

# Integrating Computer Modeling into the Curriculum

Francisco Esquembre  
Universidad de Murcia  
SPAIN

---

# Table of contents

1. Introduction: Why should we?
2. Problems with the traditional approach
3. Proposed solution
4. Integrating computer modeling into the curriculum  
Case studies: Mathematics, Chemistry, Physics,
5. Future plans

# **1. Introduction: Why should we integrate computer modeling into the curriculum?**



# 1. Why should we? Ideological claim

Because:

- Modeling can help improve students' understanding of Science.
- Every graduate in Science and Engineering should (must) be familiar with the process of programming a computer for scientific, reasonably complex, purposes.

This second claim can still be considered heretic by many faculty in my University. For all the reasons why ideas are usually considered heretic! (ignorance, resistance to change, fear)

# 1. Why should we? Abilities required

- Graduates don't need to be expert programmers
- But they need to
  - know the order of magnitude required
  - have done it fluently at least during one 1-year course
  - appreciate how programming can positively affect their knowledge of science
  - add programming to their basic skills (Computer Literacy [1]).

[1] A. A. diSessa. "Changing minds. Computers, learning , and literacy". MIT press, 2000.

# 1. Why should we? What to teach

- Numerical analysis and general programming (job-oriented) skills
  - Graduates in Mathematics,
  - Engineering
- Model and simulate physical phenomena
  - Graduates in Physics,
  - Chemistry,
  - Biology,
  - Engineering

Though the borderline is not too strict...

## **2. Problems with the traditional approach**



## 2. Problems: Programming is not trivial

- Plain programming at a high-level language (C, Fortran, Java) is not trivial:
  - Many technicalities involved.
  - Long learning curve.
  - There is not much room in the curriculum for the “Introduction to programming” course.



## 2. Problems: Traditional courses are not appropriated

- Computer scientists teaching “Introduction to programming” teach standard computer-oriented stuff:
  - Use non-scientific examples (bank accounts, jobs in offices,...).
  - Do not teach scientific structures (how to define a Java interface for a function  $f: \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^m$ ).
  - No computer graphics (scientific plots).
  - No graphical user interfaces.
- The result is abstract or boring for science students.

## 2 Problems: No balance between the technique and the science

- If you spend the time in the practical programming there is little room for the science.
- If you spend the time in the science, there is no time for enough practical sessions.
- The **Mathematica**-type solution is too much (though it was used in my Faculty for years)
  - It is too specific (no future employers in the region use it)
  - It is too advanced. Makes students would think this is what all systems do.
  - Too expensive and proprietary.

### **3. Proposed solution: Easy Java Simulations**



### 3. Proposed solution: A tool that makes scientific programming easy

- I started Easy Java Simulations (Ejs) some years ago as a collaboration with colleagues at the Physics Department.
- It became mature when I joined the OSP project due to synergy with other OSP developments.
- Ejs is a tool that makes it easy to create interactive, graphic, scientific simulations in Java.
- Originally targeted to teachers, we now use it with students.

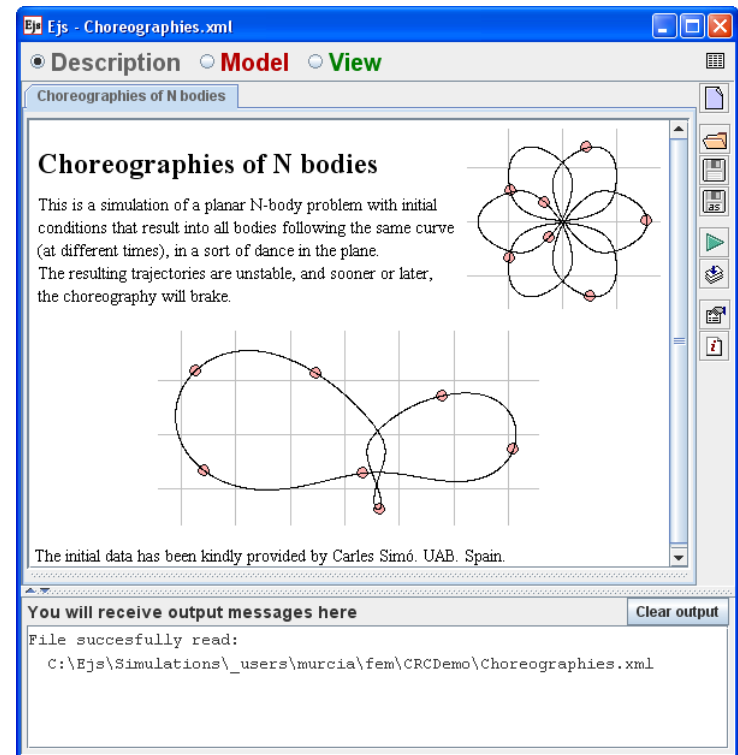
### 3. Proposed solution: Ejs' key features

- Allows users to design and build simulations at a very high, science-oriented level.
- Takes care of all the computer-specific tasks.
- The result is an independent, high quality Java application or applet ready to be published in a Web server.
- Provides powerful interaction and visualization capabilities.

Students spend their time on the science, and still get very nice, sophisticated simulations

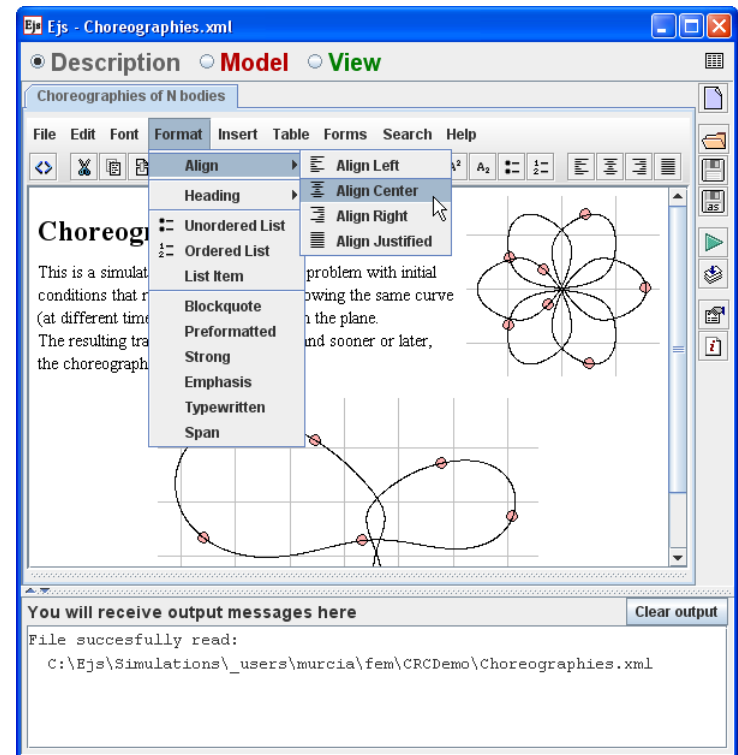
# 3. Proposed solution: How Ejs works

- Has a very simple user interface which covers the powerful OSP engine.
- Organizes the authoring process into *Description*, *Model*, and *View*.
- Each part has a dedicated editor that helps the user build it.



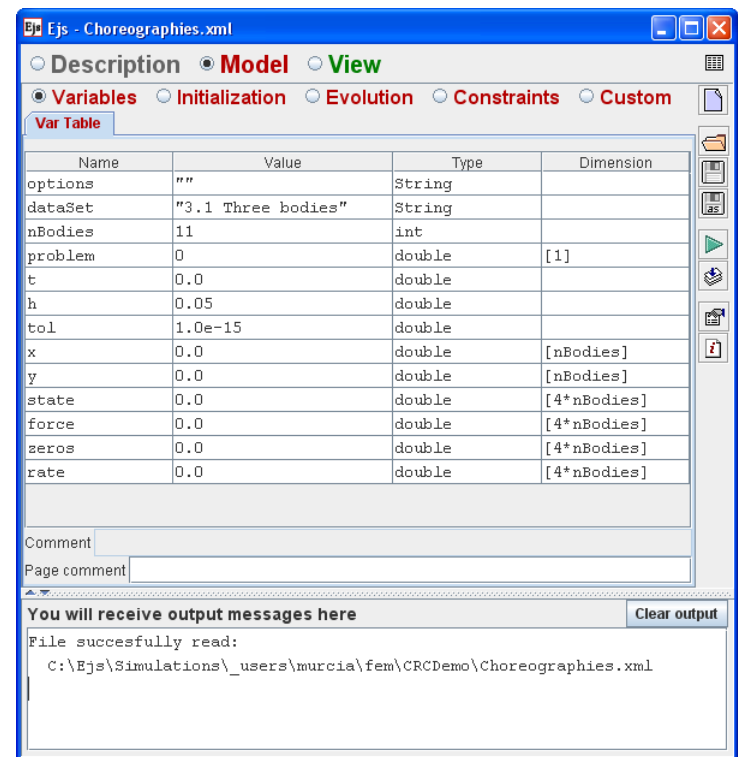
# 3. Proposed solution: How Ejs works: Description

- Provides a simple editor of HTML pages for the simulation.
- The user can add external pages too.
- Each of these pages turns into a real HTML page when the simulation is generated.



# 3. Proposed solution: How Ejs works: Model : Variables

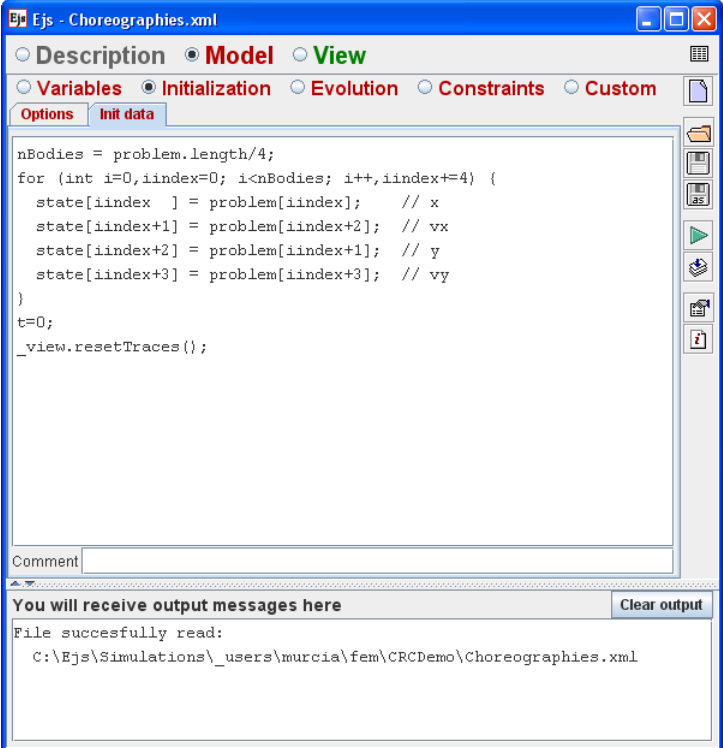
- The interface for the model provides a left-to-right procedure to specify the model.
- The first subpanel allows the definition of the variables that describe the model.
- The user just needs to type a line for each of the variables.





# 3. Proposed solution: How Ejs works: Model : Initialization

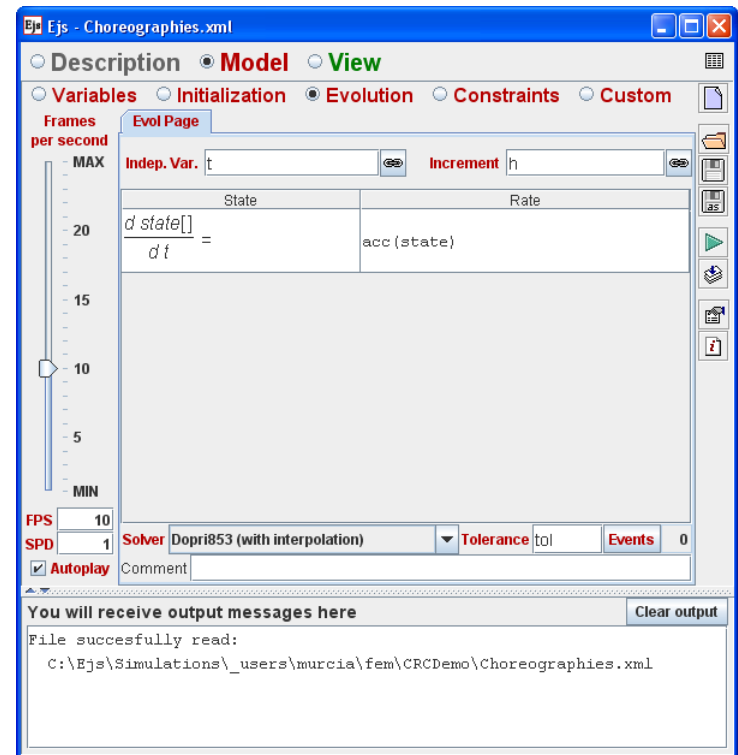
- Additional pages of Java code can be written to initialize the model.
- The user needs to write valid Java code but **only** to express algorithms.
- The editor provides specialized help.



```
Ejs - Choreographies.xml
Description Model View
Variables Initialization Evolution Constraints Custom
Options Init data
nBodies = problem.length/4;
for (int i=0,iindex=0; i<nBodies; i++,iindex+=4) {
  state[iindex ] = problem[iindex]; // x
  state[iindex+1] = problem[iindex+2]; // vx
  state[iindex+2] = problem[iindex+1]; // y
  state[iindex+3] = problem[iindex+3]; // vy
}
t=0;
_view.resetTraces();
Comment
You will receive output messages here
File succesfully read:
C:\Ejs\Simulations\_users\murcia\fem\CRCDemo\Choreographies.xml
```

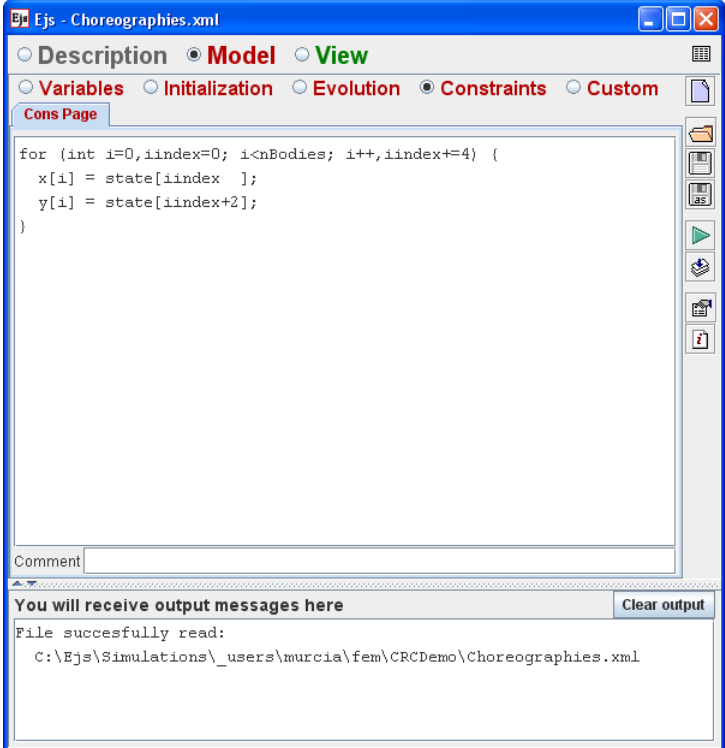
# 3. Proposed solution: How Ejs works: Model : Evolution

- The evolution can be specified with pages of plain Java code (as the initialization).
- Or with a dedicated ODE editor.
- The editor automatically generates the code for different solving algorithms.
- The editor supports arrays and events.



# 3. Proposed solution: How Ejs works: Model : Constraints

- Constraints express additional relationship between variables.
- These relationships must be ensured also under user interaction.
- They are implemented using pages of Java code.



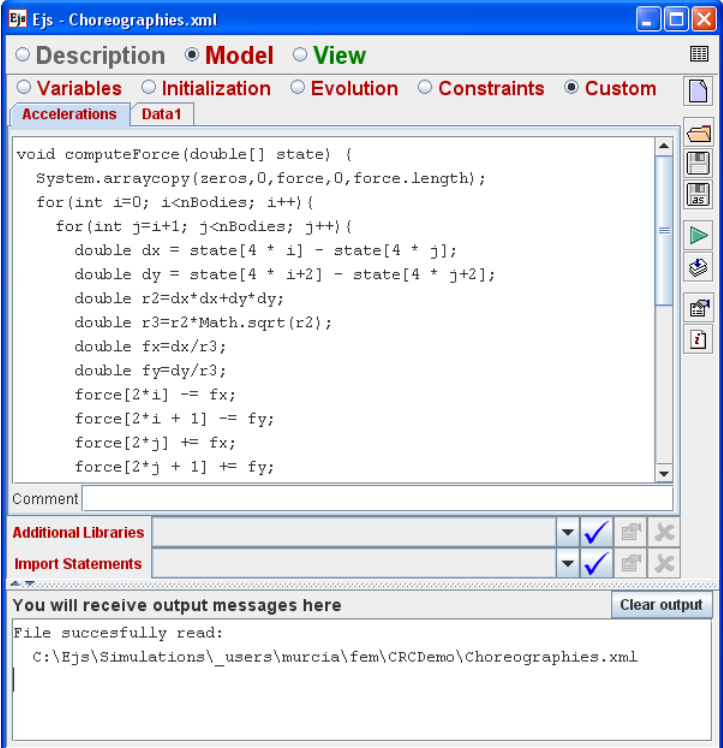
The screenshot shows the Ejs software interface with the title bar "Ejs - Choreographies.xml". The interface has several tabs: "Description", "Model" (selected), and "View". Under the "Model" tab, there are sub-tabs: "Variables", "Initialization", "Evolution", "Constraints" (selected), and "Custom". The main window displays a "Cons Page" with the following Java code:

```
for (int i=0,iindex=0; i<nBodies; i++,iindex+=4) {  
    x[i] = state[iindex];  
    y[i] = state[iindex+2];  
}
```

Below the code is a "Comment" field. At the bottom of the window, there is an output area with the text "You will receive output messages here" and a "Clear output" button. The output area shows the message: "File succesfully read: C:\Ejs\Simulations\\_users\murcia\fm\CRCDemo\Choreographies.xml".

# 3. Proposed solution: How Ejs works: Model : Custom code

- Custom pages of Java code can be created to host extra methods (subroutines and functions) for our code.
- This code must be explicitly used by the user in the other parts.



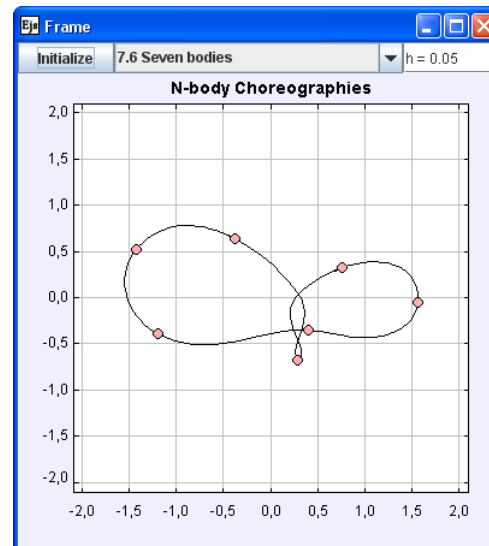
The screenshot shows the Ejs software interface with a window titled "Ejs - Choreographies.xml". The "Model" tab is selected, and the "Custom" sub-tab is active. The main text area contains the following Java code:

```
void computeForce(double[] state) {  
    System.arraycopy(zeros,0,force,0,force.length);  
    for(int i=0; i<nBodies; i++){  
        for(int j=i+1; j<nBodies; j++){  
            double dx = state[4 * i] - state[4 * j];  
            double dy = state[4 * i+2] - state[4 * j+2];  
            double r2=dx*dx+dy*dy;  
            double r3=r2*Math.sqrt(r2);  
            double fx=dx/r3;  
            double fy=dy/r3;  
            force[2*i] -= fx;  
            force[2*i + 1] -= fy;  
            force[2*j] += fx;  
            force[2*j + 1] += fy;  
        }  
    }  
}
```

Below the code editor, there are sections for "Additional Libraries" and "Import Statements", both with checkmarks. At the bottom, an output window displays the message: "File succesfully read: C:\Ejs\Simulations\\_users\murcia\fem\CRCDemo\Choreographies.xml".

# 3. Proposed solution: How Ejs works: View : Panel of Elements

- Creating the view consist in building an appropriated tree-like structure of specialized view elements.



The screenshot shows the Ejs interface for 'Ejs - Choreographies.xml'. The 'View' tab is selected, showing a 'Tree of Elements' on the left and 'Elements for the view' on the right. The 'Tree of Elements' shows a hierarchical structure: Simulation View (Frame) -> PlottingPanel -> TraceSet -> Bodies -> Panel -> Data\_Set -> Initialize -> h. The 'Elements for the view' panel is divided into three sections: 'Interface' (containing icons for file operations and simulation control), '2D Drawables' (containing icons for various 2D plots and data visualization), and '3D Drawables' (containing icons for various 3D plots and data visualization). At the bottom, there is an output window titled 'You will receive output messages here' with a 'Clear output' button. The output text reads: 'File succesfully read: C:\Ejs\Simulations\\_users\murcia\fem\CRCDemo\Choreographies.xml'.

# 3. Proposed solution: How Ejs works: View : Element properties

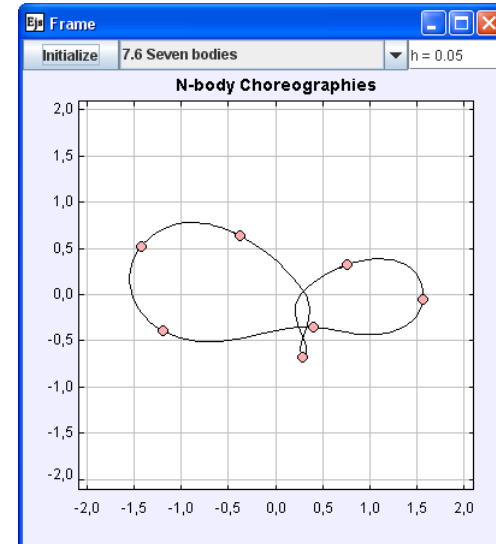
- View elements can be customized editing their so-called properties which can be linked to model variables.

The screenshot displays the Ejs software interface. The main window, titled "Ejs - Choreographies.xml", is in the "View" tab. It shows a "Tree of Elements" on the left with a hierarchy: Simulation View > Frame > PlottingPanel > TraceSet > Bodies. The "Elements for the view" panel on the right shows the "Interface" with various icons. Below the main window, there are three overlapping windows:

- Frame**: A window titled "7.6 Seven bodies" showing a plot of "N-body Choreographies". The plot has a coordinate system from -2.0 to 2.0 on both axes and shows a complex, multi-lobed trajectory with several pink circular markers.
- Properties for Bodies (ParticleSet)**: A dialog box with three tabs: "Number of Elements", "Visibility and Interaction", and "Graphical Aspect".
  - Number of Elements**: Shows "Elements" as "nBodies".
  - Position and Size**: Lists properties for X, Y, Size X, Size Y, Pixel Size, Scale X, and Scale Y, each with a linked icon.
  - Visibility and Interaction**: Includes "Visible" (checked), "Draggable" (true), "Sensitivity", and event handlers for "On Press" (\_pause()), "On Drag" (for (int i=0)), "On Release" (\_play()), "On Enter", and "On Exit".
  - Graphical Aspect**: Includes "Style", "Position", "Rotate", "Line Color", "Fill Color" (pink), and "Stroke".
- Output Window**: A window titled "You will receive output messages here" showing the message: "File successfully read: C:\Ejs\Simulations\\_users\murcia\fem\CRCDemo\Choreographies.xml".

# 3. Proposed solution: How Ejs works: Running the simulation 1

- Ejs generates from this high-level description the simulation when clicking the Run button. ▶
- The simulation can be run as an independent application...
- As a Java applet, within a complete set of HTML pages...
- In a Launcher package easily created within Ejs.



# 3. Proposed solution: How Ejs works: Running the simulation 2

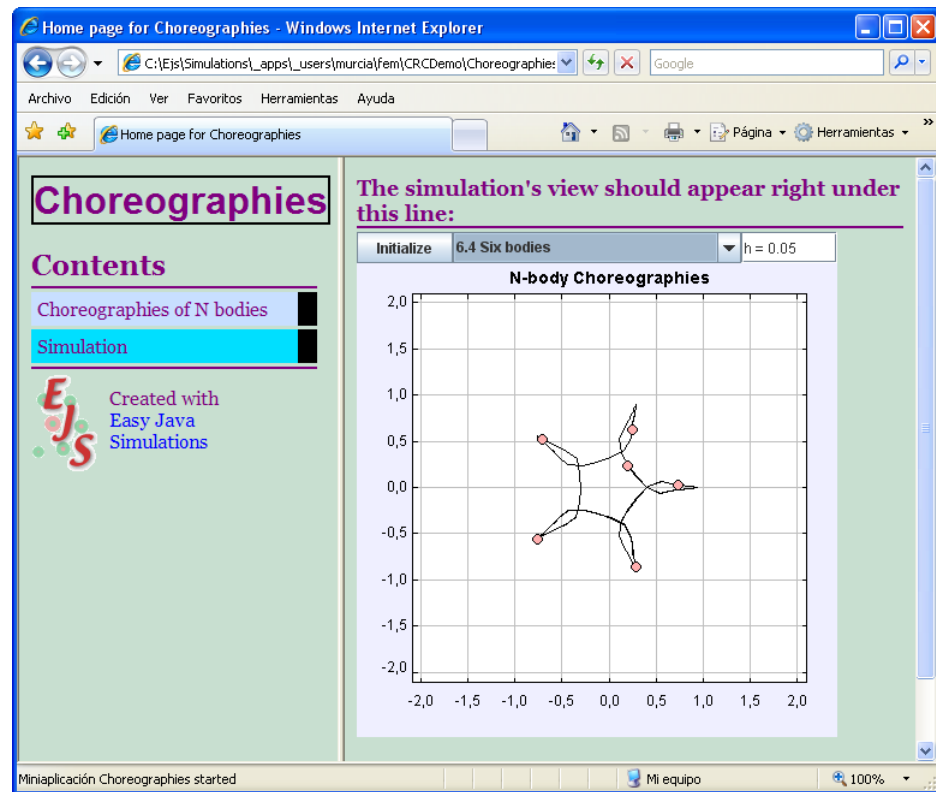
- Ejs generates from this high-level description the simulation when clicking the Run button. ▶
- The simulation can be run as an independent application...
- As a Java applet, within a complete set of HTML pages...
- In a Launcher package easily created within Ejs.

The screenshot shows a Windows Internet Explorer browser window displaying the 'Home page for Choreographies'. The address bar shows the local file path: C:\Ejs\Simulations\_apps\_users\murcia\fer\CRCDemo\Choreographies. The page has a blue header with the title 'Home page for Choreographies'. Below the header is a navigation menu with 'Contents', 'Choreographies of N bodies', and 'Simulation'. The main content area is titled 'Choreographies of N bodies' and contains the following text: 'This is a simulation of a planar N-body problem with initial conditions that result into all bodies following the same curve (at different times), in a sort of dance in the plane.' Below this text is a diagram showing a complex, multi-lobed trajectory on a grid. The text continues: 'The resulting trajectories are unstable, and sooner or later, the choreography will brake.' Below this is another diagram showing a different trajectory on a grid. At the bottom of the page, it says 'The initial data has been kindly provided by Carles Simó. UAB.' The browser's status bar at the bottom shows 'Miniaplicación Choreographies started' and 'Mi equipo'.



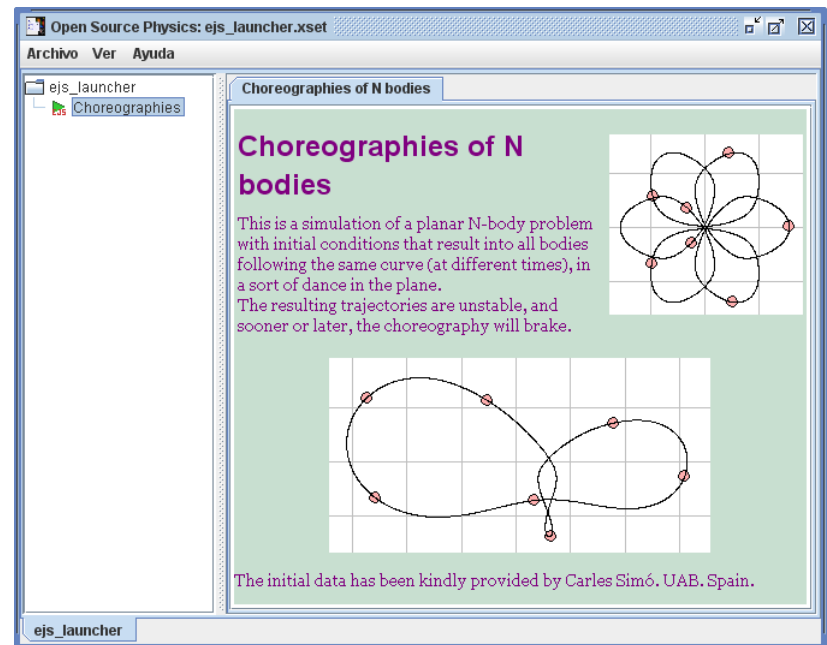
# 3. Proposed solution: How Ejs works: Running the simulation 2

- Ejs generates from this high-level description the simulation when clicking the Run button. ▶
- The simulation can be run as an independent application...
- As a Java applet, within a complete set of HTML pages...
- In a Launcher package easily created within Ejs.



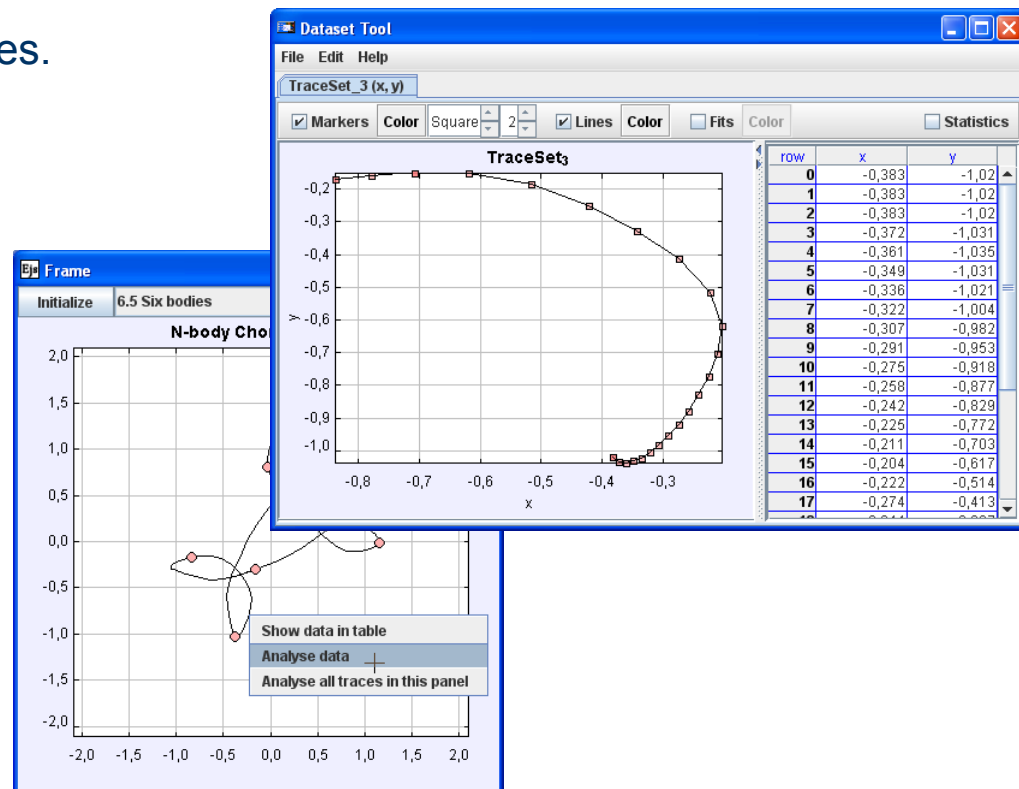
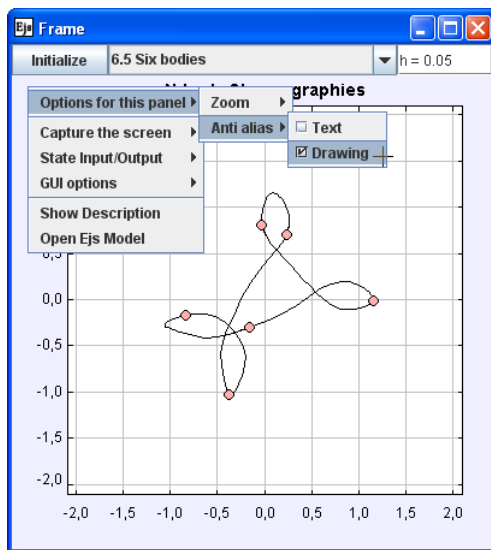
# 3. Proposed solution: How Ejs works: Running the simulation 3

- Ejs generates from this high-level description the simulation when clicking the Run button. ▶
- The simulation can be run as an independent application...
- As a Java applet, within a complete set of HTML pages...
- In a Launcher package easily created within Ejs.



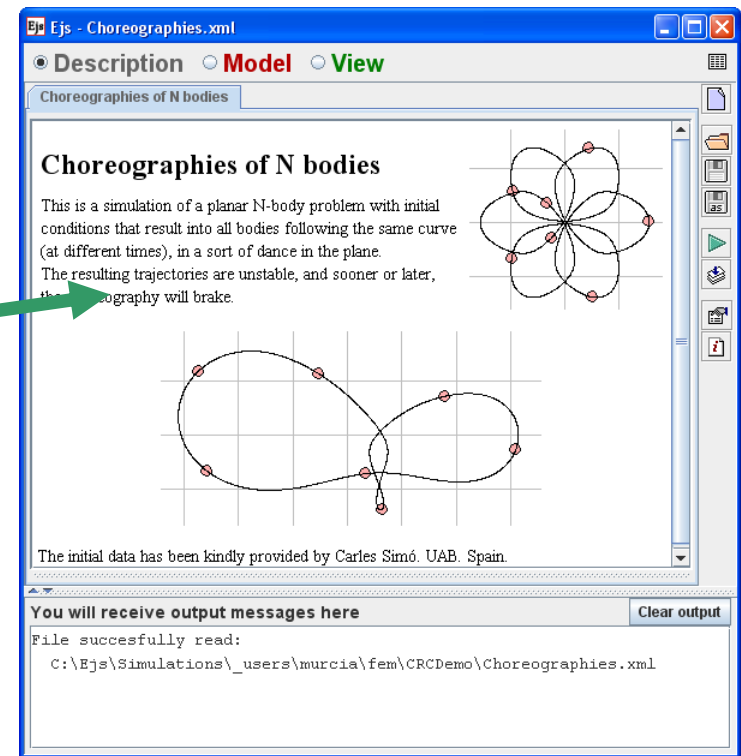
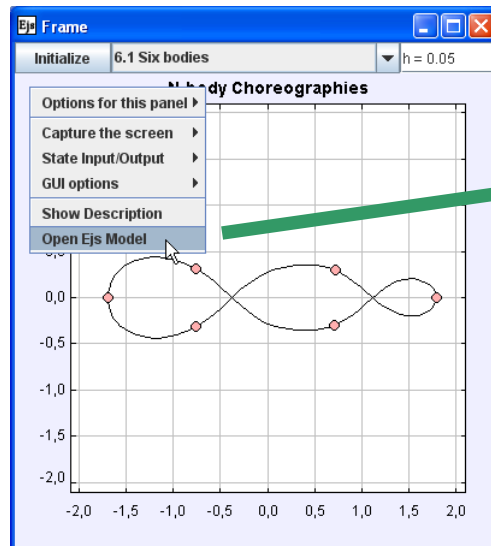
# 3. Proposed solution: How Ejs works: Running the simulation 4

- Simulations created with Ejs implement all current (and future!) OSP facilities.



# 3. Proposed solution: How Ejs works: Running the simulation 5

- Or can open Ejs with the model loaded, thus closing the circle.



# **4. Integrating computer modeling into the curriculum**

**Case studies: Mathematics, Chemistry, Physics**



## 4. Case study: Degree in Maths

- Very much focused on Numerical Analysis
- Students receive compulsory in a 5 years course:
  - 1<sup>st</sup> year: 90 hours of Introduction to programming in Java
  - 2<sup>nd</sup> year: 150 hours of Numerical Analysis I (root-finding, linear algebra, approximation and interpolation)
  - 4<sup>th</sup> year: 90 hours of Numerical Analysis II (Differentiation, Integration, ODE).

Yet, students have limited programming skills. No real object-oriented fluency. No graphics. (Reminder for Paco: say how we check the accuracy of ODE solvers.)

## 4. Case study: Degree in Maths

- In their 5<sup>th</sup> year there is an 60 hours, optional course on Numerical Methods for PDEs.
- Here we use Ejs to:
  - Visualize the solutions (especially important in PDEs).
  - Plot errors and visualize how numerical solutions converge.
  - Show dependence of convergence on the parameters (interaction).
  - Make the course more attractive to students

Run the [Launcher package of the exercises.](#)

## 4. Case study: Degree in Chemistry

- Chemistry students receive (in my opinion) very little Math courses and NO programming instruction at all.
- They have:
  - 1<sup>st</sup> year: 96 hours of Algebra, Calculus, ODE basics, and Statistics + 4 hours (!) of Mathematica + 2 hours (!) of DPGraph
  - 3<sup>rd</sup> year: A 45 hours, optional, purely theoretical course on ODE.
  - 5<sup>th</sup> year: A 45 hours, optional course, called “Complements of Mathematics”.



## 4. Case study: Degree in Chemistry

- Some 5 years ago, I changed the “Complements of Mathematics” course (which was again purely theoretical) into an introduction to modeling and simulation using Ejs.
- Here we use Ejs to provide a basic introduction to programming, create simulations with ODE-based models, visualize and plot scientific data, and a bit of numerical analysis of ODEs (up to RK and RKF).
- Since 2 years, the course is virtual. Reasonable success.

Run the [Launcher package of the exercises](#).

## 4. Case study: Degree in Physics

- 1<sup>st</sup> year: 45 hours of Introduction to programming: ‘simplified’ C, Matlab/Octave, Mathematica.
- 1<sup>st</sup> year: 60 hours of computer lab with Ejs.
  - The course matches the “Introductory Physics” course in concepts studied.
  - Doing it since 3 years.
  - The course is open to any students of any other degree as well.
- 3<sup>rd</sup> year: 75 hours of computational Physics: root-finding, linear algebra, ODE, BVP, PDE, Montecarlo, Fourier analysis. Fortran.
- 4<sup>th</sup> year: Optional 45 hours of “Advanced simulation”:  
(molecular dynamics, Montecarlo, percolation). Fortran.

## 4. Case study: Degree in Physics

- In the 1<sup>st</sup> year computer lab they use Ejs for:
  - Introduction to computers.
  - Visualization: relative motion.
  - Introduction to algorithms: Java.
  - Numerical methods for ODE: Euler method.
  - Solving the equations of motion
  - Elastic forces. Resonance.
  - Applets in HTML pages.
  - Planetary orbits.
  - Numerical integration: Work, Energy.
  - Visualization of electrical fields.
  - Particles in EM fields,
  - EM waves.
  - Numerical integration of Schrödinger's equation.

Run the [Launcher package of a student portfolio.](#)

## **5. Future plans**



## 5. Future plans: Degree in Maths

- I am considering teaching the Numerical Methods for ODEs using either OSP or Ejs.
- Using Ejs in the traditional (now theoretical) course on ODE:
  - 3<sup>rd</sup> year: 75 hours compulsory. Basic theory.
  - 3<sup>rd</sup> or 4<sup>th</sup> year: 45 hours, optional. Qualitative theory of ODEs. Dynamical systems.But this may be considered very heretic.
- New curriculum: Introduce a course on Modeling and simulation. Still to be discussed.

## 5. Future plans: Degree in Chemistry

---

- Continue offering the Ejs-based modeling and simulation course

## 5. Future plans: Degree in Physics

- Convincing the Computational Physics teacher to move to OSP.

**Thank you very much!**

