University of Hamburg
Department of Informatics
Scientific Computing

Continuous Simulation with Ordinary Differential Equations

Seminar Paper

Modeling and Simulation

Julian Busch
Matr.Nr. 6140789
9busch@informatik.uni-hamburg.de

Advisor: Nathanael Hübbe

March 31, 2013

Contents

1 Introduction

Modeling and simulation of a system describes the two processes of deriving a mathematical model that describes the behavior of a system of interest and solving the obtained model to determine the system's response in different situations. The system can be viewed as a set of components that act together to transform an input signal into an output signal. In addition to the input and output variables, the system contains a set of internal state variables. Continuous simulation, in contrast to discrete event simulation, refers to systems which input, output and state variables are defined over a range of time rather than for a set of discrete points in time $t_0, t_1, t_2, \dots$. For example, let a system receive an input from a temperature sensor, then the temperature could be read and provided by the sensor as a continuous signal or only for some discrete points in time, say every 30 seconds. Similarly, a system can change its internal state and behavior only at discrete points in time or continuously. Continuous systems are usually modeled using differential equations. In general, changes in quantities can accurately be described using differentials and derivatives and be related to other quantities via equations. Because differential equations are such a basic mathematical concept, they naturally arise in many different contexts such as physics (celestial mechanics) , earth science (weather and climate modeling), chemistry (reaction rates), biology, social sciences, economics and many more. Historically, continuous simulation is one of the first uses ever put to computers: the first electronic general-purpose computer ENIAC, announced in 1946, was first used by Los Alamos scientists to simulate the dynamics of the hydrogen bomb.

The simulation of a system requires the solution of its mathematical model. In this presentation, we will concentrate on systems that can be modeled with ordinary differential equations. After a brief introduction of the main characteristics of continuous simulation in Section 2, Section 3 will present ordinary differential equations together with Initial value problems and methods for their solution. Since many important equations that are encountered in practice cannot be solved analytically, numerical methods are commonly employed. On this note, the focus of the section as well as that of the whole presentation will be on the numerical simulation of systems that can be modeled with ordinary differential equations. Based on Euler's famous method, several more elaborate methods and extensions that take care of certain issues will be presented together with basic concepts for their analysis. Section 4 attempts to illustrate at least the basic treated concepts by applying them to an example problem, N-body simulations. For this sake, the two steps of modeling our solar system with a system of ordinary differential equations and simulating it using Euler's method are demonstrated. A conclusion is provided in Section 5.

2 Continuous Simulation

Simulating a system means to solve its mathematical model and to predict its behavior in different situations based on that solution. We will commence this section with a short introductory example of a modeling-and-simulation process of a simple continuous-time system (taken from [AA08]) and present two important criteria to classify systems. These criteria can be used to describe the class of systems that are modeled with ordinary differential equations. The section will be concluded with a note on the use of numerical methods for continuous simulation.

Consider a ball that is falling from a height of 100 meters, perhaps because it has been dropped by a person standing on top of a high building. We are interested in how much time it takes the ball to reach the ground and how fast it is during the impact. For the sake of answering these questions, we need to make certain assumptions to build a model that describes the behavior of the falling ball. For simplicity, we will assume that only a constant gravitational force is acting on the ball which is located near sea level. Initially, the ball has a position $x(0) = 100$ and a speed $v(0) = 0$. A possible model is then given by the following system of ordinary differential equations together with the two initial values for x and v:

$$\begin{aligned} x'(t) &= v(t), \qquad & v'(t) &= -9.8 \\ x(0) &= 100, \qquad & v(0) &= 0 \end{aligned}$$

The above equations together with their initial values are called an initial value problem. Initial value problems will be treated in the next section. The above problem can be solved by simply integrating the equations and choosing the constants of integration so that the initial conditions for the functions x and v are satisfied. This yields the following solution:

$$\begin{aligned} x(t) &= 100 - 0.5(9.8)t^2 \\ v(t) &= -9.8t \end{aligned}$$

Now that the position and velocity of the ball are given as functions of time, we can use these functions to simulate the fall. The answers to our initial questions are that the ball needs 4.5175 s to reach the ground and its impact speed is -44.2719 m/s. Obviously, the model is too simple to be realistic but depending on the context of its application and the required accuracy, it can be refined, for example by considering the effect of air resistance.

The system that is modeled in the falling-ball example has two important properties: its time variable is continuous and the ball is treated as a point mass. In general, systems can be classified based on the continuity of their time variable and based on their spatial characteristics, among other criteria of course. For a *continuous-time system*, the input, output and state variables are defined for all t in some time interval $[t_i, t_j]$, where for a *discrete-time system*, the variables are only defined for a discrete set of points in time $\{t_0, t_1, t_2, \ldots\}$. Regarding its spatial characteristics, a system can be either lumped or distributed. *A lumped model* is obtained by ignoring the physical dimensions of a system. In the falling-ball example, the ball was treated as a point mass with no spatial extent so that there was no dependence on spatial coordinates and only one independent variable appeared in the equations, the time variable t. In contrast, a *distributed model* does consider the physical dimension of a system and consequently, the spatial coordinates appear as independent variables in addition to t. For example, the distribution of heat over time in a given region can be described by the heat equation. In this model, the temperature is a function of the spatial coordinates and the time variable t.

In this presentation, we will consider only continuous and lumped systems. These systems can be modeled with ordinary differential equations, which will be introduced in the next section. For distributed systems, the more general partial differential equations are needed. In contrast to ordinary differential equations, partial differential equations deal with functions of multiple independent variables, like for example the temperature function in the heat equation.

Because many important ordinary differential equations cannot be solved analytically, solutions are often approximated using numerical methods. It should be noted that these methods usually provide approximations of the solution at consecutive points in time $t_0, t_1, t_2, \ldots$, so that the formerly continuous model is *discretized*. This discretization process can entail several problems that need to be taken care of, for example issues involving aliasing, but we won't discuss them here.

3 Ordinary Differential Equations

This section attempts to provide a brief introduction to ordinary differential equations, which model the type of systems considered in this presentation, along with methods for their solution. We will focus on initial value problems for ordinary differential equations, of which we have already seen an example in the last section, and how to solve them numerically. Based on the simple and famous Euler method, several more elaborate numerical methods and extensions that take several issues into account are introduced along with basic concepts for their analysis. Several examples are included to illustrate the treated concepts.

3.1 Differential Equations

An *ordinary differential equation* (ODE) is an equation that contains a function of one independent variable and its derivatives. It is ordinary in the sense that no partial derivatives are involved, as in *partial differential equations*, since they only apply to functions of multiple independent variables. We have already seen examples of ODEs in the last section. Additional examples are the following two equations:

$$\begin{aligned} y'(t) &= y(t) \\ x''(t) &= \frac{F(t, x(t))}{m} \end{aligned}$$

We will further examine the first equation in the following. The second equation is a famous equation from classical mechanics and represents Newton's Second Law, which states that the total force that is exerted on a point particle is proportional to its acceleration. Since the force depends on the position of the particle, x appears in the equation along with its second derivative, so that the equation is an ODE.

In general, ODEs can take various different forms, which makes the solution as well as the whole field of study somewhat sophisticated. The *general form* of an (explicit) ordinary differential equation is

$$F\left(t, y, y', \ldots, y^{(n-1)}\right) = y^{(n)}$$

where $y = y(t)$ and $y^{(n)}$ denotes the n-th derivative of y with respect to the independent variable t, which is often interpreted as time. The order n of the highest order derivative appearing in the equation is called the *order* of the equation. If y is a vector of functions and F is a vector valued function, a *system* of *coupled* differential equations is specified, but we will mostly consider the special case of a single equation.

Most general-purpose programs for the numerical solution of ODEs expect first order equations as in input so that for higher order equations an *order reduction* is necessary. It is always possible to reduce an ODE of order n to an equivalent system of n first order equations by introducing new functions for the derivatives:

$$\begin{pmatrix} y_1' \\ y_2' \\ \vdots \\ y_{n-1}' \\ y_n' \end{pmatrix} = \begin{pmatrix} y_2 \\ y_3 \\ \vdots \\ y_n \\ F(t, y_1, \dots, y_n) \end{pmatrix}$$

The order reduction process can be illustrated with Newton's Second Law, a second order equation. Introducing the two functions $y_1 = x$ and $y_2 = x'$ (where $x' = v$ is the velocity) yields the following first order system that can be solved using methods for first order equations:

$$\begin{pmatrix} x' \\ v' \end{pmatrix} = \begin{pmatrix} v \\ F(t, x)/m \end{pmatrix}$$

It is appropriate to sort general ODEs into various different classes of equations that have special or interesting properties. Two of these classes are autonomous and linear equations. For an *autonomous equation*, the function F on the left hand side of the general form equation does not depend on t. If t represents time, these equations are also called *time-invariant systems*. Intuitively, if the system that is modeled with an autonomous equation produces an output $o(t)$ for some input $i(t)$, then for any shifted input $i(t + \delta)$, it produces the shifted output $o(t + \delta)$. The system in the falling-ball example is time-invariant because it doesn't matter, at which point in time the ball has been released. The position and velocity will change at the same rate for any starting point of the fall. A *linear equation* is an equation for which the function F can be written as a linear combination of the derivatives of y:

$$y^{(n)} = \sum_{i=0}^{n-1} a_i(t) y^{(i)} + r(t)$$

If $r(t) = 0$, a linear equation is called *homogeneous*, otherwise it is called *inhomogeneous*. Linear equations are well understood and have several special properties, regarding for example their solution and determining the stability of their solutions.

Formally, a solution of a general n-th order equation is a function that is n times differentiable and satisfies the equation. Such a solution can be defined on all of $\mathbb{R}$, in which case it is called a *global solution*, or only on some non-expandable interval, then it is called a *maximal solution*. Without additional conditions, equations have a *general solution* that contains a number of independent constants. This is because the equation describes only the slope its solutions, not the actual values. A *particular solution* is gained by setting the constants to specific values to fulfill additional conditions. For an *initial value problem*, a particular solution is specified by a given initial value $y(t_0) = y_0$, for a *boundary value problem*, conditions for more than one point are given, typically for the endpoints of some time-interval. The model in the falling-ball example is an initial value problem. Integrating the equations yielded a general solution. The particular solution that satisfied the initial values was gained by setting the constants of integration to appropriate values. Some but not all ODEs have solutions that can be written in exact and closed form. There exist several techniques for their solution, for example *direct integration*, *separation of variables*, using the *Laplace transform* and specialized methods that apply to individual classes of equations. We won't discuss these techniques in detail, instead a simple example of an application of the separation-of-variables technique is given in the following:

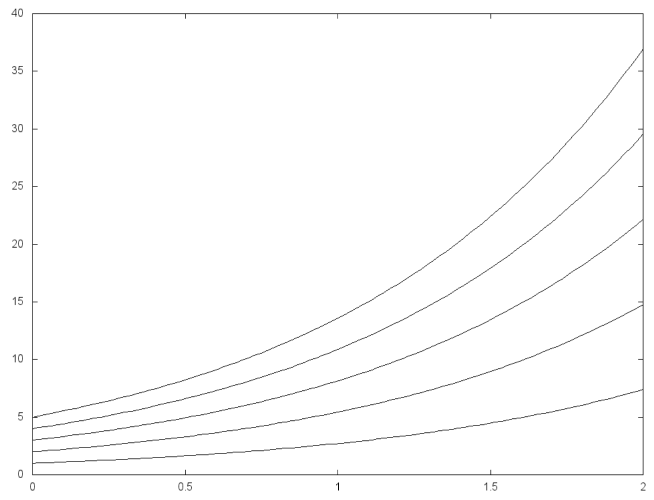
Consider the equation $y' = y$. If $y \neq 0$ on its whole domain, we can write the equation as $y'/y = 1$ so that the dependent variable y is separated on the left-hand side of the equation and integration yields

$$\int \frac{y'}{y} dt = \int 1 dt = t + C.$$

Since an antiderivative of y'/y is $\ln |y|$, we get

$$\ln |y| = t + C \quad \text{or} \quad y = \pm e^C \cdot e^t$$

so that all solutions have the form $y = Ae^t$ for some constant factor $A \neq 0$. For $A = 0$, $y = 0$ is a solution. Given an initial value $y(0) = y_0$, a particular solution $y = y_0 e^t$ is specified. The following plot shows the solutions for different initial values y_0:



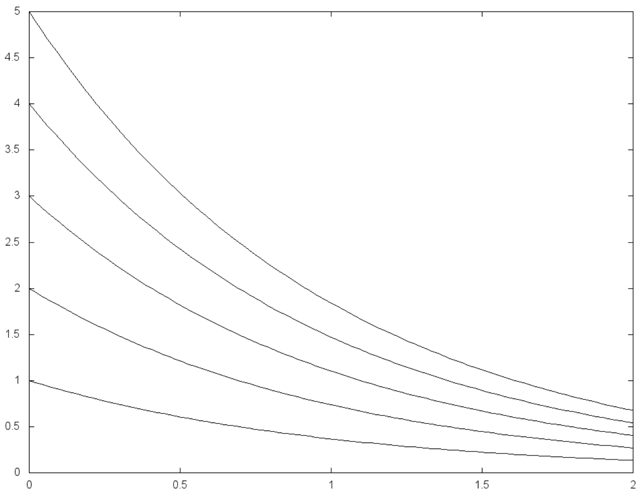
3.2 Initial Value Problems

As mentioned above, an *initial value problem* (IVP) is an ODE together with an *initial value* $y(t_0) = y_0$ that specifies a particular solution. When modeling a system, one is interested in a function $y(t)$ that describes the state of the system as a function of time, given the initial state of the system y_0 and an ODE that determines its evolution. Since most numerical methods only apply to first order equations and higher order equations can always be reduced to first order systems, the following can be used as a *standard form* of an IVP:

$$y' = F(t, y), \qquad y(t_0) = y_0$$

where in the general case the left-hand part is a first order system and the right-hand part specifies the initial values for the functions y_i. Solvers also expect that F is continuous in t in a region including y_0 and that the partial derivatives $\partial F_i/\partial y_j$ are bounded there (F_i is Lipschitz-continuous in y_j). Under these assumptions, the IVP has a solution in that region and the solution is unique.

There are several issues that need to be taken into account when solving IVPs numerically, one of them being the stability of the solutions of the differential equation. Roughly speaking, the *stability* for an ODE reflects the sensitivity of its solutions to perturbations of the initial value. An ODE is called *unstable*, if the solutions diverge with time so that perturbations will grow. An example for an unstable ODE is the equation $y' = y$ since the difference between the solutions is exponential in time and thus the solutions rapidly move away from each other. Conversely, if the solutions of an ODE converge with time, it is called *stable*. In this case, perturbations of the initial value will be damped out with time. An example is the equation $y' = -y$ with solutions $y = Ae^{-t}$:



If the solutions are neither converging or diverging, the equation is said to be *neutrally stable*. A formal definition of stability that can often be found in literature involves of the Jacobian matrix $J_f(t, y)$ which contains the partial differentials $\partial F_i(t, y)/\partial y_j$. The stability of the corresponding equation can be determined based on certain properties of the eigenvalues of J_f. Furthermore, the definition can be simplified for linear equations.

Another important and related property of differential equations is *stiffness*. A stable ODE for which the solutions converge too rapidly is called *stiff*. Since some numerical methods are very inefficient for stiff equations, they need to be treated with special care. Examples will be provided in the following section.

3.3 Numerical Methods

Even in practice, there are many differential equations that need to be attacked numerically because either an analytical solution does not exists or it would be too difficult or inefficient to compute it. However, an approximation provided by a numerical method is often good enough to solve a given problem and there are several theoretical concepts that can be used to analyze and control the incurred error. The numerical methods considered in this presentation calculate a sequence of approximation values $y_1, y_2, y_3, \ldots$ for the solution y at consecutive points in time $t_0+h, t_0+2h, t_0+3h, \ldots$, where h is called the *step-size*. The idea is to predict future values of y by simulating the behavior of the system that is modeled by the differential equation and starts at the initial state y_0. These methods are sometimes referred to as *discrete variable methods* since the values of the approximate solution are calculated step by step using discrete increments.

In stepping from one value to the next, the numerical method usually incurs some error so that the new approximation value lies on a different solution of the differential equation than the previous one. The stability of the equation has an influence on whether such errors are magnified or diminished with time.

3.3.1 Euler's Method

Euler's method is probably the most simple and popular method for solving initial value problems for ODEs. It is also a trivial case of more general techniques and can serve as a starting point for the development of more advanced classes of methods. As a discrete variable method, Euler's method steps from one approximation value to the next and generally lands on a different solution after each step. These solutions are called *local solutions*. Consider the local solution u_n on which the approximation value y_n lies and its Taylor expansion around t_n:

$$u_n(t_n + h) = u_n(t_n) + hu_n'(t_n) + \frac{1}{2}h^2 u_n''(t_n) + O(h^3)$$

Euler's method uses the first two terms as an approximation for the value $u_n(t_n + h)$ and computes the next approximation value y_{n+1} as

$$\begin{aligned} y_{n+1} &= u_n(t_n) + hu_n'(t_n) \\ &= y_n + hF(t_n, y_n) \end{aligned}$$

since the derivative $u_n'(t_n)$ is given by the differential equation as $F(t_n, u_n(t_n)) = F(t_n, y_n)$. Geometrically, Euler's method starts at the initial value and takes small steps along the tangent lines through the previous approximation values, as illustrated in Figure 3.1. It can be seen that using a smaller step-size leads to a better approximation, in fact, for any (useful) method, the numerical solution converges to the exact solution as $h \to 0$ so that a smaller step-size produces better approximation values, for the cost of a greater number of computations. For Euler's method, this behavior can easily be verified by rearranging the stepping-rule to get

$$u_n'(t_n) = \frac{u_n(t_n + h) - u_n(t_n)}{h}.$$

The right-hand side is a differential quotient that converges to the left-hand side as $h \to 0$ so that the method provides an exact solution in the limit.

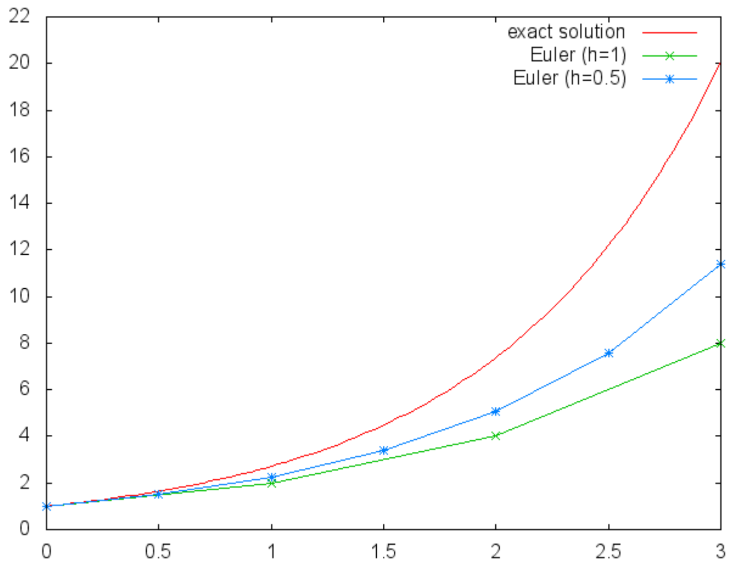


Figure 3.1: Euler's method applied to the IVP $y' = y,\ y(0) = 1$ (exact solution in red) using step-sizes $h = 1$ (green) and $h = 0.5$ (blue).

We will also try to apply Euler's method to a stiff equation. The IVP $y' = -2.3y, \quad y(0) = 1$ will serve as an example. Figure 3.2 shows the result of the application of Euler's method with step-sizes $h = 1$ and $h = 0.7$. It can be seen that the numerical solution for $h = 1$ oscillates and grows without bound. For $h = 0.7$, the solution converges but is still relatively far off the exact solution. The stiffness of the differential equations forces Euler's method to use step-sizes that are much smaller than a step-size that would be appropriate to the behavior of the solution.

3.3.2 Analysis

The following presents several concepts that can be used to evaluate the quality of numerical methods. When using a numerical method, two different types of errors can

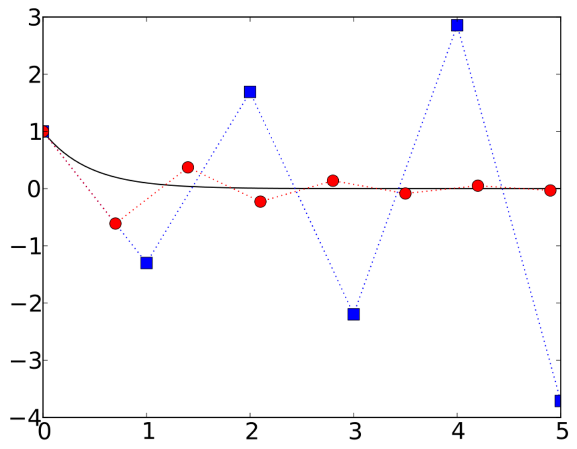


Figure 3.2: Euler's method applied to the IVP $y' = -2.3y$, $y(0) = 1$ (exact solution in black) using step-sizes $h = 1$ (blue) and $h = 0.7$ (red). Source: `http://en.wikipedia.org/wiki/File:Instability_of_Euler's_method.svg`

be incurred, rounding errors and truncation errors. *Rounding errors* are due to the finite precision of floating-point arithmetic while *truncation errors* are due to the method itself and would remain even if all arithmetic could be performed exactly. These two types of errors are not independent since for example a smaller step-size reduces the truncation error but also leads to a greater rounding error. However, in practice the truncation error is usually the dominant type of error. It can be further broken down into the

- *Global truncation error*
$$e_n = y_n - y(t_n)$$
which is the difference between the computed solution and the true solution which is passing through the initial value

and the

- *Local truncation error*
$$l_n = y_n - u_{n-1}(t_n)$$
which is made by a single step of the method, where u_{n-1} is the local solution which is passing through the previous approximation value y_{n-1}.

The global error is not necessarily equal to the sum of the local errors. For an unstable equation the global error is generally grater than the sum of the local errors since the solutions move away from each other and thus the local errors are magnified. What we want for a method is a small global error, but only the local error can be controlled directly, for example by reducing the step-size.

How much the local error decreases with the step-size for a certain method is depicted by its *order*. A method has order p if

$$l_n = O(h^{p+1})$$

which means that if the step-size is halved, the local truncation error that is incurred in one step of the method is divided by a number that has a magnitude of 2^{p+1}. Euler's method for example has order 1 which can be shown by comparing the approximation value that is computed in one step with the exact local solution given in the form of a Taylor series.

Similarly to the concept of stability for differential equations, *stability* can be defined for numerical methods. Intuitively, a method is said to be *stable* if it produces stable solutions so that errors are not magnified. In the previous example, we have seen that Euler's method can produce unstable solutions for stiff equations if the step-size is too large. In general, the stability of a method does not only depend on the stability of the equation being solved but also on the method itself and the step-size being used. Even a high-order method can produce large global errors when it has a low stability. In literature, the term 'stability' is not clear-cut, several different definitions can be found.

3.3.3 Generalizations

Euler's method is often not accurate enough for practical applications due to its low order and stability issues. There are several approaches for developing more accurate methods based on Euler's method of which the following attempts to give a brief survey, starting with the Backward Euler method.

The *Backward Euler method* differs from Euler's method in that it evaluates the function F at the end of the time-step interval and not at the beginning. Exchanging $F(t_n, y_n)$ with $F(t_{n+1}, y_{n+1})$ leads to

$$y_{n+1} = y_n + hF(t_{n+1}, y_{n+1}).$$

Since y_{n+1} appears on both sides of the equation, the Backward Euler method needs to solve an algebraic equation. For non-trivial cases, standard iterative solution methods like Fixed-point iteration or Newton's method can be used. A starting guess for an iterative method can be obtained by computing a step with an explicit method or simply by using a previous approximation value. The Backward Euler method has the same order as Euler's method and it is not obvious why using information at time t_{n+1} and solving an algebraic equation provides a better solution, but it turns out that the Backward Euler method has better stability characteristics and is effective for stiff equations.

The Backward Euler method is an example of an implicit method. *Explicit methods*, like Euler's method, use information at time t_n to compute the solution at time t_{n+1}. *Implicit methods* use information at time t_{n+1} and thus need to evaluate F with the argument y_{n+1} before its value is known which leads to the solution of an algebraic equation. The approximation value y_{n+1} that is to be computed is only given *implicitly* as the solution of the algebraic equation, hence the name. Implicit methods are generally more stable than comparable explicit methods.

Upon reaching time t_n, a numerical method has already computed several values $y_n, y_{n-1}, \dots$ and $F(t_n, y_n), F(t_{n-1}, y_{n-1}), \dots$. It can be shown that these values can be exploited to construct higher-order methods, different classes of methods can be defined based on how these values are combined. Inside a class the methods differ in how many previous values they use to compute the next approximation value. In general, the more previous values are used, the higher the order of the method. Important classes of methods are the implicit *Backward Differentiation Formulas*, the explicit *Adams-Bashforth methods*, the implicit *Adams-Moulton methods* and the explicit and implicit *Runge-Kutta methods.* The Euler method is included in all of the mentioned classes of explicit methods as a trivial case, similarly the Backward Euler method is a trivial case of all the mentioned implicit methods.

We will close our survey of numerical methods by presenting two additional methods that can also be sorted into the classes mentioned above, the Trapezoidal rule and Heun's method, along with an example.

The *Trapezoidal rule* is an implicit second order method that can be considered as both an implicit Runge-Kutta method and an Adams-Moulton method. It can be viewed as a combination of the Euler and Backward Euler methods since it uses the average of their respective slope estimates:

$$y_{n+1} = y_n + \frac{h}{2}\left(F(t_n, y_n) + F(t_{n+1}, y_{n+1})\right)$$

A starting guess for y_{n+1} can be provided by an explicit method and be corrected with the implicit formula, either repeatedly (a Fixed-point iteration) or a fixed number of times. Together, the two methods are called a *predictor-corrector pair.* When correcting to convergence, the predictor plays only a very small role and thus the order and stability characteristics of the predictor-corrector pair are that of the corrector. If the predicted value is corrected only a fixed number of times, the order and stability characteristics of both methods have an influence and the analysis is more complicated.

An example of a predictor-corrector pair is *Heun's method* which results from predicting with Euler's method and correcting once with the Trapezoidal rule:

$$\begin{aligned} p_{n+1} &= y_n + hF(t_n, y_n) \\ y_{n+1} &= y_n + \frac{h}{2}\left(F(t_n, y_n) + F(t_{n+1}, p_{n+1})\right) \end{aligned}$$

Performing a single correction step amounts to an explicit method of order 2.

The following example compares the performance of the implicit Trapezoidal rule with that of the explicit Euler method for a stiff equation. Consider the IVP $y' = -15y,\ y(0) = 1$ with the exact solution $y(t) = e^{-15t}$. Figure 3.3 shows two solutions that were computed by Euler's method using step-sizes $h = 1/4$ and $h = 1/8$ and one solution that was computed by the Trapezoidal rule using a step-size of $h = 1/8$. Euler's method exhibits its typical unstable behavior, even when the step-size is halved, while the Trapezoidal rule provides a much better result that is relatively accurate even in the region near the initial value.

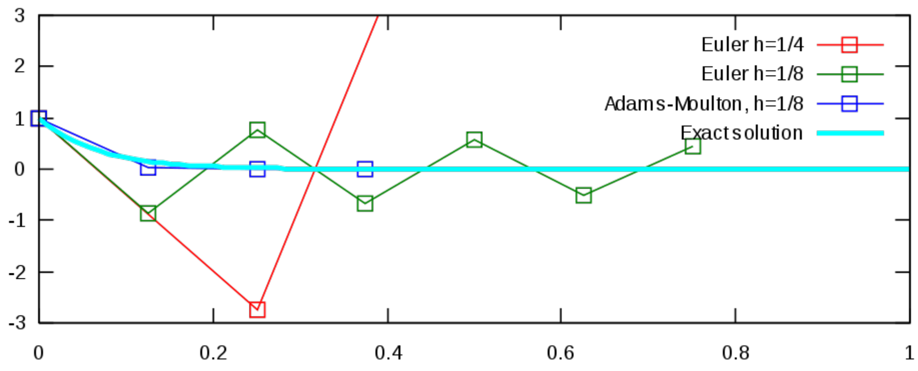


Figure 3.3: The exact solution of the IVP $y' = -2.3y$, $y(0) = 1$ (cyan) and numerical solutions computed by Euler's method using step-sizes $h = 1/4$ (red) and $h = 1/8$ (green) and by the Trapezoidal rule using a step-size of $h = 1/8$ (blue). Source: `http://en.wikipedia.org/wiki/File:StiffEquationNumericalSolvers.svg`

3.3.4 Adaptive Step-size

Until now, all numerical methods used a constant step-size h. However, it is appropriate to use a different size for each step to ensure stability or keep the local error below some tolerance level that can be specified by the user. In some regions of the solution, the step-size needs to be reduced to ensure stability or accuracy while in other regions, the step-size can be increased to save computation time. An example for a method that achieves the accuracy goal is the *Runge-Kutta-Fehlberg method* RKF45 which produces a 5th- and a 4th-order estimate for the approximation value and uses the difference as an estimate for the local error. The step-size is then adapted to the error estimate. The standard function for solving initial value problems for ODEs in MATLAB and OCTAVE, `ode45`, is an implementation of the RKF45 method.

4 Application: N-body Simulations

For the sake of illustrating the basic concepts presented in the previous sections, their application is demonstrated for gravitational N-body simulations, a widely used tool in astrophysics. This section is also inspired by a project on which the author worked during the last summer semester together with Marcel Lebek.

The *gravitational N-body problem* is the problem of predicting the motion of N gravitationally interacting particles. Its applications range from systems of few bodies to solar systems and even systems of galactic and cosmological scale. We will use the N-body problem to model our solar system, considering the Sun, the eight inner and outer planets and the dwarf-planet Pluto, so that in our case $N = 10$.

4.1 Model

The mathematical formulation of the gravitational N-body problem can be obtained by simply combining two physical laws from classical mechanics, Newton's Law of Universal Gravitation and Newton's Second Law of Motion, to get the following second order system:

$$a_i = x_i'' = G \cdot \sum_{j \neq i} \frac{m_j(x_j - x_i)}{\|x_j - x_i\|^3} \qquad (i = 1, \dots, N)$$

which consists of $3N$ equations, one per body and spatial coordinate, where initial values for the positions x_i and the velocities v_i are given. Notice that the equations are coupled, because the acceleration a_i depends on the positions of all bodies. This needs to be taken into account when solving the system. The formulation of the N-body problem in terms of ordinary differential equations is possible because the bodies are treated as point particles. It has already been published by Isaac Newton in his famous *Principia* in 1687. The problem has been completely solved for $N = 2$, for $N = 3$, solutions exist for special cases. However, no practical analytical solution for the general case is known until today and thus the problem is usually solved using numerical methods. Since the presented methods only apply to first order equations, we will need to reduce the system to a first order system, before we can solve it. In the previous section, the order reduction process has already been demonstrated for Newton's Second Law, so we can just adopt the result to obtain the first order system

$$\begin{aligned} x_i' &= v_i \qquad (i = 1, \dots, N) \\ v_i' &= a_i = G \cdot \sum_{j \neq i} \frac{m_j (x_j - x_i)}{\|x_j - x_i\|^3} \qquad (i = 1, \dots, N) \end{aligned}$$

which consists of now $6N$ equations.

4.2 Simulation

Since we cannot solve our solar system model analytically, we will simulate it using a numerical method, for simplicity, Euler's method. Initial values for the positions x_i and the velocities v_i of the bodies considered in our model can be obtained from `http://ssd.jpl.nasa.gov/?horizons`. Starting with the initial values, Euler's method computes a series of approximations for the positions and velocities of all bodies, the values after a time-step of length h are given by

$$\begin{gathered} x_i(t_n + h) = x_i(t_n) + h v_i(t_n) \quad \text{and} \\ v_i(t_n + h) = v_i(t_n) + h a_i(t_n). \end{gathered}$$

A solver can easily be implemented in a programming language of choice and the obtained approximation values can be used for further calculations or visualization. Figure 4.1 shows the trajectories of the inner planets over a time interval of one year which have been computed with Euler's method using 1000 time-steps per day. To obtain stable orbits, we were forced to use a very small step-size which makes the simulation computationally inefficient and increases the influence of rounding errors. In practice, one would use more elaborate methods which are closer to the nature of the problem to guarantee accuracy and efficiency. Individual techniques exist for the use in different astrophysical domains. An overview is provided by [HT08].

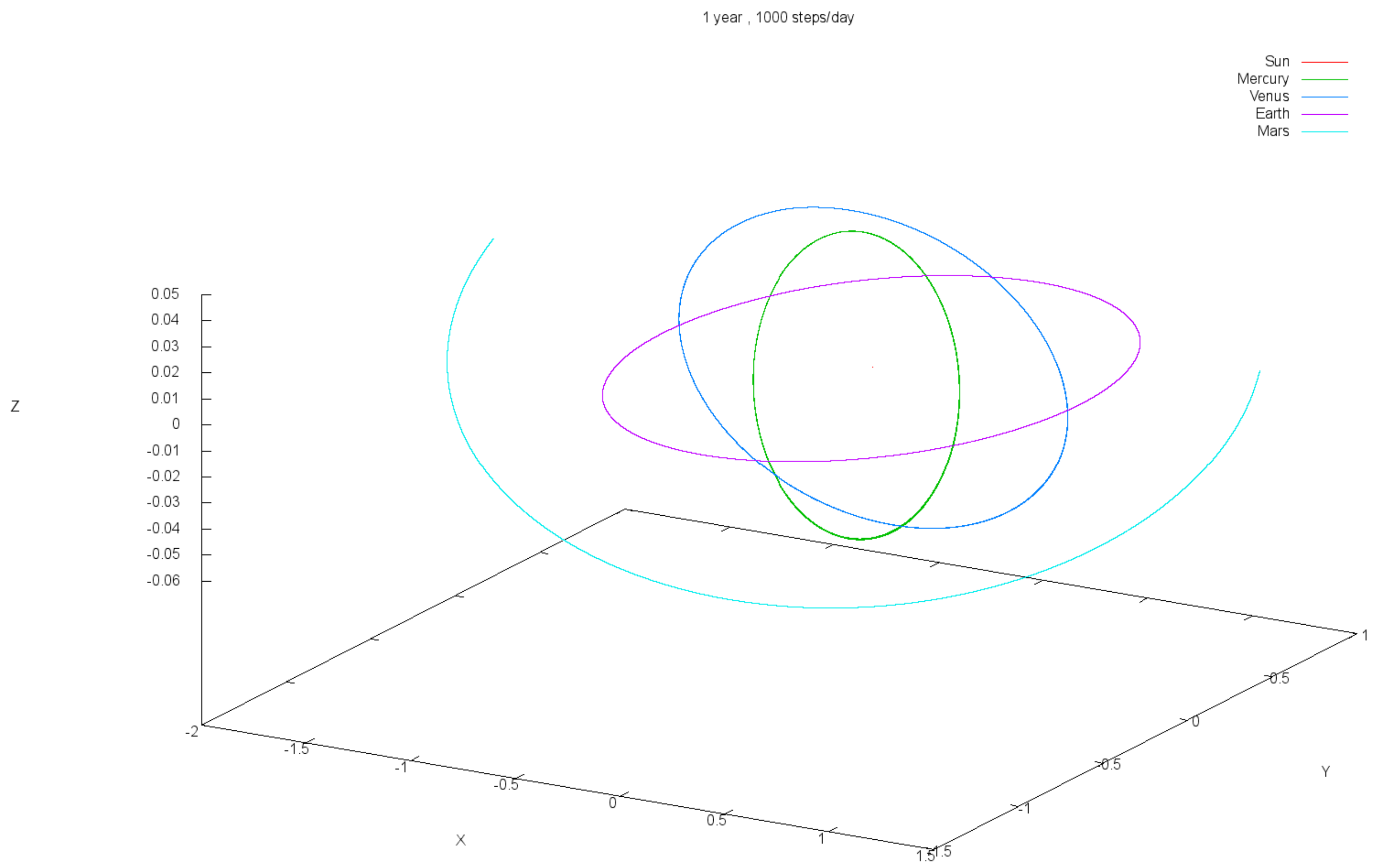


Figure 4.1: Trajectories of the inner planets over a time interval of one year, computed with Euler's method using 1000 time-steps per day.

5 Conclusion

In the course of this presentation, the basic concepts for the modeling and simulation of continuous and lumped systems have been introduced. The considered class of systems can be modeled with ordinary differential equations and simulated in some cases by computing analytical solutions, otherwise by computing an approximate solution using numerical methods. The focus was shifted on the numerical solution of initial value problems for ordinary differential equations where an initial state of the system is given in terms of an initial value for the ODE, which determines the evolution of the system, and the solution function $y(t)$ of the ODE, which describes the state of the system as a function of time, is sought.

Discrete variable methods approximate y by a finite sequence of approximation values $y_1, y_1, y_3, \ldots$ starting from the initial value y_0. These methods can incur different kinds of errors and can be evaluated according to their order and stability characteristics. Many different methods exist, of which several classes have been mentioned and some concrete methods have been presented and compared to each other in examples. In general, higher order methods provide better accuracy but when dealing for example with stiff ODEs, a high order alone cannot ensure a low error. In this case, implicit methods like Backward Differentiation Formulas or Adams-Moulton methods should be employed. Most of the available solvers also vary the step-size being used according to the behavior of the solution to ensure accuracy and stability on one hand and to save computation time on the other hand. In conclusion, it can be said that the choice of an appropriate method for a given problem is crucial for a successful application. Beyond the standard methods, individual numerical techniques exist for solving problems in specific contexts.

Bibliography

[AA08] Samir Al-Amer. Lecture notes for se207: Modeling and simulation. `http://faculty.kfupm.edu.sa/SE/samirha/SE207_072/SE207_March2008.htm`, 2008.

[But08] John C. Butcher. *Numerical Methods for Ordinary Differential Equations.* John Wiley & Sons, 2008.

[Dia96] Florin Diacu. The solution of the n-body problem. *The Mathematical Intelligencer*, 18, No. 3:66–70, 1996.

[Gor03] Dimitry Gorinevsky. Lecture notes for ee392m: Control engineering methods for industry. `http://www.stanford.edu/class/archive/ee/ee392m/ee392m.1034/Lecture2_Models.pdf`, Winter Quarter 2002-2003.

[Hea97] M.T. Heath. *Scientific computing: an introductory survey.* McGraw-Hill series in computer science. McGraw-Hill, 1997.

[Hea02] Michael T. Heath. Lecture notes for 'scientific computing: An introductory survey'. `http://cse.web.cs.illinois.edu/heath/scicomp/notes/index.html`, 2002.

[HT08] Piet Hut and Michele Trenti. N-body simulations (gravitational). *Scholarpedia*, 3(5):3930, 2008.

[TS07] S. Thompson and L. F. Shampine. Initial value problems. *Scholarpedia*, 2(2):2861, 2007.

[Wik13a] Wikipedia. Continuous simulation — wikipedia, the free encyclopedia, 2013. [Online; accessed 31-March-2013].

[Wik13b] Wikipedia. Numerical methods for ordinary differential equations — wikipedia, the free encyclopedia, 2013. [Online; accessed 31-March-2013].

[Wik13c] Wikipedia. Ordinary differential equation — wikipedia, the free encyclopedia, 2013. [Online; accessed 31-March-2013].