МОДЕЛЬ ГРАФИЧЕСКОГО ПРЕДСТАВЛЕНИЯ ИНФОРМАЦИИ В ПРОГРАММНОМ КОМПЛЕКСЕ ОБРАБОТКИ ЭКСПЕРИМЕНТАЛЬНЫХ ДАННЫХ

MODEL OF GRAPHICAL REPRESENTATION OF INFORMATION IN THE SOFTWARE PACKAGE FOR PROCESSING EXPERIMENTAL DATA

Yu. Kostikov
V. Pavlov
A. Romanenkov
V. Ternovskov

Summary. In this paper, we consider a model of visual representation of the results of field experiments for constructing phenomenological mathematical models and for comparing the experimental data with the results of numerical simulation. The methods and approaches of mapping, saving, converting and interactive data modification are presented. Methods for accelerating the display of data are suggested in the case when the amount of data is large. As an original option, an internal scripting language is proposed, on which non-trivial data processing can be implemented, for example, obtaining values of a given functional dependence on the original data.

Keywords: mathematical modeling, graphical representation of data, WPF control element, level of firmware.

Костиков Юрий Александрович
К.ф.-м.н., Московский авиационный институт (национальный исследовательский университет)
jkostikov@mail.ru
Павлов Виталий Юрьевич
К.ф.-м.н., Московский авиационный институт (национальный исследовательский университет)
Романенков Александр Михайлович
К.т.н., доцент, Московский авиационный институт (национальный исследовательский университет)
romanaleks@gmail.com
Терновсков Владимир Борисович
К.т.н., доцент, Московский авиационный институт (национальный исследовательский университет)
vternik@mail.ru

Аннотация. В данной работе рассматривается модель визуального представления данных натурных экспериментов для построения феноменологических математических моделей и для сопоставления экспериментальных данных с результатами численного моделирования. Представлены методики и подходы отображения, сохранения, конвертации и интерактивной модификации данных. Предложены методы ускорения отображения данных в том случае, когда объем данных велик. В качестве оригинальной опции предложен внутренний скриптовый язык, на котором можно реализовать нетривиальную обработку данных, например, получение значений заданной функциональной зависимости от исходных данных.

Ключевые слова: математическое моделирование, графическое представление данных, элемент управления WPF, уровень микропрограмм.

Введение

При построении феноменологических моделей на основе анализа экспериментальных данных и при сопоставлении результатов математического моделирования с результатами натурных экспериментов почти всегда возникает задача наглядного отображения данных в виде, удобном для анализа и сопоставления результатов. При этом тривиальное отображение функциональной зависимости или построение диаграммы не всегда является конечной целью отображения данных. Возникают потребности в отображении не единичного графика, а, например, серии графиков. Бывает также необходимо выполнить некоторую работу с выведенными данными, например, отбросить явно некорректные точки или подправить некоторые точки графика, скрыть какую-либо его часть, выполнить сглаживание или усреднение по некоторому множеству точек или набору параметров, построить необходимую интерполяцию или аппроксимацию. С задачами такого рода элементы управления, которые предназначены для отображения графиков, не справляются. Резонным оправданием в данном случае является следование принципу разделения областей функциональной ответственности (области ответственности компонентов или элементов управления не должны пересекаться). При использовании стандартных готовых решений возникает дополнительная проблема выполнения вспомогательной конвертации презентуемых данных для того, чтобы воспользоваться стандартным элементом управления. Но этот подход часто является неприемлемым, так как процесс конвертации, помимо того, что он может являться долгим, также может являться вообще не осуществимым если, например, массив данных представляет собой не одну функцию, а серии функции, таких что области их задания пересекаются и для некоторых

значении аргументов существует более одного значения измерения. В данной работе предлагаются решения описанных проблем.

В различных программных продуктах по обработке данных у пользователей появляется необходимость применять собственные методы математической обработки данных и неоднократно повторять какую-либо последовательность действий в процессе работы, например, проводить комплексную настройку отображения результатов. Бывает также необходимо, имея перед глазами графическое представление данных, выполнить их модификацию: передвинуть некоторые точки, деформировать график или его часть, повернуть график, или задать новую функциональную зависимость на основе уже имеющихся данных. Такие опции особенно полезны при построении феноменологических моделей, если при модификации графического представления автоматически изменяются константы и коэффициенты в уравнениях, аппроксимирующих экспериментальные данные.

Как уже было отмечено, в общем случае такой функционал элемент управления для построения графиков предоставлять не должен, и приходится выполнять лишнюю работу: в стороннем приложении готовить новые данные для отображения, выполнять их конвертацию и затем импортировать обратно в приложение. Такие операции, кроме больших временных затрат, требуют достаточно высокой квалификации пользователей, что несомненно влечет дополнительные расходы на их обучение.

Решением, которое предлагается в данной работе, является разработка внутреннего языка элемента управления. А именно, разработка и внедрение системы микропрограмм, включающей описание грамматики языка (Grammar), синтаксический анализатор (Parser), компилятор (Assembler), интегрированную среду разработки (IDE). Предусматриваются различные варианты сохранения программного алгоритма действий: непосредственное написание кода (гибкий) и взаимодействие с предоставляемым интерфейсом (упрощенный). Фактически элемент управления позволяет создавать скрипты, результатом работы которых является модификация исходных данных, создание новых производных данных. Удобство такого подхода состоит в том, что фактическая работа ведется над исходными данными, нет необходимости выполнять какую-либо дополнительную подготовку данных или обработку их в сторонних приложениях.

Графический модуль

Данный модуль (Graphing) предназначен для графического представление и визуального анализа данных. Иными словами, область ответственности модуля — это построение графиков и визуальная демонстрация данных эксперимента. При этом поддерживается возможность графического, в интерактивном режиме, редактирования копии результатов эксперимента. Предоставляется опция ручного корректирования данных.

В модуле реализован функционал вычисления разницы между двумя выбранными графиками. Часто возникает задача не только грубо, на глаз, оценить какие экспериментальные значения больше либо меньше, но и знать точное отклонение одних объектов от других. Для решения этой задачи разработана возможность вычисления разности между ординатами выбранных графиков. А также предоставляется программная возможность вычисления нормы в пространстве $L_p(\Omega)$, где Ω — множество аргументов:

$$\|f\|_p = \left(\int_\Omega |f(x)|^p dx \right)^{\frac{1}{p}}$$

Модуль реализован таким образом, что присутствует возможность вычислить значение $\|f\|_p$ для нескольких значений p и визуально оценить полученные результаты. На рисунке 1 представлена структура модуля.

Кратко опишем его структуру. Класс RendererBase предназначен для инициализации начальных настроек для последующей отрисовки.

Под начальной настройкой подразумевается инициализация базовых OpenGL библиотек и установления необходимых связей. Стоит отметить, что выбор работы на столь низком уровне (уровень работы с видеокартой) обусловлен тем, что модуль на программном уровень реализует ускоренное отображение данных, что позволяет избежать «подвисания» при обработке больших объемов данных и обеспечивает плавную работу графического модуля.

Класс GridRenderer отвечает за отрисовки и масштабирование сетки; PlotRenderer — за отрисовку и масштабирование графических объектов в области вывода; GraphControl — элемент управления, отвечающий за отображение набора графиков; GraphSettings — класс, определяющий пользовательские настройки графика, такие как параметры пера, параметры заливки и прозрачность; Scaler — класс, отвечающий за изменение масштаба изображения; Series — содержит набор точек, настройки графика и вспомогательный функционал для работы над точками.

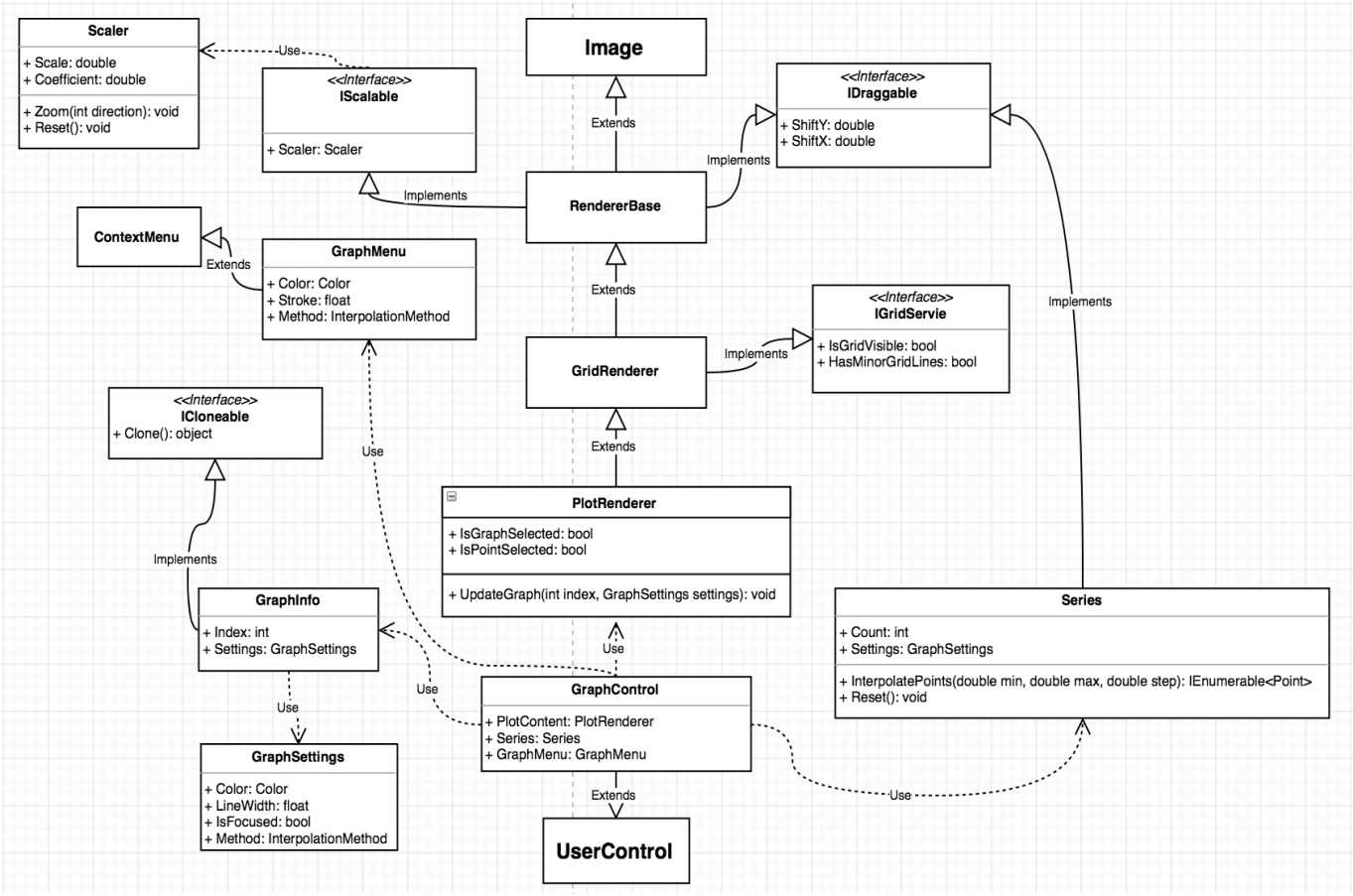

Рис. 1.Структура модуля Graphing

Помимо визуальной компоненты модуля реализован компонент Math. Данный компонент отвечает за реализацию математических методов и алгоритмов. Нередко при отображении данных возникает проблема изломов, разрывов, что может быть обусловлено не только малым количеством точек, но и природой данных. Для устранения такого рода нежелательных эффектов предлагаются различные методы интерполяции и аппроксимации, которые позволяют по заданному набору данных получить, при необходимости, промежуточные значения, а также позволяет придать линиям графика гладкость и плавность.

Интерполяционная часть модуля реализована таким образом, что позволяет в случае необходимости без особого труда добавлять новые методы интерполяции. Такая гибкость достигается благодаря организации полиморфного поведения конкретных реализации методов интерполяции. Перечислим основные методы, которые реализованы на данный момент: метод наименьших квадратов, метод Ньютона, метод Лагранжа, метод Якоби, метод Лаггера, метод Лежандра, метод Эрмита, метод Чебышёва, метод сплайновой интерполяции, метод Тригонометрической интерполяции [5, 6, 7]. На рисунке 2 представлена структура модуля Math.

Каждый алгоритм реализован в виде отдельного класса, который наследуется от базового класса InterpolationAlgorithmBase.

Базовый класс получает набор точек для дальнейшего построения, а также описывает два виртуальных метода SetupInterpolation и InterpolateValue, которые переопределяются в каждом из производных классов. Метод SetupInterpolation отвечает за построение интерполянта, то есть в нём реализована логика нахождения коэффициентов полинома, с помощью которых мы можем с некоторой погрешностью восстановить значения исходной функции в заданных точках. Одним из достоинств построенной архитектуры является независимость от конкретной реализации, что позволяет использовать реализацию методов интерполяции на любом.net совместимом языке программирования [3, 4].

Обратимся теперь к модулю микропрограмм. Данный модуль является неотъемлемой частью графическо-

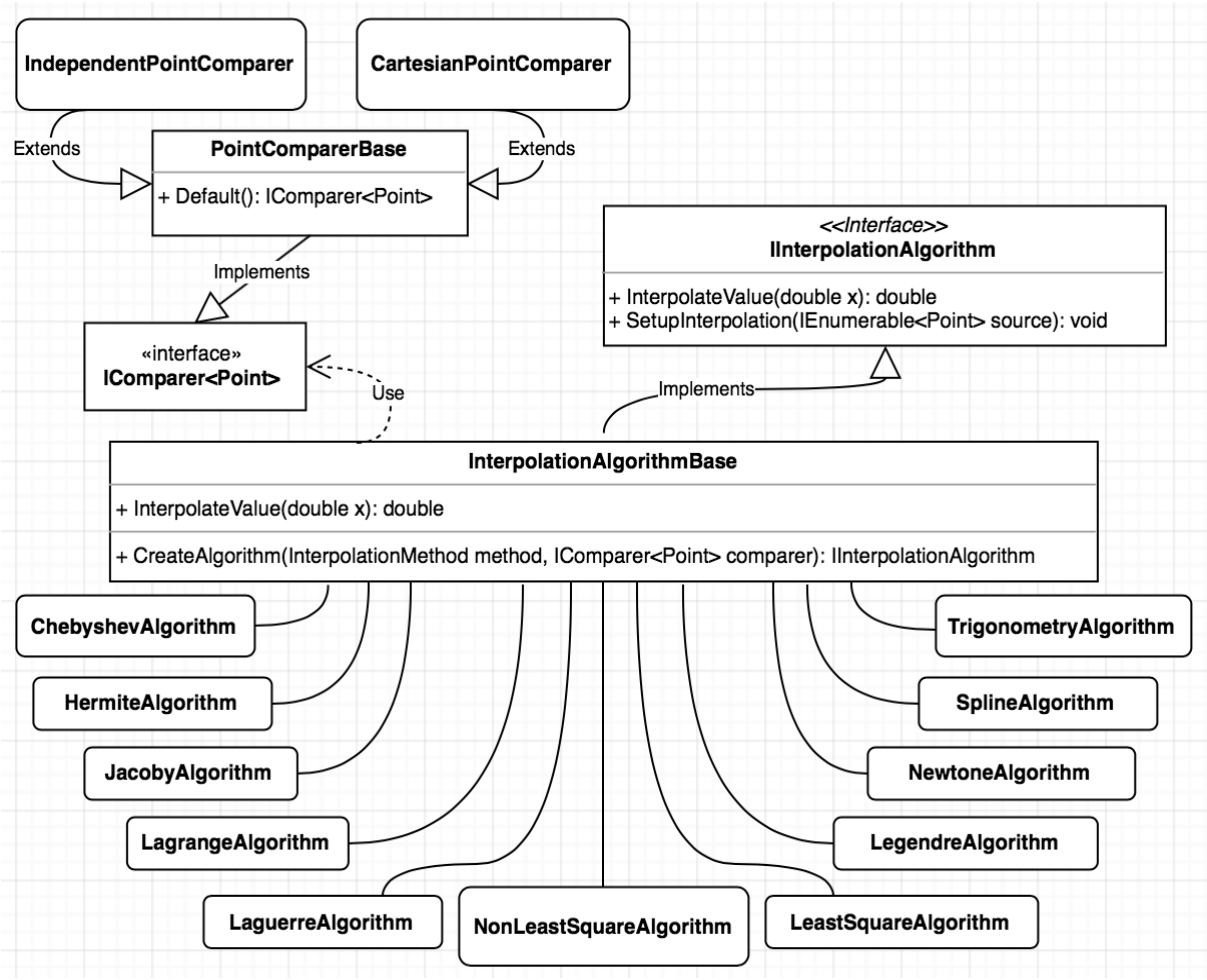

Рис. 2. Структура модуля Math

го модуля, но также допускает независимую реализацию и модификацию. Детально опишем его структуру.

Компонент Grammar

Данный компонент предназначен для определения алфавита, лексического состава, набора правил и конструкций языка. Для этого необходимо задать парадигму программирования и систему типов языка.

Как правило, среднестатистический пользователь программного продукта не имеет навыков написания кода, следовательно, все возможные ошибки требуется находить на этапе компиляции и формировать их подробное описание, что приводит к необходимости строгой типизации. Для программного осуществления действий, предписываемых графическим интерфейсом, и упрощения реализации специфичных методов обработки данных наиболее простым вариантом является парадигма процедурного программирование.

В алфавит языка будут включаться все русские и латинские буквы, цифры, пробельные символы, специальные лексемы конструкций языка. Для упрощения написания микропрограмм код должен анализироваться без учета регистра букв и табуляции.

При разработке лексического состава языка необходимо определить формат записи комментариев, различных типов неименованных констант, таких как строчные, целочисленные и вещественные; назначить знаки для всех типов операций и служебные слова, участвующие в построении конструкций.

Строго определенная структура микропрограмм облегчает синтаксический разбор в процессе компиляции и понимание кода пользователем. Возможный вариант:

1. Необязательный блок объявления глобальных переменных
2. Необязательный блок объявления функций
3. Обязательный блок инструкций микропрограммы

Структура функции:

4. Необязательный блок объявления локальных переменных
5. Обязательный блок инструкций функции.

Компонент Parser

Данный компонент предназначен для построения на основе грамматики языка объектной модели абстрактного синтаксического дерева пользовательской микропрограммы, где корень — точка входа, внутренняя вершина — инструкция, лист — операнд. Используется непосредственно в редакторе для анализа кода в процессе его непосредственного написания пользователем для уведомления в случае некорректных конструкций и при компиляции решения.

В качестве средства разработки была выбрана библиотека Irony, которая поставляется через пакетный менеджер Nuget и имеет лицензию MIT. Irony представляет собой готовое решение для описания грамматики и создания абстрактного синтаксического дерева.

Assembler

Компонент предназначен для компиляции пользовательской микропрограммы в динамически подключаемую библиотеку.

Первым шагом производится семантический разбор кода. В процессе разбора абстрактное синтаксическое дерево переводится в дерево выражений. Выделяются коллекции для хранения списка объявленных переменных и их значений, списка пользовательских функций. На этом этапе происходит поиск ошибок, связанных с некорректным использованием синтаксически верных конструкций, например, использование не объявленных переменных, выполнение операций над данными, неприводимых друг к другу типов. В случае вызова функции, не объявленной в соответствующем блоке, она ищется в списке стандартных, реализованных непосредственно при разработке языка.

После получения готовой к выполнению конструкции необходимо записать её в динамически подключаемую библиотеку. Так как разработка ведется в объектно-ориентированном стиле, файл будет хранить класс (Script), содержащий метод (Run) вызова лямбда-выражения.

В качестве средства построения дерева выражения используется стандартное API платформы, определённое в пространстве System.Linq.Expressions. Оно предоставляет механизмы динамического компилирования и выполнения лямбда-выражений. Формирование библиотеки производится средствами пространства System.Reflection.Emit.

IDE

Пользователям необходимо предоставить интегрированную среду разработки микропрограмм. Она должна включать:

- интерфейс для открытия и создания исходных файлов, запуска компиляции
- систему уведомлений для отображения статуса сборки микропрограммы и подробного описания ошибок, обнаруженных в процессе синтаксического анализа и при формировании дерева выражений
- механизм синтаксической подсветки, выделяющие ключевые слова языка и некорректные конструкции

В качестве средства разработки была выбрана стандартная библиотека Windows Presentation Foundation (WPF) [1, 2].

WPF представляет собой обширный API-интерфейс для создания настольных графических программ, имеющих насыщенный дизайн и интерактивность. В рамках технологии WPF возможна быстрая и гибкая настройка пользовательского интерфейса под нужды пользователя.

Основные возможности:

- возможность декларативного определения графического интерфейса с помощью специального языка разметки XAML, что позволяет отделять функциональную логику от пользовательского интерфейса.
- использование возможностей привязки и шаблонов данных позволяет полностью отделить бизнес-логику приложения от пользовательского интерфейса.
- такие механизмы, как стили и ресурсы, позволяют стандартизировать форматирование и многократно использовать его по всему приложению, а также дают возможность изменить способ отображения элементов.
- аппаратное ускорение графики — все компоненты пользовательского интерфейса визуализируются с помощью видеокарты, что повышает производительность приложения.
- независимость от разрешения экрана: поскольку в WPF размеры всех элементов вычисляются в независимых от устройства единицах, приложения на WPF легко масштабируются под экраны с разным разрешением.

В качестве демонстрации приведем примеры отображения серии экспериментальных данных (Рисунок 3) и код, который может быть исполнен на уровне микро-

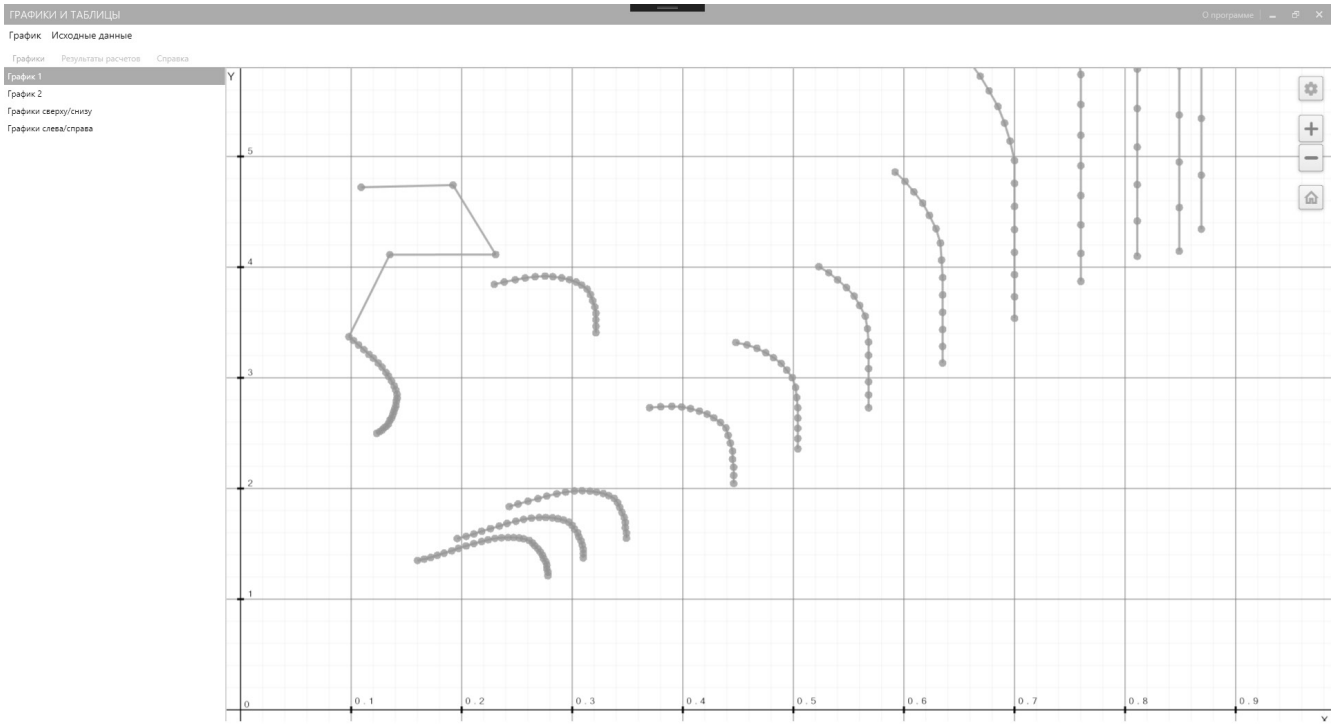

Рис. 3. Серия данных

```
var
    matrix: array[5, 5] of real;
    i, j: integer;

begin
    for i:=0 to 4 do
        for j:=0 to 4 do
            matrix[i,j]:=log(j, 10);
end
```

Рис. 4. Заполнение двумерного массива

программ данного модуля (в данном случае — это заполнение двумерного массива). Как видно, показаны серии подобных данных, так же показана возможность выбирать и модифицировать объекты, которые находятся в области вывода.

Заключение

Результатом данной работы является графический модуль, который реализует модель визуального отображения числовых данных с возможностями гибкой настройки параметров отображения и интерактивный режим управления отображением. Модуль поддерживает разные режимы отображения и позволяет выполнять как элементарную обработку эмпирических данных, так и весьма нетривиальную, например, вычислять различные виды отклонений, выполнять интерполяцию с помощью специальных полиномов.

Помимо обработки данных на уровне отображений, реализована возможность обработки данных на лету. А именно, в работе продемонстрирована стратегия разработки языка программирования для решения задач по обработке данных. Перечислены необходимые компоненты и средства их реализации. Указаны существенные аспекты, на которые нужно обращать внимание при разработке такого рода программных решений.

В процессе разработки был решен ряд задач, напрямую оказывающих влияние на производительность, качество представления объектов. Данный модуль реализован на языке C# с применением технологии построения пользовательских интерфейсов — WPF и библиотеки OpenGL. Модуль найдёт применение в системах для обработки данных натурных экспериментов, для построения и коррекции феноменологических математических моделей, для сопоставления результатов математического моделирования с экспериментальными данными.

ЛИТЕРАТУРА

1. М. Макдональд. WPF: Windows Presentation Foundation в.NET 4.5 с примерами на C# 5.0 для профессионалов. Изд. Вилльямс. 2015
2. Эндрю Троелсон. Язык программирования C# 5.0 и платформа.NET 4.5. Изд. Вилльямс. 2015
3. Стив Макконнелл. Совершенный код. Русская Редакция, Питер.2007
4. Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования. Изд. Питер. 2016
5. А. Самарский, А. Гулин. Численные методы. Изд. Главная редакция физико-математической литературы издательства «Наука». 1989
6. В. С. Рябенький. Введение в вычислительную математику. Изд. ФИЗМАТЛИТ. 2008
7. П. Суетин. Классические ортогональные многочлены. Изд. Главная редакция физико-математической литературы издательства «Наука». 1979
8. Демидов Л. Н., Терновсков В. Б., Григорьев С. М., Крахмалев Д. В. Информационные технологии. Кнорус Учебник\ Москва, 2017.
9. Терновсков В. Б., Данилина М. В., Литвинов А. Н., Кулакова Е. Ю. Роль синергетической концепции образования. Сборник статей победителей IV Международной научно-практической конференции. 2016. С. 69–72.
10. Демидов Л. Н., Костиков Ю. А., Павлов В. Ю., Терновсков В. Б., Современные информационные технологии. Издательство МИРТ Учебник\ Москва, 2017.

© Костиков Юрий Александрович (jkostikov@mail.ru), Павлов Виталий Юрьевич,
Романенков Александр Михайлович (romanaleks@gmail.com), Терновсков Владимир Борисович (vternik@mail.ru).
Журнал «Современная наука: актуальные проблемы теории и практики»

Московский авиационный институт