Programming Seminar: 1. Ordinary Differential Equations

Mauricio Sevilla[1,2]

[1]Universidad Nacional de Colombia
[2] E-mail: jmsevillam@unal.edu.co

4th of April of 2018

UNIVERSIDAD
NACIONAL
DE COLOMBIA

Contents

Introduction

Solving differential equations is a daily task in physics. Some times this is not a trivial thing to do analytically.

This is why we need to develop numerical tools to solve those equations

Analytics

First we will take some specific cases just to compare the numerical results, to do so, it is necessary to know the analytical solution of some problems.

We will use for verification

- Decay Model.
- Harmonic Oscillator.

Decay Model

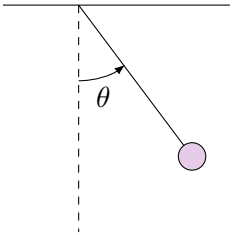
The description of the nuclear and atomic decay follow the same ODE

$$\frac{dP(t)}{dt} = -\frac{1}{\tau}P(t) \tag{1}$$

And the solution is

$$P(t) = A\exp\left(-\frac{t}{\tau}\right) \tag{2}$$

Pendulum: Small Oscillations



The pendulum equation goes as,

$$\frac{d^2\theta(t)}{dt^2} + \omega_0^2 \sin(\theta(t)) = 0 \tag{3}$$

but, if we consider the case of small oscillations $\sin(\theta) \approx \theta$, so we get a harmonic oscillator equation,

$$\frac{d^2\theta(t)}{dt^2} + \omega_0^2 \theta(t) = 0 \tag{4}$$

And the solution is

$$\theta(t) = A \sin(\omega_0 t + \delta) \tag{5}$$

Pendulum: Forced & Damped

In this case we have

$$\frac{d^2\theta(t)}{dt^2} + \gamma \frac{d\theta(t)}{dt} + \omega_0^2 \sin(\theta(t)) = F(t) \tag{6}$$

We may have some idea of the solution, but

How to calculate it analytically?

It depends strongly on $F(t)$

Numerics I: Theory

The numerics is based on discretization, the computer cannot *understand* continuous variables such as time, position, momentum.

The sampling is a very important part on the discretization.

The discretization can be done in many ways,

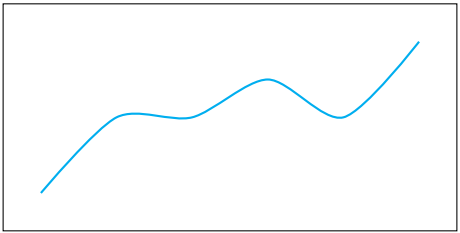


Figure 1: Representation of the discretization

The discretization can be done in many ways,

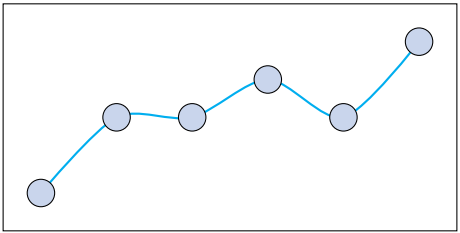


Figure 1: Representation of the discretization

The discretization can be done in many ways,

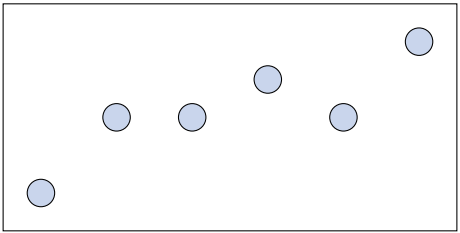


Figure 1: Representation of the discretization

The discretization can be done in many ways,

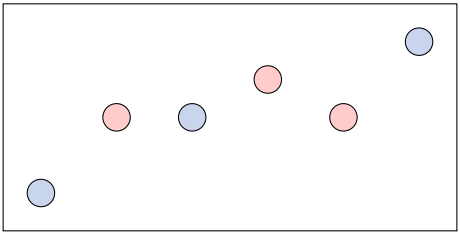


Figure 1: Representation of the discretization

Euler Method I: First Order

Let's consider the specific case of a differential equation of first order.

$$\frac{dx(t)}{dt} = f(x,t) \tag{7}$$

The Taylor expansion of $x(t)$ is

$$x(t+h) = x(t) + \frac{dx(t)}{dt}h + \frac{1}{2!}\frac{d^2x(t)}{dt^2}h^2 + \cdots$$

if we consider h small, we can take the first order approximation

$$x(t+h) \approx x(t) + \frac{dx(t)}{dt}h$$

$$\frac{dx(t)}{dt} = \frac{x(t+h) - x(t)}{h} \tag{8}$$

The solution using the Euler method for the differential equation

$$\frac{dx(t)}{dt} = f(x,t)$$

is

$$x(t+h) = x(t) + hf(x,t) \tag{9}$$

Higher Order Differential Equations

If we do the same as before, for example in a second order ODE, it is necessary to know the first derivative,

$$\frac{d^2x}{dt^2} = f\left(x, \dot{x}, t\right)$$

with $\dot{x} = \frac{dx}{dt}$. If we take $v = \dot{x}$, we get

$$\frac{d^2x}{dt^2} = \frac{dv}{dt} = f(x, v, t) \tag{10}$$

So, we have to solve the system

$$\frac{dx}{dt} = v \tag{11a}$$

$$\frac{dv}{dt} = f(x, v, t) \tag{11b}$$

Euler Method II

The Euler method for the two order differential equation is

$$\begin{aligned} x_{i+1} &= x_i + h v_i \\ v_{i+1} &= v_i + h f(x_i, v_i, t_i) \end{aligned}$$

Euler-Cromer Method

A variation of the Euler method is the Euler-Cromer method, it is also called modified Euler,

$$\begin{aligned} v_{i+1} &= v_i + hf(x_i, v_i, t_i) \\ x_{i+1} &= x_i + hv_{i+1} \end{aligned}$$

Numerics II: Programming

Now, we are going to see how to solve the problems previously mentioned using a computer. The codes are made in `python` just for us to learn some basics concepts and ideas of the language.

Decay Model - Python

Code Decay.py

```
def f(P0):
        return -P0/tau
tau=1.
h=0.01
N=1000
P=1.

for i in range(N):
        print i*h,P
        P=P+h*f(P)
```

We solved the equation

$$\frac{dP(t)}{dt} = -\frac{P}{\tau}$$

using the Euler's method.

Decay Model - C++

Code Decay.cpp

```
#include<iostream>
const double tau=1.;
double f(double P0);
int main(void){
  double h=0.01;
  int N=1000;
  double P=1.;
  for(int i=0;i<N;i++){
    std::cout<<i*h<<'\t'<<P<<std::endl;
    P=P+h*f(P);
  }

  return 0;
}

double f(double P0){
  return -P0/tau;
}
```

We solved the equation

$$\frac{dP(t)}{dt} = -\frac{P}{\tau}$$

using the Euler's method.

Results

to run the program, we use on the terminal the command

```
$ python Decay.py > DataDecay.dat
```

Or the C++ version

```
$g++ Decay.cpp && ./a.out > DataDecay.dat
```

So, in the file `DataDecay.dat` we get data, so, in order to do the analysis them we need to plot them, to do so we use GNUplot.

GNUplot

First, to open GNUplot, use on the terminal

```
$ gnuplot
```

If not installed,

- on Ubuntu

```
$ sudo apt-get install gnuplot
```

- on Mac

```
$ brew install gnuplot
```

Then to plot the data saved on DataDecay.dat,

```
set terminal pdf size 4,2.5
set output 'Decay.pdf'
set xlabel 'Time'
set ylabel 'Probability P(t)'
plot 'DataDecay.dat' w lp t 'Simulation', exp(-x) t 'Theoretical'

set terminal qt
replot
```

Understanding GNUplot

- `set terminal pdf size 4,3.5`:
  GNUplot uses different *terminals* in order to make the output in different formats. `size` is optional, every terminal has different units, pdf uses inches.
- `set output Decay.pdf`:
  The file where the plot is saved.
  Important: The output must have the same extension than the terminal.
- `set xlabel 'Time'`:
  Configures the labels of the plot.
- `plot 'DataDecay.dat' w lp t 'Simulation'`:
  plot the data with lines and points, the comma separates between plots in the same figure.

The file `Decay.pdf` is still open. To close it we must change the terminal and plot again.

```
set terminal qt
replot
```

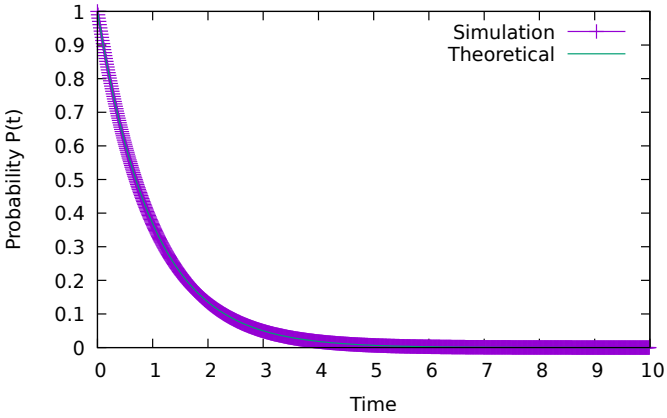


Figure 2: Results of the simulation and theoretical curve

The results of the simulation coincide with the theoretical expectation.

Now, Let's test the method with a different example.

Pendulum: Small Oscillations

We solved the equation

$$\frac{d\theta(t)}{dt} = -\omega_0^2\theta$$

using the Euler's method.

Code Harmonic1.py

```
import math as m
def g(theta0,omega0):
        return omega0
def f(theta0,omega0,t):
        return -w0**2.*theta0
gamma=.2
w0=1.
omega=1.
theta=0.
h=0.1
N=1000
delta=1.
w=.2
F=2.
for i in range(N):
        print i*h,theta,omega
        theta1=theta+h*g(theta,omega)
        omega1=omega+h*f(theta,omega,i*h)
        theta=theta1
        omega=omega1
```

Pendulum: Small Oscillations

Code Harmonic1.cpp

```
#include<iostream>
#include<cmath>
const double w0=1.;
const double h=0.1;
double g(double theta0,double omega0,double t);
double f(double theta0,double omega0,double t);
int main(void){
  double omega=1,omega1; double theta=0,theta1;
  int N=1000;
  for(int i=0;i<N;i++){
    std::cout<<i*h<<'\t'<<theta<<'\t'<<omega<<std::endl;
    theta1=theta+h*g(theta,omega,i*h);
    omega1=omega+h*f(theta,omega,i*h);
    theta=theta1;
    omega=omega1;}
  return 0;}
double g(double theta0,double omega0,double t){
  return omega0;}
double f(double theta0,double omega0,double t){
  return -w0*w0*theta0;}
```

Then to plot the data saved on DataHarmonic1.dat,

```
set terminal pdf size 4,2.5
set output 'Harmonic1.pdf'
set xlabel 'Time'
set ylabel 'Angle {/Symbol q}(t)'
plot 'DataHarmonic1.dat' w lp t 'Simulation', sin(x) t 'Theoretical'

set terminal qt
replot
```

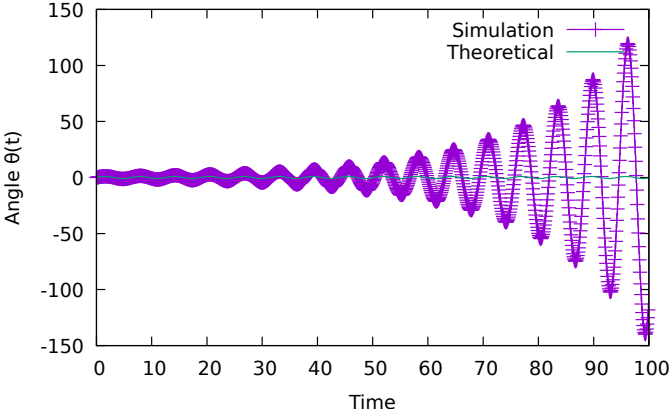


Figure 3: Results of the simulation and theoretical curve

In this case, the method diverges from the theoretical expectation. It seems to be *creating* energy.

To solve that, we must change the numerical method.

Code Harmonic2.py

```
import math as m
def g(theta0,omega0):
        return omega0
def f(theta0,omega0,t):
        return -w0**2.*theta0
gamma=.2
w0=1.
omega=1.
theta=0.
h=0.1
N=1000
delta=1.
w=.2
F=2.
for i in range(N):
        print i*h,theta,omega
        omega=omega+h*f(theta,omega,i*h)
        theta=theta+h*g(theta,omega)
```

We solved the equation

$$\frac{d\theta(t)}{dt} = -\omega_0^2\theta$$

using the Euler-Cromer's method.

Code Harmonic2.cpp

```
#include<iostream>
#include<cmath>
const double w0=1.;
double g(double theta0,double omega0,double t);
double f(double theta0,double omega0,double t);
int main(void){
  double omega=1;
  double theta=0;
  double h=0.1;
  int N=1000;
  for(int i=0;i<N;i++){
    std::cout<<i*h<<'\t'<<theta<<'\t'<<omega<<std::endl;
    omega=omega+h*f(theta,omega,i*h);
    theta=theta+h*g(theta,omega,i*h);
  }
  return 0;
}
double g(double theta0,double omega0,double t){
  return omega0;}
double f(double theta0,double omega0,double t){
  return -w0*w0*theta0;}
```

Then to plot the data saved on Harmonic2.dat,

```
set terminal pdf size 4,2.5
set output 'Harmonic2.pdf'
set xlabel 'Time'
set ylabel 'Angle {/Symbol q}(t)'
plot 'DataHarmonic2.dat' w lp t 'Simulation', sin(x) t 'Theoretical'

set terminal qt
replot
```

In this case, we get way better results than in the previous case.

This method belongs to a family of methods called *Symplectic* i.e. preserves energy.

With this on mind, we can ask ourselves for a problem without analytical solution.

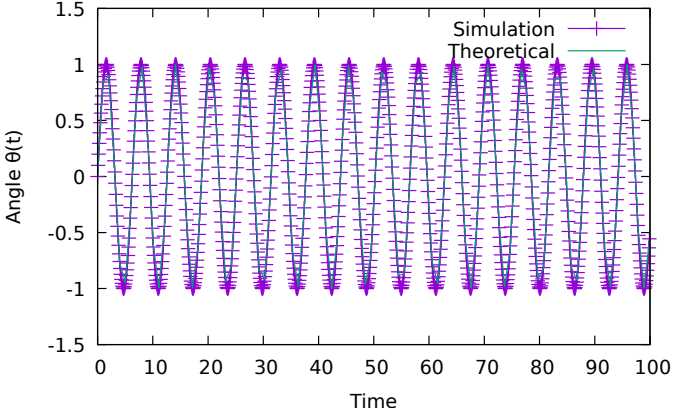


Figure 4: Results of the simulation and theoretical curve

Pendulum: Forced & Damped

Code Pendulum1.py

```
import math as m
def g(theta0,omega0):
        return omega0
def f(theta0,omega0,t):
        return -w0**2.*m.sin(theta0)-gamma*omega0+F*m.sin(w*t+delta)
gamma=.2
w0=1.
omega=1.
theta=0.
h=0.1
N=1000
delta=1.
w=.2
F=.4
for i in range(N):
        print i*h,theta,omega
        omega=omega+h*f(theta,omega,i*h)
        theta=theta+h*g(theta,omega)
```

In this case the equation is

$$\frac{d^2\theta(t)}{dt^2} + \gamma \frac{d\theta(t)}{dt} + \omega_0^2 \sin(\theta(t)) = F(t)$$

and the external force

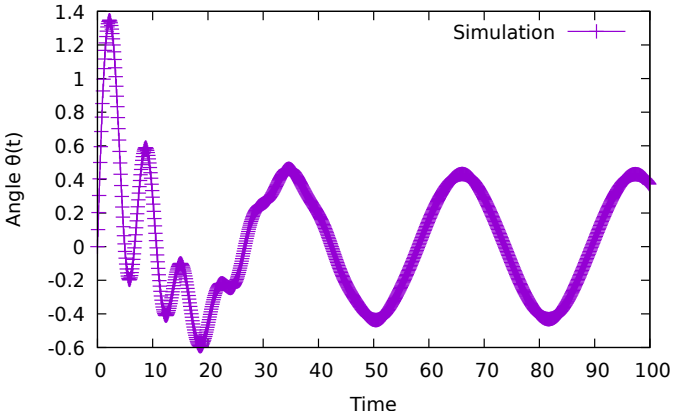
$$F(t) = F \sin(\omega t + \delta)$$

Pendulum: Forced & Damped

Code Pendulum1.cpp

```
#include<iostream>
#include<cmath>
const double w0=1., gamma=0.2;
const double h=0.1, delta=1.0;
const double F=0.4, w=0.2;
const int N=1000;
double g(double theta0,double omega0,double t);
double f(double theta0,double omega0,double t);
int main(void){
  double omega=1;
  double theta=0;
  for(int i=0;i<N;i++){
    std::cout<<i*h<<'\t'<<theta<<'\t'<<omega<<std::endl;
    omega=omega+h*f(theta,omega,i*h);
    theta=theta+h*g(theta,omega,i*h);
  }
  return 0;
}
double g(double theta0,double omega0,double t){
  return omega0;}
double f(double theta0,double omega0,double t){
  return -w0*w0*sin(theta0)-gamma*omega0+F*sin(w*t+delta);}
```

Small External Force



in this case we have two different regimes,

- Damped-Forced regime.
- Asymptotic regime.

Figure 5: Results of the simulation and theoretical curve

Large External Force

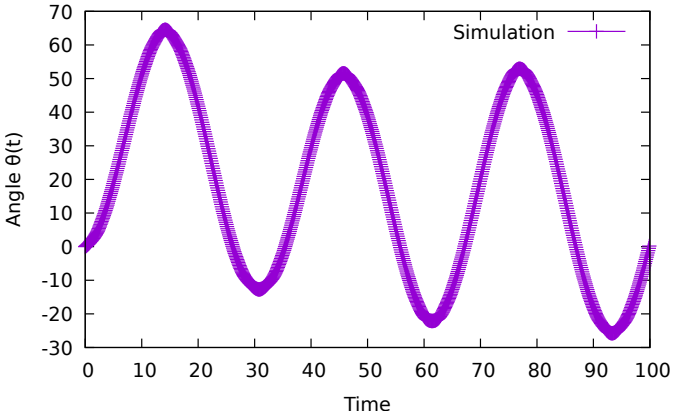


Figure 6: Results of the simulation and theoretical curve

In this case, the external force is big enough to get an angle $\theta > 2\pi$.

Behaviour of the angle in function of time, It is completely different from the case of a small force.

It has strong consequences!

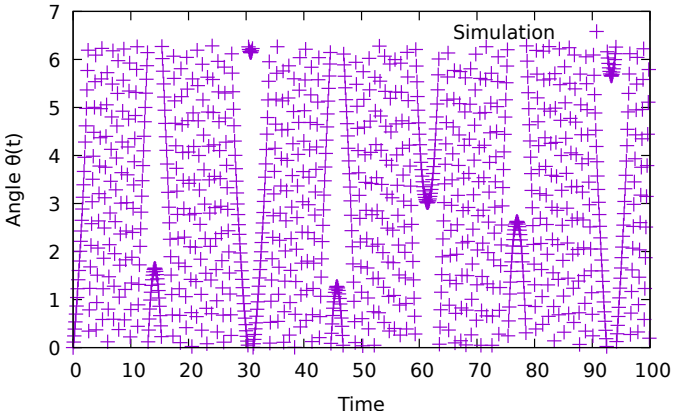


Figure 7: Results of the simulation and theoretical curve

Lyapunov Exponents

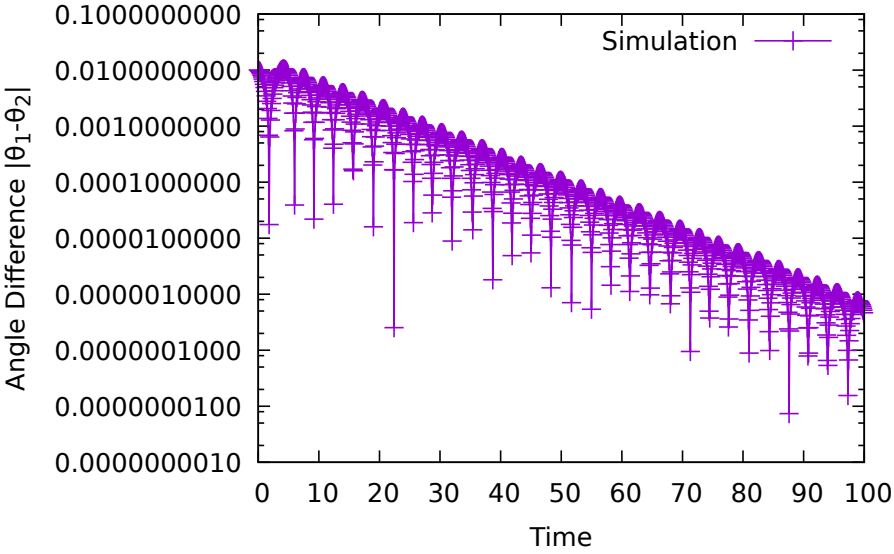
Exercise

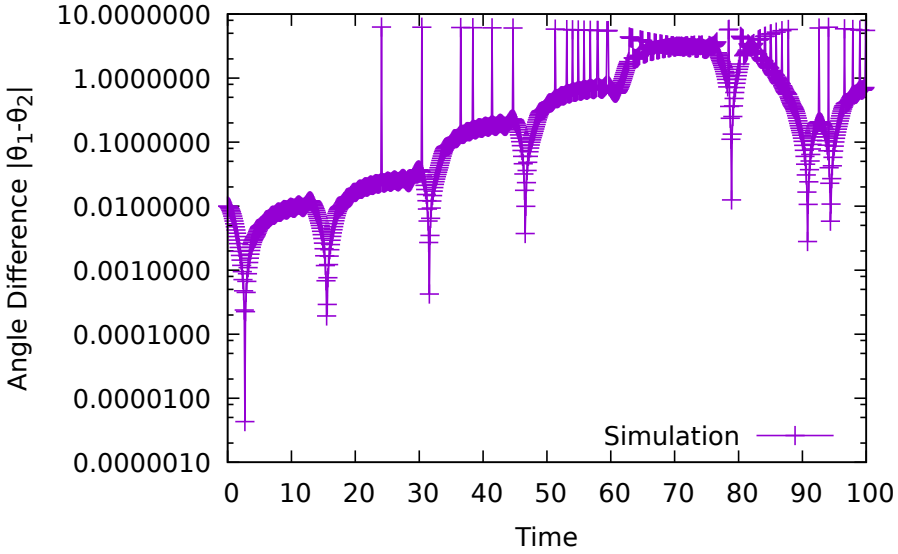
Take the last code and evaluate two different initial conditions (close between each other, but different) and evaluate the difference of the angle with time.
This dependence is exponential and the exponent is known as the Lyapunov coefficient.

The calculation of the Lyapunov exponent is done by

$$\Delta\theta \sim \exp(\lambda t) \tag{12}$$

Can you tell the difference on λ when using a small external force and a large one?


Simulation
Angle Difference $|\theta_1-\theta_2|$
0.1000000000
0.0100000000
0.0010000000
0.0001000000
0.0000100000
0.0000010000
0.0000001000
0.0000000100
0.0000000010
0
10
20
30
40
50
60
70
80
90
100
Time


10.0000000
1.0000000
0.1000000
0.0100000
0.0010000
0.0001000
0.0000100
0.0000010
Angle Difference |θ1-θ2|
0
10
20
30
40
50
60
70
80
90
100
Time
Simulation

References I

[1] Mauricio Sevilla.
Programming seminar: 2018-01.
https://github.com/jmsevillam/Seminar, 2018.
[Online; accessed 25-June-2018].

[2] Python Software Foundation.
The python tutorial.
https://docs.python.org/3/tutorial/index.html, 2018.
[Online; accessed 25-June-2018].

[3] Gnuplot team.
gnuplot homepage.
http://www.gnuplot.info/, 2018.
[Online; accessed 25-June-2018].

References II

[4] cplusplus.com.
C++ language.
http://www.cplusplus.com/doc/tutorial/, 2018.
[Online; accessed 25-June-2018].

[5] A.P. French.
Vibrations and Waves.
M.I.T. introductory physics series. Taylor & Francis, 1971.

[6] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery.
Numerical Recipes 3rd Edition: The Art of Scientific Computing.
Cambridge University Press, 2007.