МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Физико-технический факультет**

**Кафедра теоретической физики и компьютерных технологий**

**КУРСОВОЙ ПРОЕКТ**

**РАЗРАБОТКА МОДУЛЯ ОБРАЗОВАТЕЛЬНОГО ПРИЛОЖЕНИЯ**

**”НЕЛИНЕЙНЫЕ КОЛЕБАНИЯ”**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Приймачук Виктор Сергеевич

Курс 3

Направление 09.03.02 Информационные системы и технологии

Научный руководитель

преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.В. Сухачев

Нормоконтролер инженер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г. Д. Цой

Краснодар 2018

**Содержание**

|  |
| --- |
| Введение……………………………………………………………………………... 3 |
| 1 Теоретическая основа разработки приложения……………………………….….4 |
| 1.1 Нелинейные колебания и двойной маятник……………………………….....4 |
| 1.2 Численное моделирование хаотических колебаний………………………….5 |
| 1.3 Язык программирования С++………………………………………………….9 |
| 1.4 QT – кросс‑платформенный инструмент разработчика……………………10 |
| 2 Проектирование и разработка приложения……………………………………...14 |
| 2.1 Визуальная составляющая приложения……………………………………..14 |
| 2.2 Логическая составляющая приложения……………………………………..16 |
| Заключение…………………………………………………………………………..27 |
| Список использованных источников……………………………………………....28 |

**ВВЕДЕНИЕ**

Информатизация системы образования является одной из форм проникновения компьютерных систем во все сферы общественной жизни. В таких системах компьютерный компонент выступает как инструмент эффективной реализации человеческой деятельности. В настоящее время создано большое количество обучающих компьютерных программ тренажерного типа, которые реализуют процесс усвоения знаний. Такие системы позволили автоматизировать некоторые операции преподавательского труда. Системы «обучающийся — компьютер», могут быть эффективными, несмотря на то, что в них в явном виде отсутствует преподаватель. Часть его функций может выполнить сам компьютер. Реализуется такое обучение с помощью компьютерных программ, использующих учебные модели физических объектов, явлений и процессов.

Цель моего курсового проекта заключается в создании модуля образовательного приложения на тему “Нелинейные колебания” на примере хаотического поведения двойного маятника.

**1 Теоретическая основа разработки**

**1.1 Нелинейные колебания и двойной маятник**

Нелинейные колебания - колебания в физические системах, описываемые нелинейными системами обыкновенных дифференциальных уравнений

, где , X содержит члены не ниже 2-й степени по компонентам вектора x, f(t) – вектор-функция времени t, – малый параметр (либо и Х = Х(t, x)). Все физические системы являются нелинейными. Одна из наиболее характерных особенностей нелинейных колебаний - это нарушение в них принципа суперпозиции колебаний: результат каждого из воздействий в присутствии другого оказывается иным, чем в случае отсутствия другого воздействия [1].

Маятник — система, подвешенная в поле тяжести и совершающая механические колебания. Колебания совершаются под действием силы тяжести, силы упругости и силы трения. Во многих случаях трением можно пренебречь.

Двойной маятник, который состоит из одного маятника, подвешенного к другому, является потенциально хаотической системой. Это означает, что при определенном диапазоне параметров незначительное изменение одного из начальных условий запуска может оказать существенное влияние на последующее движение маятника. В результате движение двойного маятника предельно сложно предсказать-проявляя, казалось бы, случайное или хаотичное поведение. Движение двойного маятника можно моделировать с помощью системы обыкновенных дифференциальных уравнений. Однако, поскольку эти уравнения не имеют аналитического решения, они должны быть решены численно. Это можно сделать с помощью компьютера с соответствующим образом написанной программой.

**1.2 Численное моделирование хаотических колебаний**

Двойной маятник-это один маятник, подвешенный к другому как и показано на рисунке 1. Если предположить, что система не зажата и стержень не имеет массы, то можно вывести следующие уравнения движения [2]:

(1)

; (2)

; (3)

(4)

где - угол положения тела от вертикали;

- это угловой момент тела;

длина соединяющего стержня;

- масса тела, ∆ = −;

M = +;

g – ускорение свободного падения.

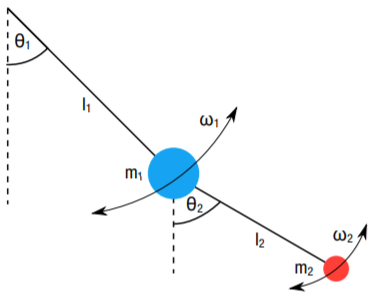


Рисунок 1 - Схема двойного маятника

Для моделирования двойного маятника необходимо решить эти уравнения при условии t. Поскольку аналитического решения не существует, оно должно быть выполнено численно. Существует несколько методов решения. Наиболее простым из них является метод Эйлера, который утверждает, что при заданной функции y (t) получается

, (5)

где

∆t - малый промежуток времени;

y'(t) - производной y(t) с учетом времени.

Поэтому, учитывая начальные условия запуска при y(0), можно вычислить y(t).

Более распространенным и продвинутым методом решения обыкновенных дифференциальных уравнений является метод Рунге-Кутты. В то время как метод Эйлера просто вычисляет производную один раз в y(t) метод Рунге-Кутты вычисляет производную несколько раз между y(t) и y(t +∆t), используя предыдущую производную в качестве исходной позиции и усредняют. Наиболее распространенным вариантом является метод Рунге-Кутты четвертого порядка, который вычисляет производную в общей сложности четыре раза для шага ∆t.

В лагранжевой механике для описания системы используются обобщенные координаты и обобщенные скорости. В нашем случае в качестве таких переменных взяты углы отклонения маятников ,  и угловые скорости  ,. Используя указанные переменные, записали нелинейную систему двух дифференциальных уравнений Лагранжа [3]:

+

(6)

(7)

С помощью преобразований Лежандра привели нелинейную систему ДУ Лагранжа к системе уравнений первого порядка Гамильтона. Для моделирования движения двойного маятника я воспользовался классическим методом Рунге-Кутты 4-го порядка точности. Предварительно несколько упростив дифференциальные уравнения Гамильтона, полагая, что длины маятников одинаковы: l1=l2=l и введя также μ, равный отношению массы второго маятника к массе первого: μ=. Тогда система уравнений принимает следующий вид:

, (6)

где

; (7)

. (8)

Данную систему перепишем в векторной форме:

(9)

где

(10)

. (11)

Вектор Z составлен из 4-х канонических переменных данной системы, а компоненты вектора f соответствуют правым частям дифференциальных уравнений.

Метод Рунге-Кутты предполагает на каждом шаге последовательное вычисление четырех промежуточных векторов:

(12)

(13)

(14)

(15)

Значение вектора Zn+1 в следующем временном узле вычисляется по формуле:

. (16)

Суммарная ошибка данного алгоритма на конечном интервале имеет порядок O(), т.е. точность вычислений возрастает в 16 раз при уменьшении временного шага τ в два раза.

**1.3 Язык программирования С++**

«C++» -- компилируемый статически типизированный язык программирования общего назначения. Поддерживая разные парадигмы программирования, сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником -- языком C, наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования. Название «C++» происходит от языка C, в котором унарный оператор ++ обозначает инкремент переменной [4].

Являясь одним из самых популярных языков программирования, C++ широко используется для разработки программного обеспечения. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (например, видеоигры). Существует несколько реализаций языка C++ -- как бесплатных, так и коммерческих. Их производят Проект GNU, Microsoft, Intel и Embarcadero (Borland). C++ оказал огромное влияние на другие языки программирования, в первую очередь на Java и C#.

При создании C++ Бьёрн Страуструп стремился сохранить совместимость с языком C. Множество программ, которые могут одинаково успешно транслироваться как компиляторами C, так и компиляторами C++, довольно велико -- отчасти благодаря тому, что синтаксис C++ был основан на синтаксисе C.

Стандарт C++ состоит из двух основных частей: описание ядра языка и описание стандартной библиотеки. Также существует огромное количество библиотек C++, не входящих в стандарт. В программах на C++ можно использовать многие библиотеки C [5].

Стандартизация определила язык программирования C++, однако за этим названием могут скрываться также неполные, ограниченные варианты языка. Первое время язык развивался вне формальных рамок, спонтанно, по мере встававших перед ним задач. Развитию языка сопутствовало развитие кросс-компилятора cfront. Новшества в языке отражались в изменении номера версии кросс-компилятора. Эти номера версий кросс-компилятора распространялись и на сам язык, но применительно к настоящему времени речь о версиях языка C++ не ведут.

**1.4 QT – кросс‑платформенный инструмент разработчика**

Отличительная особенность Qt от других библиотек — использование Meta Object Compiler (MOC) — предварительной системы обработки исходного кода. MOC позволяет во много раз увеличить мощь библиотек, вводя такие понятия, как cлоты и сигналы. Кроме того, это позволяет сделать код более лаконичным. Утилита MOC ищет в заголовочных файлах на c++ описания классов, содержащие макрос Q\_OBJECT, и создаёт дополнительный исходный файл на c++, содержащий метаобъектный код [6].

Qt позволяет создавать собственные плагины и размещать их непосредственно в панели визуального редактора. Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Qt комплектуется визуальной средой разработки графического интерфейса «Qt Designer», позволяющей создавать диалоги и формы в режиме WYSIWYG. в поставке Qt есть «Qt Linguist» — графическая утилита, позволяющая упростить локализацию и перевод программы на многие языки; и «Qt Assistant» — справочная система Qt, упрощающая работу с документацией по библиотеке, а также позволяющая создавать кросс-платформенную справку для разрабатываемого на основе t по. начиная с версии 4.5.0 в комплект Qt включена среда разработки «Qt Creator», которая включает в себя редактор кода, справку, графические средства «Qt Designer» и возможность отладки приложений. «Qt Creator» может использовать GCC или Microsoft VC++ в качестве компилятора и GDB в качестве отладчика. Для Windows версий библиотека комплектуется компилятором, заголовочными и объектными файлами MinGW.

Библиотека разделена на несколько модулей, для четвёртой версии библиотеки это [7]:

- QtCore — классы ядра библиотеки, используемые другими модулями;

- QtGui — компоненты графического интерфейса;

- QtNetwork — набор классов для сетевого программирования. Поддержка различных высокоуровневых протоколов может меняться от версии к версии.

- QtOpenGL — набор классов для работы с OpenGL;

- QtSvg — классы для отображения и работы с данными Scalable Vector Graphics (SVG);

- QtDesigner — классы создания расширений QtDesigner для своих

собственных виджетов;

- QtUiTools — классы для обработки в приложении форм Qt Designer;

- QtAssistant — справочная система;

- QtWebKit — модуль WebKit, интегрированный в Qt и доступный через её классы;

- QtCLucene — модуль для поддержки полнотекстового поиска, применяется в новой версии Assistant;

- ActiveQt — модуль для работы с ActiveX и COM технологиями для Qt-разработчиков под Windows.

В Qt имена классов выглядят как MyClassName, а имена методов – как myMethodName. При этом имена всех классов Qt начинаются с Q, например QObject, QList или QFont. Большинству классов соответствуют заголовочные файлы с тем же именем. Для эффективной работы с классами на стадии выполнения в Qt используется специальная объектная модель, расширяющая модель C++ [8]. В частности, добавляются следующие возможности:

- древовидные иерархии объектов;

- взаимодействие объектов через сигналы и слоты;

- свойства объектов.

Многие объекты определяются значением сразу нескольких свойств, внутренними состояниями и связями с другими объектами. Они представляют собой индивидуальные сущности, и для них не имеет смысла операция буквального копирования, а также разделение данных в памяти. В Qt эти объекты наследуют свойства QObject в тех случаях, когда объект требовалось бы рассматривать не как сущность, а как значение (например, при хранении в контейнере) – используются указатели. Иногда указатель на объект, наследуемый от QObject, называют просто объектом.

При создании пользовательских интерфейсов взаимодействие объектов часто осуществляется через обратные вызовы, т.е. передачу кода для последующего выполнения (в виде указателей на функции, функторов и т.д.).

Также популярна концепция событий и обработчиков, в которой обработчик действует как перехватчик события определенного объекта. В Qt вводится концепция сигналов и слотов. Сигнал отправляется при вызове метода.

Программисту при этом нужно только указать прототип метода в разделе signals. слот является методом, исполняемым при получении сигнала. Слоты могут объявляться в разделе pulic slots, protected slots или private slots. При этом уровень защиты влияет лишь на возможность вызова слотов в качестве обычных методов, но не на возможность подключения сигналов к слотам.

Модель сигналов и слотов отличается от модели событий и обработчиков тем, что слот может подключаться к любому числу сигналов, а сигнал может подключаться к любому числу слотов. При отправке сигнала будут вызваны все подключенные к нему слоты. Таким образом, для эффективной работы с классами на стадии выполнения Qt использует специальную объектную модель, в которой при помощи наследования от QObject и генерирования кода компилятором метаобъектов реализованы:

- иерархии объектов и система свойств;

- система сигналов и слотов;

- динамическая работа с классами

**2 Проектирование и разработка приложения**

**2.1 Визуальная составляющая приложения**

Приложение состоит из 2-х окон спроектированных с помощью QtDesigner. Когда пользователь открывает приложение, перед ним появляется главное окно, где большую часть пространства занимает окно с обучающим текстом, ниже которого находится кнопка как представлено на рисунке 2. При нажатии на эту кнопку запускается окно, представленное на рисунке 3 с настраиваемым моделированием хаотического движения двойного маятника.

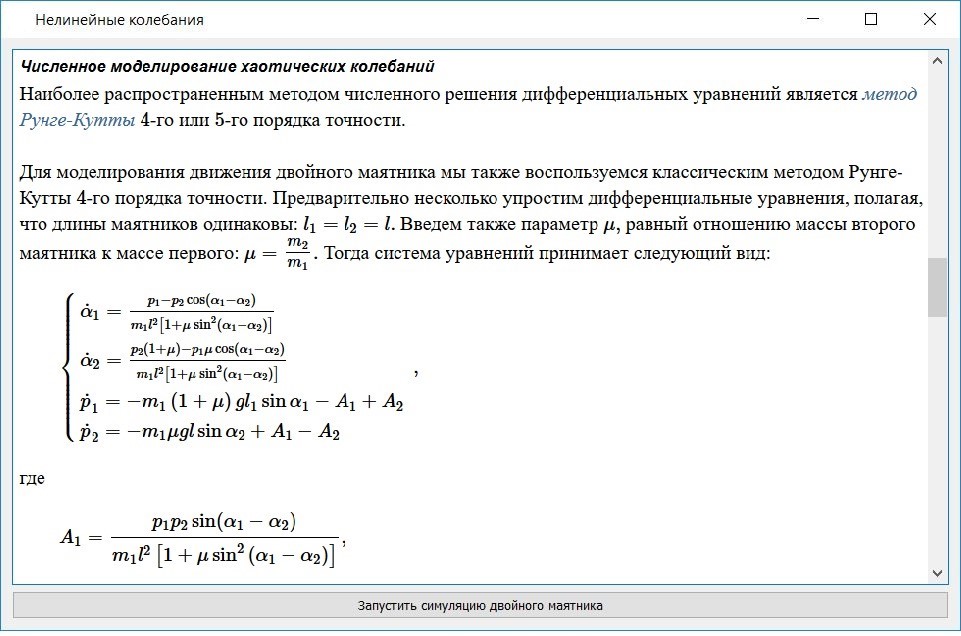


Рисунок 2 – Главное окно приложения

Графический интерфейс окна моделирования (Рисунок 3) создан с учетом представления произвольного числа маятников, каждый из которых имеет разные начальные условия запуска, чтобы облегчить визуальное сравнение. Скриншот сравнения маятников с различными условиями запуска представлен на рисунке 4.

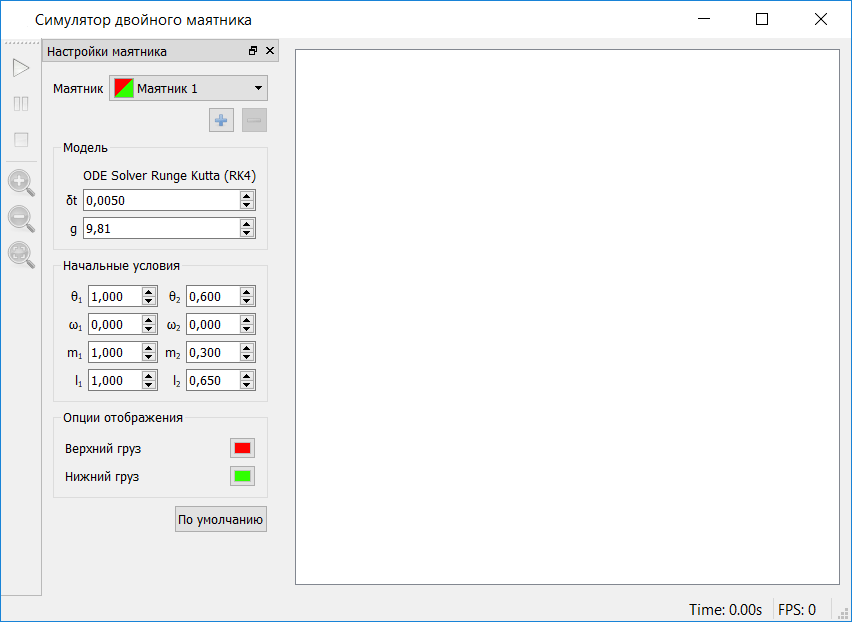


Рисунок 3 – Симулятор двойного маятника

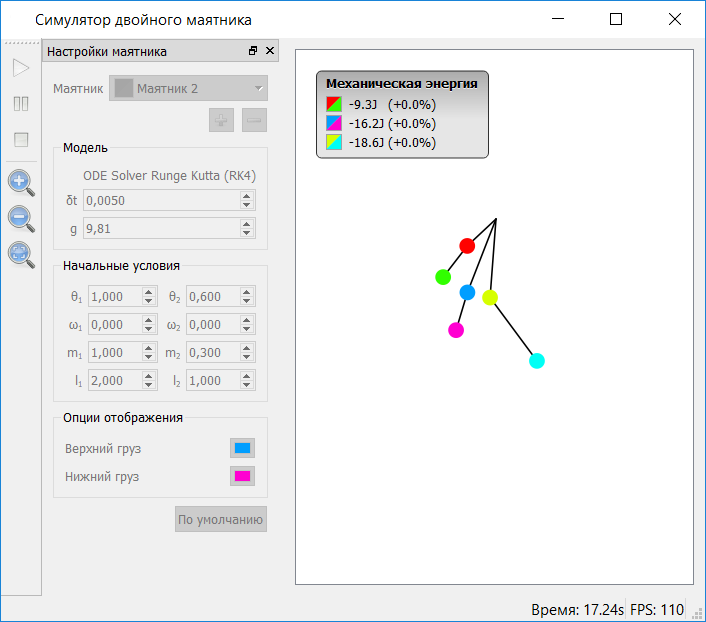


Рисунок 4 - Снимок экрана графического интерфейса, показывающий

три двойных маятника в случайное время качания

**2.2 Логическая составляющая приложения**

Поскольку сама программа насчитывает более 1000 строк кода c++, были прикреплены только основные классы, имитирующие двойной маятник (Листинг 1,2,3,4).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | 16 | | 17 | | 18 | | 19 | | 20 | | 21 | | 22 | | 23 | | 24 | | 25 | | 26 | | 27 | | 28 | | 29 | | 30 | | 31 | | 32 | | 33 | | 34 | | 35 | | 36 | | 37 | | 38 | | 39 | | 40 | | 41 | | 42 | | 43 | | 44 | | 45 | | 46 | | 47 | | 48 | | 49 | | 50 | | 51 | | 52 | | 53 | | 54 | | 55 | | 56 | | 57 | | 58 | | 59 | | 60 | | 61 | | 62 | | 63 | | 64 | | 65 | | 66 | | 67 | | 68 | | 69 | | 70 | | 71 | | 72 | | 73 | | 74 | | 75 | | 76 | | 77 | | 78 | | 79 | | 80 | | 81 | | 82 | | 83 | | 84 | | 85 | | 86 | | 87 | | 88 | | 89 | | 90 | | 91 | | 92 | | 93 | | 94 | | 95 | | 96 | | 97 | | 98 | | 99 | | 100 | | 101 | | 102 | | 103 | | 104 | | 105 | | 106 | | 107 | | 108 | | 109 | | 110 | | 111 | | 112 | | 113 | | 114 | | 115 | | 116 | | 117 | | 118 | | 119 | | 120 | | 121 | | 122 | | 123 | | 124 | | 125 | | 126 | | 127 | | 128 | | 129 | | 130 | | 131 | | 132 | | 133 | | 134 | | 135 | | 136 | | 137 | | 138 | | 139 | | 140 | | 141 | | 142 | | 143 | | 144 | | #ifndef DOUBLEPENDULUM\_H  #define DOUBLEPENDULUM\_H  // Моделирует состояние маятника  struct Pendulum  {  Pendulum()  : theta(0.0), omega(0.0), l(0.0), m(0.0)  {}  Pendulum(double \_theta, double \_omega, double \_l, double \_m)  : theta(\_theta), omega(\_omega), l(\_l), m(\_m)  {}  // Угол маятника от вертикали  double theta;    // Момент импульса маятника  double omega;  // длина ведущего стержня между маятником и осью  double l;  // Масса тела на конце маятника  double m;  };  class DoublePendulum  {  public:  DoublePendulum(const Pendulum& upper, const Pendulum& lower, double dt=0.005, double g=9.81);  virtual ~***DoublePendulum***();  /\* Продвигает уравнение по шагам m\_dt до обновлния  newTime. newTime -- время продвижения системы.  Всегда newTime >= time()\*/  void **update**(double newTime);  double **theta1**()  {  return m\_theta1;  }  double **omega1**()  {  return m\_omega1;  }  double **m1**()  {  return m\_m1;  }  double **l1**()  {  return m\_l1;  }  double **theta2**()  {  return m\_theta2;  }  double **omega2**()  {  return m\_omega2;  }  double **m2**()  {  return m\_m2;  }  double **l2**()  {  return m\_l2;  }  double **time**()  {  return m\_time;  }  double **initEnergy**() const  {  return m\_initEnergy;  }  double **energy**() const;    /\* Возвращает строковое представление  используемого метода решения. \*/  virtual const char \****solverMethod***() = 0;  protected:  /\* Список ДУ движения, который должен быть решен, чтобы численно оценить двойную маятниковую систему относительно времени. \*/  enum  {  THETA\_1,  OMEGA\_1,  THETA\_2,  OMEGA\_2,  NUM\_EQNS  };  /\* Метод вычисляет числовые производные theta и  Omega для верхнего и нижнего тела \*/  void **derivs**(const double \*yin, double \*dydx);  /\* Вызывается для решения уравнений движения системы путем продвижения theta и omega на 1 шаг. \*/  virtual void ***solveODEs***(const double \*yin, double \*yout) = 0;  //Угол первого маятника от вертикали (в рад).  double m\_theta1;  //Угловое ускорение первого маятника (dθ/dt).  double m\_omega1;  //Длина первого маятника (в м).  const double m\_l1;  //Масса первого маятника (в кг).  const double m\_m1;  //Угол второго маятника от вертикали (в рад).  double m\_theta2;  //Угловое ускорение второго маятника (dθ/dt).  double m\_omega2;  //Длина второго маятника (в м).  const double m\_l2;    //Масса второго маятника (в кг).  const double m\_m2;  //Размер шага при численном решении ДУ.  const double m\_dt;  //Ускорение силы тяжести (обычно 9.81 м\*с^2).  const double m\_g;  /\*Текущее время, для которого omega и theta  оцениваются.\*/  double m\_time;  //Начальная механическая энергия  double m\_initEnergy;  };  #endif // DOUBLEPENDULUM\_H |

Листинг 1 – Код файла doublependulum.h

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | 16 | | 17 | | 18 | | 19 | | 20 | | 21 | | 22 | | 23 | | 24 | | 25 | | 26 | | 27 | | 28 | | 29 | | 30 | | 31 | | 32 | | 33 | | 34 | | 35 | | 36 | | 37 | | 38 | | 39 | | 40 | | 41 | | 42 | | 43 | | 44 | | 45 | | 46 | | 47 | | 48 | | 49 | | 50 | | 51 | | 52 | | 53 | | 54 | | 55 | | 56 | | 57 | | 58 | | 59 | | 60 | | 61 | | 62 | | 63 | | 64 | | 65 | | 66 | | 67 | | 68 | | 69 | | 70 | | 71 | | 72 | | 73 | | 74 | | 75 | | 76 | | 77 | | 78 | | 79 | | 80 | | 81 | | 82 | | 83 | | 84 | | 85 |   86  87  88  89 | #include "doublependulum.h"  #include <cmath>  #include <cassert>  DoublePendulum::**DoublePendulum**(const Pendulum& upper, const Pendulum& lower, double dt, double g) :  m\_theta1(upper.theta),  m\_omega1(upper.omega),  m\_l1(upper.l), m\_m1(upper.m),  m\_theta2(lower.theta),  m\_omega2(lower.omega),  m\_l2(lower.l), m\_m2(lower.m),  m\_dt(dt), m\_g(g), m\_time(0.0),  m\_initEnergy(energy())  {}  DoublePendulum::~***DoublePendulum***()  {}  // Высчитывает энергию двойного маятника  double DoublePendulum::**energy**() const  {  double pe = -(m\_m1 + m\_m2) \* m\_g \* m\_l1 \* cos(m\_theta1)  - m\_m2 \* m\_g \* m\_l2 \* cos(m\_theta2);  double ke = 0.5 \* m\_m1 \* m\_l1\*m\_l1 \* m\_omega1\*m\_omega1 + 0.5 \* m\_m2  \* (m\_l1\*m\_l1 \* m\_omega1\*m\_omega1  + m\_l2\*m\_l2 \* m\_omega2\*m\_omega2  + 2 \* m\_l1 \* m\_l2 \* m\_omega1 \* m\_omega2 \* cos(m\_theta1 - m\_theta2));  return pe + ke;  }  void DoublePendulum::**update**(double newTime)  {  assert(newTime >= m\_time);  do  {  const double yin[NUM\_EQNS] = { m\_theta1, m\_omega1, m\_theta2, m\_omega2 };  double yout[NUM\_EQNS];  *solveODEs*(yin, yout);  m\_theta1 = yout[THETA\_1];  m\_omega1 = yout[OMEGA\_1];  m\_theta2 = yout[THETA\_2];  m\_omega2 = yout[OMEGA\_2];  } while ((m\_time += m\_dt) < newTime);  }  void DoublePendulum::**derivs**(const double \*yin, double \*dydx)  {  // delta = θ2 - θ1  const double delta = yin[THETA\_2] - yin[THETA\_1];  // 'M' - общая масса системы, m1 + m2;  const double M = m\_m1 + m\_m2;    // Знаменатель для выражения ω1  double den = M\*m\_l1 –  m\_m2\*m\_l1\*cos(delta)\*cos(delta);    // dθ/dt = ω, по определению  dydx[THETA\_1] = yin[OMEGA\_1];  // Вычисляет ω1  dydx[OMEGA\_1] = (m\_m2\*m\_l1\*yin[OMEGA\_1]\*yin[OMEGA\_1]\*sin(delta)\*  cos(delta) + m\_m2\*m\_g\*sin(yin[THETA\_2])\*cos(delta)+  m\_m2\*m\_l2\*yin[OMEGA\_2]\*yin[OMEGA\_2]\*sin(delta)-  M\*m\_g\*sin(yin[THETA\_1])) / den;  // Снова, dθ/dt = ω для θ2 так же будет  dydx[THETA\_2] = yin[OMEGA\_2];  /\* Умножение знаменателя для выражения ω1 (den) на отношение длины двух тел \*/  den \*= m\_l2 / m\_l1;  // Вычисляет ω2  dydx[OMEGA\_2] = (m\_m2\*m\_l2\*yin[OMEGA\_2]\*yin[OMEGA\_2]\*sin(delta)\*cos (delta)+M\*m\_g\*sin(yin[THETA\_1])\*cos(delta)  - M\*m\_l1\*yin[OMEGA\_1]\*yin[OMEGA\_1]\*sin(delta)  - M\*m\_g\*sin(yin[THETA\_2]))/den;} |

Листинг 2 – Код файла doublependulum.сpp

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | #ifndef DOUBLEPENDULUMRK4\_H  #define DOUBLEPENDULUMRK4\_H  #include "doublependulum.h"  class DoublePendulumRK4 : public DoublePendulum  {  public:  DoublePendulumRK4(const Pendulum& upper, const Pendulum& lower,  double dt=0.005, double g=9.81);  const char \****solverMethod***();  void ***solveODEs***(const double \*yin, double \*yout);  };#endif // DOUBLEPENDULUMRK4\_H |

Листинг 3 – Код файла doublependulumrk4.h

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | 16 | | 17 | | 18 | | 19 | | 20 | | 21 | | 22 | | 23 | | 24 | | 25 | | 26 | | 27 | | 28 | | 29 | | 30 | | 31 | | 32 | | 33 | | 34 | | 35 | | 36 | | 37 | | 38 | | 39 | | 40 | | 41 | | 42 | | 43 | | 44 | | 45 | | 46 | | 47 | | 48 | | 49 | | 50 | | 51 | | #include "doublependulumrk4.h"  DoublePendulumRK4::**DoublePendulumRK4**(const Pendulum& upper, const Pendulum& lower, double dt, double g) :  DoublePendulum(upper, lower, dt, g)  {  }  const char \*DoublePendulumRK4::***solverMethod***()  {  return "Runge Kutta (RK4)";  }  void DoublePendulumRK4::***solveODEs***(const double \*yin, double \*yout)  {  double dydx[NUM\_EQNS], dydxt[NUM\_EQNS], yt[NUM\_EQNS];  double k1[NUM\_EQNS], k2[NUM\_EQNS], k3[NUM\_EQNS], k4[NUM\_EQNS];  // Первый шаг  derivs(yin, dydx);  for (int i = 0; i < NUM\_EQNS; ++i)  {  k1[i] = m\_dt \* dydx[i];  yt[i] = yin[i] + 0.5 \* k1[i];  }  // Второй шаг  derivs(yt, dydxt);  for (int i = 0; i < NUM\_EQNS; ++i)  {  k2[i] = m\_dt \* dydxt[i];  yt[i] = yin[i] + 0.5 \* k2[i];  }  // Третий шаг  derivs(yt, dydxt);  for (int i = 0; i < NUM\_EQNS; ++i)  {  k3[i] = m\_dt \* dydxt[i];  yt[i] = yin[i] + k3[i];  }  // Четвортый шаг  derivs(yt, dydxt);  for (int i = 0; i < NUM\_EQNS; ++i)  {  k4[i] = m\_dt \* dydxt[i];  yout[i] = yin[i] + k1[i] /  6.0 + k2[i] / 3.0 + k3[i] /  3.0 + k4[i] / 6.0;  }} |

Листинг 4 – Код файла doublependulumrk4.cpp

**ЗАКЛЮЧЕНИЕ**

Цель курсового проекта состояла в том, чтобы создать модуль обучающего приложения на тему “Нелинейные колебания” моделирующий двойную маятниковую систему и ее хаотическое поведение. В качества инструмента был использован язык программирования С++ и фреймворк Qt вместе с IDE Qt Creator. В программе используется метод Рунге-Кутты четвертого порядка обеспечивающий достаточную точность при проведении расчетов.

При выполнении были решены следующие задачи:

1 Изучены основы поведения нелинейных колебаний.

2 Построена числовая модель двойного маятника.

3 Был создан образовательный модуль, визуализирующий хаотическое поведение двойного маятника при разных начальных условиях.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 Энциклопедии и словари [Электронный ресурс]; Нелинейные колебания. // – (Рус.). – URL: http://enc-dic.com/enc\_math/Nelinene-kolebanija-2491.html [2 января 2018].

2 Научная библиотека [Электронный ресурс]; Двойной маятник. // – (Рус.). – URL: http://lib.sernam.ru/book\_g\_tm.php?id=119 [15 января 2018].

3 Дифференциальные уравнения [Электронный ресурс]; Двойной маятник. // – (Рус.). – URL: http://www.math24.ru/двойной-маятник.html [3 февраля 2018].

4 Дейтел Х., Дейтел П. Как программировать на C++ / Х. Дейтел, П. Дейтел. – М.: Бином, 2017. – 1021 с.

5 Программирование на языке С++ в среде Qt Creator / Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк,О. В. Чеснокова, А. С. Чмыхало. – М.: ALT Linux, 2015. – 448 c.

6 Шлее М. Qt 5.3. Профессиональное программирование на C++ / М. Шлее. – СПб.: БХВ–Петербург, 2015. – 928 c.

7 CrossPlatform.RU Все о кроссплатформенном программировании [Электронный ресурс]; Основные классы Qt. // – (Рус.). – URL: http://doc.crossplatform.ru/qt/4.5.0/mainclasses.html [3 марта 2018].

8 Журнал физика-программиста [Электронный ресурс]; Моделирование физических процессов. // – (Рус.). – https://delirium-00.livejournal.com/954.html [10 марта2018].