CRANFIELD UNIVERSITY

Mateusz Golab

Forward integration of simultaneous ordinary differential equations with graphical output

School of Engineering

Software Engineering for Technical Computing

MSc THESIS

Academic Year: 2011 - 2012

Supervisor: Dr Peter Sherar, Prof Joanna Polanska

August 2012

CRANFIELD UNIVERSITY

School of Engineering

Software Engineering for Technical Computing

MSc THESIS

Academic Year 2010 - 2011

Mateusz Golab

Forward integration of simultaneous ordinary differential equations with graphical output

Supervisor: Dr Peter Sherar, Prof Joanna Polanska

August 2012

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science

This thesis is submitted in accordance with the Double Degree programme regulations. Home institution : Silesian University of Technology, Poland

© Cranfield University 2012. All rights reserved. No part of this publication may be reproduced without the written permission of the copyright owner.

ABSTRACT

Click here to enter abstract text

Keywords:

ODE, Runge-Kutta, Modified Midpoint, Predictor - Corrector, Web development, GWT, AppEngine, Datastore, Unit testing.

ACKNOWLEDGEMENTS

Click here to enter acknowledgement text

TABLE OF CONTENTS

[ABSTRACT i](#_Toc331890279)

[ACKNOWLEDGEMENTS ii](#_Toc331890280)

[LIST OF FIGURES v](#_Toc331890281)

[LIST OF TABLES vi](#_Toc331890282)

[LIST OF EQUATIONS vii](#_Toc331890283)

[LIST OF ABBREVIATIONS viii](#_Toc331890284)

[1 Introduction 1](#_Toc331890285)

[1.1 Ordinary differential equations 1](#_Toc331890286)

[1.2 Euler’s method 2](#_Toc331890287)

[1.3 Aims and objectives 2](#_Toc331890288)

[1.4 Motivation 2](#_Toc331890289)

[2 Literature review 3](#_Toc331890290)

[2.1 ODE numerical routines 3](#_Toc331890291)

[2.1.1 Runge-Kutta methods 3](#_Toc331890292)

[2.1.2 Modified midpoint 5](#_Toc331890293)

[2.1.3 Richardson extrapolation 6](#_Toc331890294)

[2.1.4 Rosenbrock 8](#_Toc331890295)

[2.1.5 Predictor- Corrector 8](#_Toc331890296)

[2.2 Technologies 10](#_Toc331890297)

[2.2.1 AJAX approach 10](#_Toc331890298)

[2.2.2 Google Web Toolkit 11](#_Toc331890299)

[2.2.3 AppEngine 12](#_Toc331890300)

[2.2.4 Datastore 13](#_Toc331890301)

[3 Methodologies chosen and Application design 16](#_Toc331890302)

[3.1 The Prototyping Model 16](#_Toc331890303)

[3.2 Test Driven Development 17](#_Toc331890304)

[3.3 Architecture 17](#_Toc331890305)

[3.4 Design patterns 17](#_Toc331890306)

[3.5 Technologies applied 17](#_Toc331890307)

[4 Testing 18](#_Toc331890308)

[4.1 Test driven development approach 18](#_Toc331890309)

[4.2 Testing methods 18](#_Toc331890310)

[4.2.1 Unit testing 18](#_Toc331890311)

[4.2.2 Integration testing 18](#_Toc331890312)

[4.2.3 System testing 18](#_Toc331890313)

[4.2.4 Cross – browser testing 18](#_Toc331890314)

[5 Implementation 19](#_Toc331890315)

[5.1 Parser 19](#_Toc331890316)

[5.2 Solver 19](#_Toc331890317)

[5.3 Graph viewer 19](#_Toc331890318)

[5.4 Datastore connector 19](#_Toc331890319)

[6 Results 20](#_Toc331890320)

[6.1 Results validation and verification 20](#_Toc331890321)

[6.2 Application’s outputs 20](#_Toc331890322)

[7 Discussion and conclusion 21](#_Toc331890323)

[7.1 Solvers correctness 21](#_Toc331890324)

[7.2 Limitations 21](#_Toc331890325)

[7.3 Problems faced 21](#_Toc331890326)

[7.4 Quality of implementation 21](#_Toc331890327)

[7.5 Conclusion 21](#_Toc331890328)

[7.6 Future work 21](#_Toc331890329)

[REFERENCES 24](#_Toc331890330)

[APPENDICES 26](#_Toc331890331)

# LIST OF FIGURES

[Figure 2-1 4th Order Runge-Kutta method 4](#_Toc331890332)

[Figure 2-2 Richardson extrapolation used in the Burlisch-Stoer method with substep n = 2,4,6 7](#_Toc331890333)

[Figure 2-3 Aitkens-Neville polynomial extrapolation tableau 7](#_Toc331890334)

[Figure 2-4 Comparison of AJAX light traffic needs vs legacy HTML applications 11](#_Toc331890335)

LIST OF TABLES

**Nie można odnaleźć pozycji dla spisu ilustracji.**

Table 2-1 Data types available in Datastore

LIST OF EQUATIONS

[(1‑1) 1](#_Toc331890336)

[(2‑1) 3](#_Toc331890337)

[(2‑2) 4](#_Toc331890338)

LIST OF ABBREVIATIONS

|  |  |
| --- | --- |
| API  CU  GAE  GWT  JDO  JPA  JVM  ODE  PaaS | Application Programming Interface  Cranfield University  Google App Engine  Google Web Toolkit  Java Data Objects  Java Persistence API  Java Virtual Machine  Ordinary differential equation  Platform as a Service |
| RPC  SDK  SWT  UI | Remote Procedure Call  Software Development Kit  Standard Widget Toolkit  User Interface |
|  |  |
|  |  |
|  |  |

# Introduction

The Introduction chapter points aims and objectives of the thesis project. All requirements of the project are explained in this chapter. Moreover, main motivators to perform this specific topic are included as well.

## Ordinary differential equations

General definition of the ordinary differential equation of order :

|  |  |
| --- | --- |
|  | (‑) |

Important issues involving ODEs are boundary conditions. In other words algebraic conditions on the values of the function . In general they divide into two broad categories :

* Initial value problems

All are given at some starting point and it is desired to find the at some final point or list of points with specified intervals, for example .

* Two-point boundary value problems

Where boundary conditions are specified at more than one point. For example at starting and final point.

General definition of the system of ODEs of order :

|  |  |
| --- | --- |
|  | (‑2) |

|  |  |
| --- | --- |
|  | (‑3) |

## Euler’s method

Euler’s method is a first-order numerical method for solving ODEs . It is not recommended method for practical use, however it is important conceptually for advanced methods.

|  |  |
| --- | --- |
|  | (‑4) |

|  |  |
| --- | --- |
|  | (‑5) |

## Aims and objectives

The main objective of this thesis project is to develop following application :

1. Parsing equations entered by the user
2. Solving series of 1st order simultaneous ordinary differential equations (linear or non-linear) for initial value problem.
3. Presenting the solution on the 2D graph.
4. Storing and loading equations along with parameters entered by the user.

## Motivation

In general, two main motivators of this project are :

1. Discovering most efficient numerical methods for solving series of ordinary differential equations.
2. Familiarise with AJAX applications development and Google AppEngine along with Datastore.

# Literature review

This chapter demonstrates knowledge about recommended practical numerical methods for solving ODEs. Presented literature review gives an insight into major types of numerical routines taking into account efficiency and accuracy. What is more, technologies supporting AJAX applications development along with App Engine are also presented in this chapter. It is important to perform detailed research about technologies essential to develop the project.

## ODE numerical routines

Describes types of practical numerical methods for solving ODEs including Runge-Kutta methods, Bulirsch-Stoer, Rosenbrock metods and predictor-corrector rmethods.

### Runge-Kutta methods

Runge-Kutta methods propagate a solution over an interval by combining the data from several Euler-style steps. Each step involves one evaluation of the right-hand of the function .

|  |  |
| --- | --- |
|  | (2‑) |

Thenusing the information obtained to match a Taylor series expansion up to higher order.

Developing higher – order methods made Runge-Kutta competitive with the other numerical methods in many cases. It is usually the fastest method when moderate accuracy is required (≤ ) and evaluation of the function is not too expensive. There are few kinds of Runge-Kutta methods : 2nd order method, (called midpoint method), 4th order method and also method with adaptive stepsize. [numerical recipes]

#### 4th Order Runge-Kutta

The most often used Runge-Kutta method is fourth – order formula . In general it is superior to 2nd order method, however high order does not always mean high accuracy. 4th order method requires four evaluations of the function. [numerical recipes]

|  |  |
| --- | --- |
|  | (‑) |

|  |  |
| --- | --- |
|  | (2‑3) |

|  |  |
| --- | --- |
|  | (2‑4) |

|  |  |
| --- | --- |
|  | (2‑5) |

|  |  |
| --- | --- |
|  | (2‑6) |



Figure 2- 4th Order Runge-Kutta method

During single step, the derivative is evaluated four times. Once at initial point, (1) twice at midpoints (2)(3) and once at trial endpoint (4). Final function value is calculated on the basis of these derivatives. [numerical recipes]

Each step in the sequence of steps is treated in an identical manner, so prior behaviour of the solution is not used in its propagation. Such approach is mathematically proper, since any point along the trajectory of an ODE can be an initial point. [numerical recipes]

#### Runge-Kutta with adaptive stepsize

The purpose of adaptive stepsize method is to achieve predetermined accuracy in the solution with minimum computational effort. It is possible to face very smooth interval , while performing Runge-Kutta steps. Few great strides instead of small steps should speed through such undifferentiated interval, what may result in significant gain in efficiency. The idea of adaptive stepsize method is to control the size of the step and increase it when possible maintaining required level of accuracy. It is important to estimate truncation error to control accuracy level while increasing step size. Obviously the calculation of this information will add to the computational overhead, however it is profitable investment in terms of efficiency.

### Modified midpoint

Modified midpoint method is a second order method like 2nd order Runge-Kutta, however with the advantage of requiring only one derivative evaluation per single step instead of two evaluations present in Runge-Kutta. This method generates the solution as a vector of values from a point  to a point by a sequence of substeps each of size . Where

|  |  |
| --- | --- |
|  | (2‑7) |

The total number of function evaluations required by this method is . The formulas essential to provide the solution for are as follows

|  |  |
| --- | --- |
|  | (2‑8) |

|  |  |
| --- | --- |
|  | (2‑9) |

|  |  |
| --- | --- |
|  | (2‑10) |

|  |  |
| --- | --- |
|  | (2‑11) |

The is the final approximation to whereas represents intermediate approximations calculated along in steps of .

### Richardson extrapolation

Richardson extrapolation bases on idea of extrapolating a computed value to the value that would have been obtained if the stepsize had been remarkably smaller than it actually was. The practical numerical method using this idea is called Bulirsch-Stoer method.

#### Burlisch – Stoer

The idea of Burlisch-Stoer method is to perform iterations of modified midpoint method . Each iteration uses various number of substepsfor modified midpoint method ends up with polynomial extrapolation of the given values. Burlish and Stoer originally proposed following sequence of substeps :

|  |  |
| --- | --- |
|  | (2‑12) |

However sequence discovered by Deuflhard is usually more efficient :

|  |  |
| --- | --- |
|  | (2‑13) |

In terms of number of iterations usually 8 gives satisfactory results.



Figure 2- Richardson extrapolation used in the Burlisch-Stoer method with substep n = 2,4,6

We use Aitkens-Neville algorithm in order to perform extrapolation, which is described by the following tableau :



Figure 2- Aitkens-Neville polynomial extrapolation tableau

The first column of the tableau is formed by modified midpoint first iteration with n = 2 .

|  |  |
| --- | --- |
|  | (2‑14) |

Where is computed with the stepsize

Successive columns can be filled by using following recurrence :

|  |  |
| --- | --- |
|  | (2‑15) |

The final solution which is can be achieved after performing each successive iteration.

#### Stepsize Control Algorithm for Bulirsch-Stoer

### Rosenbrock

Rosenbrock methods are competitive with other numerical ODEs integrators in terms of moderate accuracies (tolerances of order . Moreover these methods remain reliable for more stringent parameters . The formula of Rosenbrock method is as follows :

|  |  |
| --- | --- |
|  | (2‑16) |

Where corrections are found after solving following linear equations :

|  |  |
| --- | --- |
|  | (2‑17) |

The coefficients are fixed and Jacobian matrix is denoted by . [ numerical recipes]

### Predictor- Corrector

Predictor – Corrector methods are a subcategory of methods called “multistep” and “multivalue”. These methods have had long historical run. It is said that that predictor-corrector integrators have had their day. For high precision applications and right-hand side expensive evaluations Bulirsch-Stoer method dominates. For moderate precision problems Runge-Kutta methods dominates. However there is possibly one exceptional case where predictor-corrector dominates. It is the case of high-precision solutions of very smooth equations with complicated right-hand side evaluations.

Considering multistep approach it is important to realize the difference between integrating an ODE and finding the integral of a function. For a function , the integrand has a dependence on the independent variable . However for an ODE, the “integrand” (which is right-hand side) depends both on and dependent variables . So in order to advance the solution of from to we have :

|  |  |
| --- | --- |
|  | (2‑18) |

According to a multistep approach is approximated by a polynomial passing through several previous points and possibly through .

The formula that is evaluating the integral (2-18) at is then of the form:

|  |  |
| --- | --- |
|  | (2‑19) |

Where

There is a method called which solves an implicit formula of the form (2-19) for . Such method is called . The idea of this method is to take initial guess for , then insert it into the right-hand side of (2-19) and get updated value of . In order to get initial value of we have to extrapolate the polynomial fit to the derivative from the previous points to the new point . The next stage of solving process is made by which is using the prediction step’s value of to the derivative. In conclusion Predictor-corrector method comprises of three separated processes :

* Predictor step
* Evaluation of the derivative from the latest value of .
* Corrector step

#### Adams-Bashforth-Moulton

Probably the most popular method is method. This method has good stability properties . The predictor part is called The Adams-Bashforth :

|  |  |
| --- | --- |
|  | (2‑20) |

The Adams-Moulton part is the corrector :

|  |  |
| --- | --- |
|  | (2‑21) |

## Technologies

Literature review concerning technologies is mainly focused on web application point of view. AJAX approach is taken into account along with Google Web Toolkit. Also Google App Engine’s possibilities for maintaining web-based applications are described.

### AJAX approach

AJAX which stands for Asynchronous JavaScript and XML refers to a set of web-development techniques including the use of JavaScript, CSS and asynchronous HTML requests. The biggest advantage of this set of techniques is bringing desktop-like experience to the client-side of a web application. AJAX applications asynchronously request only the necessary information from a server. That is why they reduce the traffic and the server load comparing to non-AJAX web applications , which synchronously request whole web pages from their servers. Good example to realize it is following situation. Users are trying to log in to their e-mail application. After authentication process the only real job of the application is to replace a login link by some user welcoming message. Following figure presents how web traffic is happening for AJAX and a HTML-only non-AJAX applications for described scenario. [ gwt + appeng]



Figure 2- Comparison of AJAX light traffic needs vs legacy HTML applications

### Google Web Toolkit

Google Web Toolkit is a development toolkit that enables you to build AJAX applications using the Java language which is compiled to JavaScript. GWT minimizes the cross-browser issues and enables productive development of high-performance web applications. It encapsulates the object API and provides a set of ready-to use interface components and widgets. Google Web Toolkit provides framework for building RPC services, which provide certain functionalities that can be accessed asynchronously from web application.

GWT consists of four main components listed below, which provide functionalities for writing AJAX applications.

* **GWT Java to JavaScript Compiler**

GWT compiler compiles and optimizes GWT applications written in Java to JavaScript. That is why an application can be deployed to a web container.

* **GWT Hosted Web Browser**

This component enables you to run and execute GWT applications in JVM , in hosted mode, without compiling to JavaScript . GWT provides such possibility by embedding a special SWT browser control, that contains hooks into the JVM.

* **JRE emulation library**

Contains JavaScript implementations (for the client-side implementation ) of most widely used packages in Java standard class library like , which are used on the client-side. On server-side implementation you are free to use entire Java class library.

* **GWT Web UI class library**

This provides a set of interfaces and classes that enable you to create various UI components and widgets. Moreover you are free to use ready-to-use widgets in your applications.

### AppEngine

Google App Engine is a PaaS cloud computing platform for developing and hosting web applications using Google’s servers and infrastructure. Supports Java, Python and Go runtime environments. GAE as a platform is designed for scalability , robustness and performance. It is very well integrated with Google Web Toolkit. Once you develop application using GWT, you can deploy it on the GAE.

App Engine distributes requests for applications across multiple web servers to prevent application from interfering with another. In order to achieve it applications run in a restricted “sandbox” environment. Applications in this environment can execute code, store and load data from App Engine’s Datastore and examine web requests.

In terms of limitations , an App Engine application cannot :

* Write to the filesystem. All operations connected with storing and quering data have to be done using Datastore.
* Access another host directly or open a socket.
* Make some other kind of system call

### Datastore

The Google App Engine’s Datastore is a non-centralized persistent store, based on Google’s BigTable technology. It provides robust and scalable storage for web applications running on App Engine. Datastore is not a relational database based on join queries, it is rather a property-value store holding specified objects known as . It was designed to manage scaling to very large data sets in distributed architecture. For this reason Datastore is different than traditional relational databases.

Java Datastore API provides low-level operations on entities. Google App Engine SDK includes implementations of JDO and JPA interfaces for modelling and persisting data. It is possible to use Datastore API directly in your applications, however there is also an option to use a framework which simplifies Datastore usage for Java developers. There are three frameworks recommended by Google App Engine team.

* **Objectify**

Very simple and convenient interface to the App Engine Datastore. Helps with avoiding some complexities of JDO/JPA and low-level Datastore.

* **Twig**

This framework provides a configurable object persistence interface that improves support for inheritance, polymorphism and generic types. Similarly to Objectify helps you to avoid Datastore complexities.

* **Slim3**

Slim3 is not limited only to Datastore. It is a MVC framework which can be used for wide variety of App Engine functions.

Each has one or more named properties . Each property can have one or more values. The available data types defined for Datastore are presented in Table 2-1.

Table 2-1 Data types available in Datastore



There are following limitations concerning Datastore :

Table 2-2 Datastore limitations

|  |  |
| --- | --- |
| Limit | Amount |
| Maximum entity size | 1 megabyte |
| Maximum transaction size | 10 megabytes |
| Maximum number of values in all indexes for an entity | 5000 |
| Text string (short) | Up to 500 Unicode characters |
| Text string (long) | Up to 1 megabyte |
| Byte string (short) | Up to 500 bytes |
| Byte string (long) | Up to 1 megabyte |
| Floating-point types | 64-bit double precision |

# Methodologies chosen and Application design

This section is a combination of methodologies chosen for this thesis project and Design part. This chapter also includes description of software lifecycle model which was chosen for this project. Moreover, description of agile approach applied in this thesis is also described.

## The Prototyping Model

Prototyping lifecycle model is an example of Evolutionary model. In general prototype means a version of the software used to test different solutions to the problem. Such approach allows better understanding of the ways solution might work and could be useful to choose the best solution for the final version of the software. The most important factor in prototyping is the feeding back of experience from each iteration into improving the design.



Figure 3-1 The prototyping lifecycle

During development process of this thesis project, features of the application were developed according to the prototyping model. At the beginning of the cycle, requirements are gathered . Then quick design phase is performed. Another step is to develop prototype according to the prepared design. The next step is evaluation of the developed prototype in agreement with thesis supervisor. Another step is gathering comments about prepared prototype and refining it by repeating the cycle from design part. This process is repeated as long as feature requires improvements and corrections .

## Test Driven Development

Test Driven Development is one of the Agile methodologies.

## Architecture

## Design patterns

## Technologies applied

# Testing

## Test driven development approach

## Testing methods

### Unit testing

Unit testing in gwt

### Integration testing

### System testing

### Cross – browser testing

# Implementation

## Parser

## Solver

## Graph viewer

## Datastore connector

# 

# Results

## Results validation and verification

## Application’s outputs

# Discussion and conclusion

## Solvers correctness

## Limitations

## Problems faced

## Quality of implementation

## Conclusion

## Future work

REFERENCES

[1] Press, William H.; Teukolsky, Saul A.; Vetterling, William T. (2007), “Numerical recipes: the art of scientific computing”, Third edition, Cambridge University Press, Cambridge

[2] Kirpekar, Sujit, (2003), “Implementation of the Bulirsch Stoer extrapolation method” Department of Mechanical Engineering, University of California, Berkeley

[3] Guermeur, Daniel; Unruh, Amy; (2010), “Google App Engine Java and GWT Application Development”, Packt Publishing, Birmingham.

[4] Chaganti, Prabhkar, (2007), “Google Web Toolkit : GWT Java Ajax Programming”, Packt Publishing, Birmingham.

[5] <https://developers.google.com/web-toolkit/overview> (accessed 3rd of August 2012)

[6] <https://developers.google.com/appengine/docs/java/runtime#The_Sandbox> (accessed 3rd of August 2012)

[7] <https://developers.google.com/appengine/docs/java/datastore/overview> (accessed 4th of August 2012)

[8] <https://developers.google.com/appengine/docs/java/datastore/entities> (accessed 4th of August)

[9] Barnes, Stuart 2012, Advanced Software Engineering : Software lifecycle models, course notes, Cranfield University.

[10]

APPENDICES

Whilst Heading 1 to Heading 6 can be used to number headings in the main body of the thesis, Heading styles 7–9 have been modified specifically for lettered appendix headings with Heading 7 having the ‘Appendix’ prefix as shown below.

Datastore

Appendix Section (Use Heading 8)

Appendix Subsection (Use Heading 9)

Creating captions in Appendices

If you have chosen to include chapter numbers in your captions then follow the instructions given here to apply the same format to the captions in your appendices. This section explains how to caption the figures and tables in your Appendices, assuming that Heading 7 is numbered “Appendix A” and that the Figures and Tables are going to be labelled ‘Figure A-1’, ‘Figure A-2’, ‘Table B-1’ etc.

You will have to create new, separate labels that look like the ‘Figure’ and ‘Table’ labels you used in the main body of your thesis.

1. Select the **References** tab on the Ribbon then click on **Insert Caption**
2. Click **New Label**. Type **Figure\_Apx** then click **OK**
3. You now have two labels for figures, called **Figure** and **Figure\_Apx**  
   Repeat for table captions.
4. In the **Caption** box, type your caption text
5. Click **Numbering**. Tick **Include chapter numbering** and choose **Heading 7** from the drop-down list of styles and click **OK** twice
6. Your caption should look something like this:

**Figure\_Apx A‑1 This is the caption text for a Figure in the Appendix**

1. Delete the extraneous ‘\_Apx’ from the caption label so it reads:  
   **Figure A‑1 This is the caption text for a Figure in the Appendix**  
   **TIP:** Instead of deleting each ‘\_Apx’ individually use **Find & Replace** to modify all the labels at once.

Creating Lists of Figures and Tables for Appendices

This template already includes a List of Figures and a List of Tables, however you will have to create two new lists for the ‘Figure\_Apx’ and the ‘Table\_Apx’ labels.

1. Place the insertion point on a blank row after the existing List of Figures
2. Select the **Insert Table of Figures** command on the **References** tab of the Ribbon
3. Set the **Caption Label** box to ‘**Figure\_Apx**’ and click **OK**  
   **Note:** Word will put a single blank line between the original and new lists preventing it from appearing as one seamless list. However if you select the blank paragraph between the tables you can hide it by opening the Font dialog box from the Home tab and selecting **Hidden**.
4. Click after the List of Tables and repeat for the Caption Label ‘Table\_Apx’