Numerical Methods: Lab II

Coding Advanced Numerical Solutions to the Simple Pendulum Problem

David Bernal
Universitat Pompeu Fabra

2017-2018

Introduction

In this lab session, we will continue investigating the Simple Pendulum problem. In the previous session, we implemented Euler and Euler-Cromer, however, we did not evaluate the error of these methods with respect to the exact solution. To get an analytical solution, we will linearize the Simple pendulum and code advanced numerical methods such as Runge-Kutta 2 (RK2), and Runge-Kutta 4 (RK4) to overcome the limitations of the first methods and improve the accuracy of our results in relation to the analytical solution. Throughout the session, we will explore the mathematics of each numerical approximation. Also, we will evaluate the error between the aforementioned numerical solutions and the exact solution.

Materials

Processing

Processing, is an open-source computer programming language with an integrated development environment (IDE) based on *Java*. The simplified C-like syntax of *Processing* is aimed to facilitate the learning of the fundamentals of visual computer programing tasks. You can download *Processing* form `www.processing.org`.

Code

We will provide code with the main skeleton of the simulation framework. Inside this code you will be asked to fill some gaps by coding specific aspects of the solution. You can download the code from the course web site.

We recommend to read carefully the material and code at the following blog `http://encinographic.blogspot.com.es/2013/05/simulation-class-intro-and-prerequisites.html`. The lab material is based on the contents of this blog.

1 Preliminary work

1.1 Background Theory concepts (10%)

1. Explain briefly how the Simple Pendulum can be *linearized* and represented through differential equations.
2. Which will be the variables used to define a state for the *linearized* Simple Pendulum?
3. Calculate the analytical solution of the *linearized* Simple Pendulum.
4. Write the code of the *DrawState()* function to plot the analytical solution of the *linearized* Simple Pendulum.

2 Practical Work

In this section, we will model the *linearized* Simple Pendulum, and will code Euler's method and advanced numerical methods such as Euler-Cromer, RK2, and RK4.

2.1 Defining the States (%15)

The first step on designing the simulation of the *linearized* Simple Pendulum is to encode the description of what we are simulating as a set of numbers. Thus, the *simulation state* consists of a set of numbers that describes the **State** of a physical model in time.

1. Set initial variables. At the Processing file *LinearizedSimplePendulum.pde*, complete the section *Set initial variables*. Give relevant names to the variables you are defining here, to make your code understandable, and assign the initial values you consider reasonable for initial conditions (e.g. PendulumThetaInit = radians (30.0)).
2. State Definition. At the Processing file *LinearizedSimplePendulum.pde*, complete the section *State definition*, and provide a vectorized version of the state definition for the *linearized Simple Pendulum* model. You have to declare a *vector state* whose size is the number of variables considered at Question 1.1, where each variable is an index to the vector position. Also consider the *current time* as another state variable.
3. Create Initial States. Complete the section *Create initial states* at the ***setup()*** function. Fill in the the state vector with the corresponding initial values of the variables considered for the state.

2.2 Programming the ***timeStep()*** and ***DrawState()*** functions (%60)

The time step function serves to calculate the increment of each of the variables of the state, and update their values, using the different numerical methods. On the other hand, the draw function serves to visualize the system. In this session, we will implement the Euler method, Euler-Cromer, RK2, and RK4 and plot their numerical solutions vs. the exact solution.

1. Find the Euler, Euler-Cromer, RK2, and RK4 equations to approximate the solution of the system, for each of the variables defining the *linearized* Simple Pendulum state.

2. Fill in the code at the *timeStep()* function, implementing the above derived equations, to update the *vector state* of the *linearized* Simple Pendulum for each of the numerical methods above mentioned.

3. Write the code of the *DrawState()* function to calculate the position of the end of the Pendulum Arm for each numerical method. Complete the code of the *DrawState()* function to plot the analytical solution of the *Linearized* Simple Pendulum in conjunction with each numerical method.

2.3 Visualization of the system (%15)

Your pde file should generate a graph as illustrated in Figure 1:

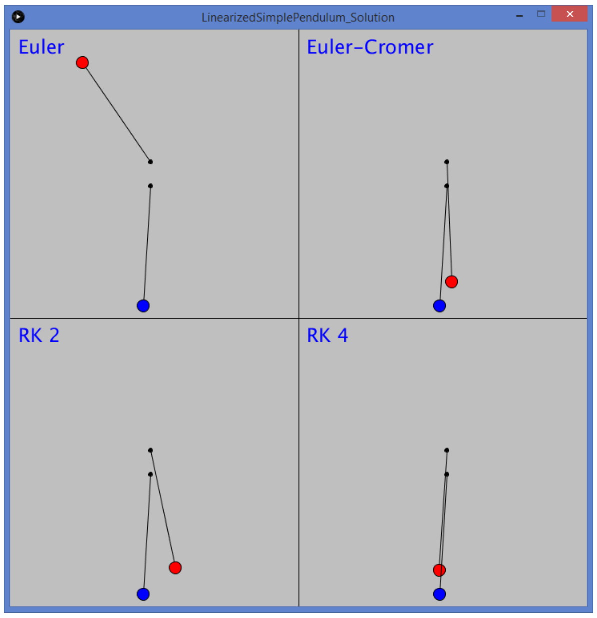

Figure 1: Coding advanced numerical solutions for the linearized Simple Pendulum

1. Add additional code within the *Draw()* function (if not done before in the *DrawState()* function) to generate a plot as illustrated in Figure 1.

Run your code. What do you observe?. Does the simulation run according to reality? Why not? Which might be the cause of the problem? Explain and provide your conclusions.

3 Submission

You have to deliver both preliminary and practical work as word and pde files, respectively. Your code should be commented and clear, easy to understand and easy to follow. You have to submit as well a written report with the answers to the questions, any observations relevant to the development of the lab, and conclusions of your work. All the files should be submitted in a zip file though the Aula Global.

IMPORTANT NOTE: we will follow a blind revision scheme, so USE ONLY YOUR NIAS at the code and the document. DO NOT PUT YOUR NAMES ANYWHERE