инициализация переменных

try

// инструкции, которые могут вызвать исключение (ошибку)

x1:=StrToFloat(Edit1.Text); //инициализация(ввод) x1(начало отрезка)

xc:=StrToFloat(Edit1.Text); //передача начальных условий (х0)в графический модуль

x2:=StrToFloat(Edit2.Text); //инициализация(ввод) x2(конец отрезка)

y:=StrToFloat(Edit3.Text); //инициализация(ввод) Y(x1)

yc:=StrToFloat(Edit3.Text); //передача начальных условий (Y0)в графический модуль

h:=StrToFloat(Edit4.Text); //инициализация(ввод) H -величины шага вычислений

//определение размера массивов содержащего значения аргумента и функции

line\_arr:=round(abs((x2-x1)/h))+1;

//Установка размера динамических массивов zx, zy

SetLength(zx, line\_arr);

SetLength(zy, line\_arr);

t:=0; // счётчик

while x1<x2

do

begin

zx[t]:=x1;

zy[t]:=y;

k1:=h\*f;

if (y+(k1/2))=0 then begin showmessage ('Деление на 0!'+#13+' Измените h'); break;

end;

k2:=h\*((y+(k1/2))-(2\*(x1+(h/2)))/(y+(k1/2)));

if (y+(k2/2))=0 then begin showmessage ('Деление на 0!'+#13+' Измените h'); break;

end;

k3:=h\*((y+(k2/2))-(2\*(x1+(h/2)))/(y+(k2/2)));

if (y+k3)=0 then begin showmessage ('Деление на 0!'+#13+' Измените h');break;

end;

k4:=h\*(y+k3-2\*(x1+(h/2)))/(y+k3);

x1:=x1+h;

y:=y+(1/6)\*(k1+2\*k2+2\*k3+k4);

t:=t+1;

Memo1.Lines.Add('x['+floattostr(t)+']='+floattostr(x1));

Memo2.Lines.Add('y['+floattostr(t)+']='+floattostr(y));

end;

except

on EConvertError do // невозможно преобразовать строку символов в число

begin

MessageDlg('Некорректные значения переменных',mtError,[mbOk],0);

exit;

end;

end;

{ | Подключение кнопок: |

| - Button3 - 'ГРАФИК': входа в графический модуль |

| - нопки N3 - 'EXAMPLE': образцовых функций | }

if ((y+(k1/2))=0) or ((y+(k2/2))=0) or((y+k3)=0) then

begin

Button3.Enabled:=False;

N3.Enabled:=False;

end

else

begin

Button3.Enabled:=True;

N3.Enabled:=True;

end;end;

//------------------------------------------------------------------------------

procedure TRKutta.Button2Click(Sender: TObject);

begin

memo1.Clear;

MEMO2.Clear;

MEMO3.Clear;

memo4.Clear;

edit1.Clear;

edit2.Clear;

edit3.Clear;

edit4.Clear;

end;

//------------------------------------------------------------------------------

{Процедура вывода окна ГРАФИК}

procedure TRKutta.Button3Click(Sender: TObject);

begin

Button3.enabled:=false;

N3.Enabled:=false;

Form2.ShowModal;

end;

//------------------------------------------------------------------------------

{Процедура выхода из программы}

procedure TRKutta.Button4Click(Sender: TObject);

begin

Close;

end;

//------------------------------------------------------------------------------

{Процедура выбора образцовой функции}

procedure TRKutta.RadioGroup1Click(Sender: TObject);

var x\_rg,y\_rg:extended;

begin

try

y\_rg:=strtofloat(edit3.Text); //ввод Y(x1)

x\_rg:=strtofloat(edit1.Text); //ввод X1

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=0 then

begin

if x\_rg=0 then

begin

ShowMessage('введите X1 неравное 0');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo3.clear;

Memo4.Clear;

Memo4.Lines.Add(' -(y+1)/x '); вывод формулы функции в окно "dY"

f:=-(y\_rg+1)/x\_rg; //вычисление значения dY

Memo3.lines.Add(''+floattostr(f)); //вывод значения dY в окно "dY(x1,y1)"

// сброс радиокнопки (с целью корретного вычисления

// значения dy(x1,y1) при вводе других данных)

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=1 then

begin

if y\_rg=0 then

begin

ShowMessage('введите Y(x1) неравное 0');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo4.Clear;

Memo4.Lines.Add('x\*(x^2-1)/(x^2+1)\*y ');

f:=x\_rg\*(power(x\_rg,2)-1)/(power(x\_rg,2)+1)\*y\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=2 then

begin

if x\_rg=0 then

begin

ShowMessage('введите X1 неравное 0');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo3.Clear;

Memo4.Clear;

Memo4.Lines.Add('(x\*cos(y/x))^2+y)/x ');

f:=x\_rg\*(POWER(cos(y\_rg/x\_rg),2)+y\_rg)/x\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=3 then

begin

if x\_rg=0 then

begin

ShowMessage('введите X1 неравное 0');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo3.Clear;

Memo4.Clear;

Memo4.Lines.Add('2\*x\*sin(x)+y\*ctg(x) ');

f:=2\*x\_rg\*sin(x\_rg)+y\_rg\*cot(x\_rg);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=4 then

begin

Memo3.Clear;

Memo4.Clear;

Memo4.Lines.Add('((e^y)+cos(x) ');

f:=(exp(y\_rg))+cos(x\_rg);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=5 then

begin

Memo3.Clear;

Memo4.Clear;

Memo4.Lines.Add('e^(x+y) ');

f:=exp(y\_rg+x\_rg);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=6 then

begin

if ((x\_rg<=0) and (y\_rg>=0))or((x\_rg>=0) and (y\_rg<=0)) then

begin

ShowMessage('X1 / Y(X1) не должны быть <=0');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo4.Clear;

Memo4.Lines.Add('(y/x)ln(y/x) ');

f:=(y\_rg/x\_rg)\*ln(y\_rg/x\_rg);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=7 then

begin

if x\_rg=0 then

begin

ShowMessage('введите X1 неравное 0');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo4.Clear;

Memo4.Lines.Add(' sin(y/x)+y/x ');

f:=sin(y\_rg/x\_rg)+y\_rg/x\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=8 then

begin

Memo4.Clear;

Memo4.Lines.Add(' 6x-sin(x) ');

f:=6\*x\_rg-sin(x\_rg);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=9 then

begin

Memo4.Clear;

Memo4.Lines.Add(' x+2y ');

f:=x\_rg+2\*y\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=10 then

begin

Memo4.Clear;

Memo4.Lines.Add(' e^x-y ');

f:=exp(x\_rg)-y\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=11 then

begin

if x\_rg=0 then

begin

ShowMessage('введите X1 не равное 0');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo4.Clear;

Memo4.Lines.Add(' y\*ctg(x) ');

f:=cot(x\_rg)\*y\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=12 then

begin

if x\_rg=0 then

begin

ShowMessage('введите X1 не равное 0');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo4.Clear;

Memo4.Lines.Add('(e^x-y)/x ');

f:=(exp(x\_rg)-y\_rg)/x\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=13 then

begin

if x\_rg\*y\_rg<0 then

begin

ShowMessage('введите X1\*Y(X1) >= 0');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo4.Clear;

Memo4.Lines.Add('(x\*y)^(1/2) ');

f:=power(x\_rg\*y\_rg,1/2);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=14 then

begin

Memo4.Clear;

Memo4.Lines.Add(' x+y ');

f:=x\_rg+y\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=15 then

begin

Memo4.Clear;

Memo4.Lines.Add(' x\*y ');

f:=x\_rg\*y\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=16 then

begin

Memo4.Clear;

Memo4.Lines.Add('e^(x\*y) ');

f:=exp(x\_rg\*y\_rg);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=17 then

begin

Memo4.Clear;

Memo4.Lines.Add('(e^x\*y) ');

f:=exp(x\_rg)\*y\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=18 then

begin

Memo4.Clear;

Memo4.Lines.Add('(x^2)\*y ');

f:=power(x\_rg,2)\*y\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=19 then

begin

Memo4.Clear;

Memo4.Lines.Add('-2\*x+(2+x+x^2)\*y^2 ');

f:=-2\*x\_rg+(2+x\_rg+power(x\_rg,2))\*power(y\_rg,2);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=20 then

begin

Memo4.Clear;

Memo4.Lines.Add('-x\*y+(x+1)\*y^2 ');

f:=-x\_rg\*y\_rg+(x\_rg+1)\*power(y\_rg,3);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=21 then

begin

Memo4.Clear;

Memo4.Lines.Add(' x^2\*y^2 ');

f:=power(x\_rg,2)\*power(y\_rg,2);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=22 then

begin

Memo4.Clear;

Memo4.Lines.Add(' e^(x-y) ');

f:=exp(x\_rg-y\_rg);

Memo3.clear;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=23 then

begin

if (x\_rg=-y\_rg) then

begin

ShowMessage('введите Y(x1)неравное X1');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo4.Clear;

Memo4.Lines.Add('(x-(y/2))/(x+y) ');

f:=(x\_rg-0.5\*y\_rg)/(x\_rg+y\_rg);

Memo3.clear;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=24 then

begin

Memo4.Clear;

Memo4.Lines.Add('(x+1)\*(y+1) ');

f:=(x\_rg+1)\*(y\_rg+1);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=25 then

begin

Memo4.Clear;

Memo4.Lines.Add('x\*e^(y) ');

f:=x\_rg\*exp(y\_rg);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=26 then

begin

if (2\*x\_rg=-y\_rg) then

begin

ShowMessage('введите Y(x1) неравное -2\*X1');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo4.Clear;

Memo4.Lines.Add('(x+1)/(2\*x+y) ');

f:=(x\_rg+1)/(2\*x\_rg+y\_rg);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=27 then

begin

if (x\_rg=0)or(y\_rg=0) then

begin

ShowMessage('введите Y(x1) неравное 0');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo4.Clear;

Memo4.Lines.Add('((x+1)\*(y^2))/x\*y ');

f:=((x\_rg+1)\*power(y\_rg,2))/x\_rg\*y\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=28 then

begin

if (x\_rg=-y\_rg) then

begin

ShowMessage('введите Y(x1) неравное -X1');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo4.Clear;

Memo4.Lines.Add('(3.4-x\*y)/(x+y) ');

f:=(3.4-x\_rg\*y\_rg)/(x\_rg+y\_rg);

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=29 then

begin

if y\_rg=0 then

begin

ShowMessage('введите Y(x1) не равное 0');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo4.Clear;

Memo4.Lines.Add('y-2x/y ');

f:=y\_rg-2\*x\_rg/y\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end;

{------------------------------------------------------------------------------}

if RadioGroup1.ItemIndex=30 then

begin

if y\_rg=0 then

begin

ShowMessage('введите Y(x1) не равное 0');

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end

else

begin

Memo4.Clear;

Memo4.Lines.Add(' x/y ');

f:=x\_rg/y\_rg;

Memo3.lines.Add(''+floattostr(f));

Button1.Enabled:=True;

RadioGroup1.Enabled:=False;

RadioGroup1.ItemIndex:=31;

end;

end

{------------------------------------------------------------------------------}

except

on EConvertError do // невозможно преобразовать строку символов в число

begin

MessageDlg('Некорректные значения переменных',mtError,[mbOk],0);

exit;

end;

end;

end;

//------------------------------------------------------------------------------

{Активация ввода образцовых функций}

procedure TRKutta.Button5Click(Sender: TObject);

begin

memo4.clear;

memo3.clear;

RadioGroup1.Enabled:=True; memo1.clear;memo2.Clear;memo3.clear;

memo3.lines.Add(''+floattostr(f));

end;

//------------------------------------------------------------------------------

{Деактивация ввода образцовых функций}

procedure TRKutta.Button6Click(Sender: TObject);

begin

RadioGroup1.Enabled:=False;

Memo3.lines.Add(''+floattostr(f));

end;

//------------------------------------------------------------------------------

{Вывод информации о программе}

procedure TRKutta.N7Click(Sender: TObject);

begin

Form3.ShowModal;

end;

//------------------------------------------------------------------------------

end.

{====================================================}

**unit RngeKutta;**

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs,StdCtrls,Rkt,Menus;

type

TRungeKutta=class(TObject)

constructor CreateGr(var hzx:array of real;var hzy:array of real;var lin\_arr:integer);

private

{ Private declarations }

public

{ Public declarations }

x0,y0:integer; //координаты начала координатных осей, 2\*x0,2\*y0 - высота

// и ширина области вывода координатной сетки

dx,dy:integer; // шаг координатной сетки (в пикселях)

y1,y2,xP,xL:integer;// засечки по оси Y и X

dlx,dly:real; // шаг меток (оцифровки) линий сетки по X и Y

cross:integer; // счетчик неоцифрованных линий сетки

dcross:integer; // количество неоцифрованных линий между оцифрованными

BaseMasht\_X,BaseMasht\_Y:extended; //МАХ значения элементов в массивах

razm:integer;

uzx:array of real; //Динамический массив координат-X

uzy:array of real; //Динамический массив координат-Y

uxc,uyc:extended; // Оцифровка по осям

mx,my:extended;

function MaxAbsElementArray(arr:array of real;razmer:integer):real;

procedure UstanovkaParametrov();

procedure KoordPloskost;

end;

TForm2 = class(TForm)

MainMenu1: TMainMenu;

N4: TMenuItem;

N5: TMenuItem;

procedure N4Click(Sender: TObject);

procedure N5Click(Sender: TObject);

private

{ Private declarations }

RungeKutta:TRungeKutta; //Объявление переменной - объекта класса

public

{ Public declarations }

end;

var

Form2: TForm2;

implementation

{$R \*.dfm}

function TRungeKutta.MaxAbsElementArray(arr:array of real;razmer:integer):real;

var i:integer;

max:real;

begin

i:=1;

max:=arr[1];

while i<>razmer do

begin

i:=i+1;

if abs(arr[i])>max then

begin

max:=abs(arr[i]);

end;

end;

MaxAbsElementArray:=max;

end;

//------------------------------------------------------------------------------

procedure TRungeKutta.UstanovkaParametrov();

begin

//определение max абсолютного значения функции (ордината)

BaseMasht\_Y:=MaxAbsElementArray(uzy,razm)+1;

//определение max абсолютного значения аргумента функции (абсцисса)

BaseMasht\_X:=MaxAbsElementArray(uzx,razm);

y0:=round(Form2.ClientHeight/2); // оси начинаются в точке (X0,Y0)

x0:=round(Form2.ClientWidth/2);

dcross:=1; // помечать линии сетки X: 1 - каждую;

// 2 - через одну;

// 3 - через две;

dlx:=round(BaseMasht\_X)/10; // вес меток оси X

dly:=round(BaseMasht\_Y)/10; // вес меток оси Y ,

dx:=round(x0/10); // шаг координатной сетки в пикселях

dy:=round(y0/10); // шаг координатной сетки в пикселях

mx:=round(x0/round(BaseMasht\_X)+1);//вычисление масштаба по оси X

my:=round(y0/round(BaseMasht\_Y)+1);//вычисление масштаба по оси Y

end;

//------------------------------------------------------------------------------

procedure TRungeKutta.KoordPloskost;

begin

uxc:=0;

uyc:=0;

with form2.Canvas do

begin

cross:=dcross;

//------------------------------------------------------------------------------

//Нанесение засечек и оцифровка координатных осей I,II квадрантов

// засечки и оцифровка по оси X

xP:=x0;

xL:=x0;

repeat

MoveTo(xP,y0-2);LineTo(xP,y0+2); // засечка

MoveTo(xL,y0-2);LineTo(xL,y0+2); // засечка

cross:=cross-1;

if cross = 0 then //оцифровка

begin

TextOut(xP-8,y0+3,FloatToStr(uxc));

TextOut(xL-8,y0+3,FloatToStr(-uxc));

cross:=dcross;

uxc:=uxc+dlx;

end;

Pen.Style:=psDot;

MoveTo(xP,2\*y0);LineTo(xP,-2\*y0); // линия сетки

MoveTo(xL,2\*y0);LineTo(xL,-2\*y0);

Pen.Style:=psSolid;

xP:=xP+dx;

xL:=xL-dx;

until (xP>2\*x0);

//------------------------------------------------------------------------------

//Нанесение засечек и оцифровка координатных осей III, IV квадрантов

// засечки, и оцифровка по оси Y

y1:=y0-dy;

y2:=y0+dy;

repeat

MoveTo(x0-2,y1);LineTo(x0+2,y1); // засечка

MoveTo(x0-2,y2);LineTo(x0+2,y2);

uyc:=uyc+dly;

TextOut(x0-20,y1,FloatToStr(uyc)); // оцифровка

TextOut(x0-20,y2,FloatToStr(-uyc));

Pen.Style:=psDot;

MoveTo(10,y1);LineTo(2\*x0-10,y1); // линия сетки

MoveTo(10,y2);LineTo(2\*x0-10,y2);

Pen.Style:=psSolid;

y2:=y2+dy;

y1:=y1-dy;

until (y2>2\*y0);

//------------------------------------------------------------------------------

MoveTo(x0,2\*y0); LineTo(x0,0); // ось Y

MoveTo(0,y0); LineTo(2\*x0,y0); // ось X

end;

end;

//-----------------------------------------------------------------------------

//создание объекта (график функции, координатная сетка, координатные оси)

constructor TRungeKutta.CreateGr(var hzx:array of real;var hzy:array of real;var lin\_arr:integer);

var n:integer;

begin

SetLength(uzx,lin\_arr); //Установка размера массива uzx

SetLength(uzy,lin\_arr); //Установка размера массива uzy

razm:= lin\_arr;

for n:=0 to lin\_arr do

begin

uzx[n]:=hzx[n]; //инициализация массива uzx

uzy[n]:=hzy[n]; //инициализация массива uzy

end;

UstanovkaParametrov;

KoordPloskost;

for n:=0 to (lin\_arr-1) do

begin

form2.canvas.Pixels[(x0+ROUND(uzx[n]\*mx)),(y0-round((uzy[n]\*my)))]:=clRed;

end;

end;

//------------------------------------------------------------------------------

{Процедура N4Click выводит график функции в установленном масштабе в окне 'График' }

procedure TForm2.N4Click(Sender: TObject);

begin

form2.Canvas.FillRect(Rect(0,0,ClientWidth,ClientHeight));

TRungeKutta.CreateGr(zx,zy,line\_arr);

end;

//------------------------------------------------------------------------------

{Процедура N5Click закрывает окно вывода графика функции - результата решния ДУ }

procedure TForm2.N5Click(Sender: TObject);

begin

close;

end;

end.

{====================================================}

**Unit Spravka;**

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, ExtCtrls, StdCtrls;

type

TForm3 = class(TForm)

Panel1: TPanel;

Label1: TLabel;

Button1: TButton;

Label2: TLabel;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

Label7: TLabel;

Label8: TLabel;

Label9: TLabel;

Label10: TLabel;

Label6: TLabel;

Label11: TLabel;

Label12: TLabel;

Label13: TLabel;

Label14: TLabel;

procedure Button1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form3: TForm3;

implementation

{$R \*.dfm}

//------------------------------------------------------------------------------

{Ввывод информации о программе}

procedure TForm3.Button1Click(Sender: TObject);

begin

Caption := 'OK';

ModalResult := mrOK;

OnClick := Button1Click;

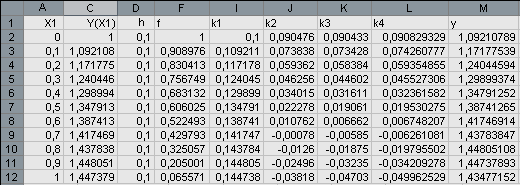
end;

end.

{===============================================}

**7. РЕШЕНИЕ ДИФФЕРЕНЦИАЛЬНОГО УРАВНЕНИЯ Y = Y−2X/Y МЕТОДОМ РУНГЕ – КУТТА В СРЕДЕ EXCEL**

дифференциальное уравнение программирование



**2.** Решение дифференциального уравнения **Y = Y−2X/Y** программой **«РЕШЕНИЕ ОДУ»** показано на рис. 5 и рис. 6.(стр. 22 - 23 )

**ВЫВОД**

Программа **«Решение ОДУ»** решает заданные дифференциальные уравнения первой степени с достаточно высокой степенью точности.