Advanced Methods in Scientific Computing
Computer Science 522

Christopher R. Johnson[1]
Department of Computer Science
University of Utah
Salt Lake City, UT
email: crj@cs.utah.edu

Winter Quarter 1997

Contents

List of Figures

Preface

This *book* is collection of notes gathered over the last few years for the graduate course Computational Engineering and Science course Advanced Methods in Scientific Computing (CS 522). The notes contain materials from both original and "borrowed" sources. They are specifically meant to supplement the CS 522 lectures and are not meant for redistribution.

Chapter 1

Overview of Computational Engineering and Science

1.1 Introduction

Let's start this course with a question, "what exactly is *computational science* or *scientific computing* or, as we here in Utah like to call it, *computational engineering and science*?"

Computational Engineering and Science[1] in its most basic definition, is the use of computers to analyze science and engineering problems. A *computational scientist* or *computational engineer*, then, is one who applies high performance computer technology in innovative and essential ways to advance the state of knowledge in their respective discipline or one who designs new methods or techniques for analysis. The other often used term one hears is *Scientific Computing*. Scientific Computing is the term usually used for the development of new software (or hardware) tools for computational applications in science and engineering. Thus, the development of a new scientific visualization system would come under the heading of research in scientific computing, whereas the use of that visualization system to solve a scientific problem would be computational science. One should note that the reference to *high performance computing* is not necessarily analagous to *supercomputing*, but rather to denote the application of the computational science paradigm in solving a science or engineering problem. Which leads us to ask just what is the computational science paradigm?

CES is inherently multidisciplinary. It requires a knowledge of computer science, applied mathematics, and a command of one - and often more - scientific or engineering disciplines [1, 2]. CES, then, can be viewed as the intersection of disciplines as illustrated graphically in figure 1.1. CES is now considered by many as the third basic scientific methodology, complementing theory and experiment. Numerical simulations "fill the gap" between theory and experiment. Simulation provides qualitative and quantitative insight into many phenomena

[1] From here on out, I will abbreviate this as CES

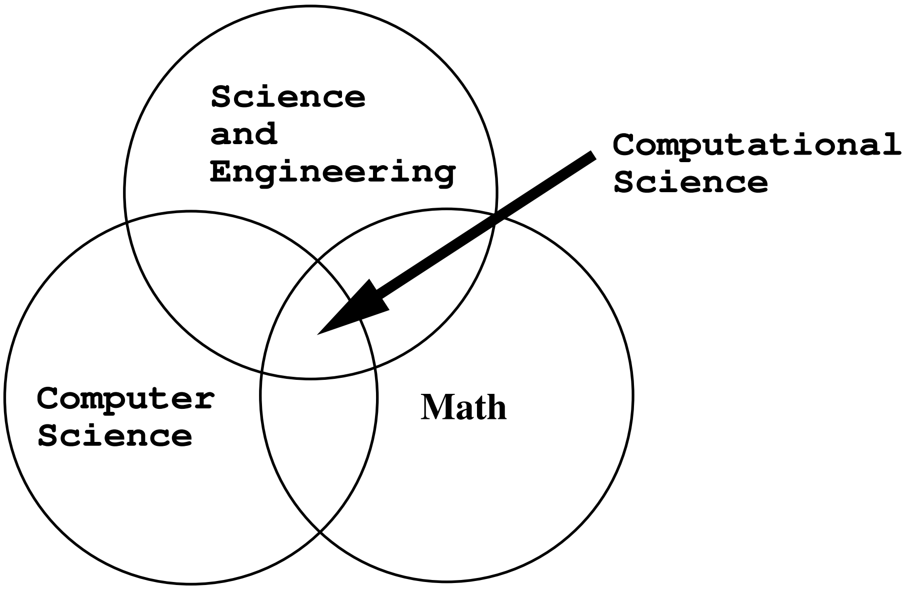

Figure 1.1: Venn diagram of the computational science paradigm.

that are too complex to be dealt with by analytical methods and too expensive, dangerous, time consuming and/or impossible to be performed experimentally. Interesting historical developments of CES can be found in [1, 3, 4]

1.2 System characterization and approach

In characterizing science and engineering problems, the world seems to be split into two distinct types of systems:

- Continuous systems - where variables undergo smooth changes and
- Discrete systems - where changes in variables take place instantaneously in discrete steps.

In the continuous systems approach, we use a continuous model of the continuous system or the continuous approximation of a discrete system. Continuous simulation models are usually described by deterministic differential and/or algebraic equations (historically, analog computers were used to solve these types of models). Components of continuous system modeling and simulation focus on the construction of a physically based model and finding solutions to the differential equation and accompanying initial and boundary conditions which satisfy the governing equations. Important issues tend to be the system size, accuracy of the model and numerical computation, and the efficiency with which the problem is solved.

In the discrete approach, we use a discrete model of the discrete system or the discrete approximation of a continuous system that are discrete approximations of the physical systems. Discrete system simulation models tend to be both stochastic and dynamic in nature. In these models, simulation consists of the observation and analysis of the results obtained by generating random events at different points in time in a computer model of the system.

Before proceeding further with system characterization, let's give a couple of examples.

1.2.1 Coffee Cup Example

As you sit down in front of your terminal, you prepare yourself to work by enjoying a nice hot cup of coffee or tea. Often, however, the coffee is too hot for immediate consumption. Thus, we have to wait awhile for the coffee to cool... but for how long?

In general, the heat flow from the coffee to the surrounding air is very complicated (involving convection, radiation, evaporation, and conduction). However, if the temperature difference is not too large, we can represent the phenomenon by noting the rate of change of the temperature of the object being proportional to the temperature difference. We can express this as a first order ordinary differential equation,

$$\frac{dT}{dt} = -r(T - T_s) \tag{1.1}$$

Where, T is the temperature of the coffee, T_s is the temperature of the surroundings, r is a cooling constant (which is very complicated itself and depends on heat transfer, thermal properties of the body, contact area, etc.), and t is time. This is Newton's law of cooling.

We can, of course solve this equation analytically by integrating:

$$\int \frac{dT}{(T - T_s)} = -\int r\,dt \tag{1.2}$$

Which yields:

$$T = T_s + C_1 e^{-rt} \tag{1.3}$$

We then apply the initial conditions that $T(t = 0) = T_0$ and $T(t \to \infty) = T_s$ which yields:

$$T = T_s - (T_s - T_0)e^{-rt} \tag{1.4}$$

We can also find an approximate solution by changing the o.d.e. into a difference equation. Consider the general o.d.e:

$$\frac{dx}{dt} = f(x, t) \tag{1.5}$$

Let's assume that at $x(t_0) = x_0$. This, along with values for the constants r and T_s, fully specifies our model of the cooling process. Such an o.d.e. is called an *initial value problem* because we have an initial value specified for the dependent variable, x at $t = 0$. While we were able to directly integrate the example above and obtain an analytical solution, there are many differential equations that have no analytic solution, or the analytic solution does not provide an explicit relation between the dependent and independent variables (x and t). In such situations, we use numerical methods to solve the differential equation. This means that we approximate the values of x at specified time steps between the initial value of time and some final value of time t. Equation 1.5 tells us how to approximate x as a value of the neighboring point.

Basically, by looking at the differential equation, one notices that the slope of a solution passing through a given point (x, t) is $f(x, t)$. One can compute values of x, starting with x_0 for subsequent values of t based upon the information from the previous values of x and t. If the value of x is calculated from a combination of more than one previous value it is called a *multi-step* method. If only one previous value of x is used to calculate a new value of x, then it is called a *one-step* method.

The simplest one-step scheme for approximating the solution to the initial value problem given above is by Euler's method [5, 6, 7, 8, 9]. We partition the time interval (t_0, t_f) into N subintervals separated at grid points $t_i = t_0 + ih$ with $h = (t_f - t_0)/N$ and seek approximations ξ_i to $x(t_i)$. The parameter h is called the stepsize. The slope at the point (x_0, t_0) given by the initial conditions is $f(x_0, t_0)$. The line passing through (x_0, t_0) having this slope is then $x - x_0 = f(x_0, t_0)(t - t_0)$. The Euler approximation is the ordinate of this line when $t = t_1 = t_0 + h$; that is,

$$\xi_1 = x_0 + hf(x_0, t_0) \tag{1.6}$$

Having found an approximation ξ_1 to $x(t_1)$, we can continue the process. Euler's method then, is the iteration:

$$\xi_{i+1} = \xi_i + hf(\xi_i, t_i) \tag{1.7}$$

Another way to look at this is utilizing the first two terms of a Taylor's series approximation, for example

$$x(t_0 + h) = x(t_0) + x'(t_0)h + x''(\theta)h^2/2 \tag{1.8}$$

where the third term is called the remainder term and represents the contribution of all the terms not included in the series. If h is small enough, then we can neglect the terms h^2 and higher and simplfy this to

$$x(t_0 + h) = x(t_0) + x'(t_0)h \tag{1.9}$$

if we allow $t_1 = t_0 + h$, then we can generalize this to

$$x_{i+1} = x_i + x_i'h. \tag{1.10}$$

By virtue of equation 1.5 we can substitute the right hand side into 1.10 yielding

$$x_{i+1} = x_i + hf(x_i, t_i), \tag{1.11}$$

which is Euler's method. The method is show graphically in Figure 1.2.

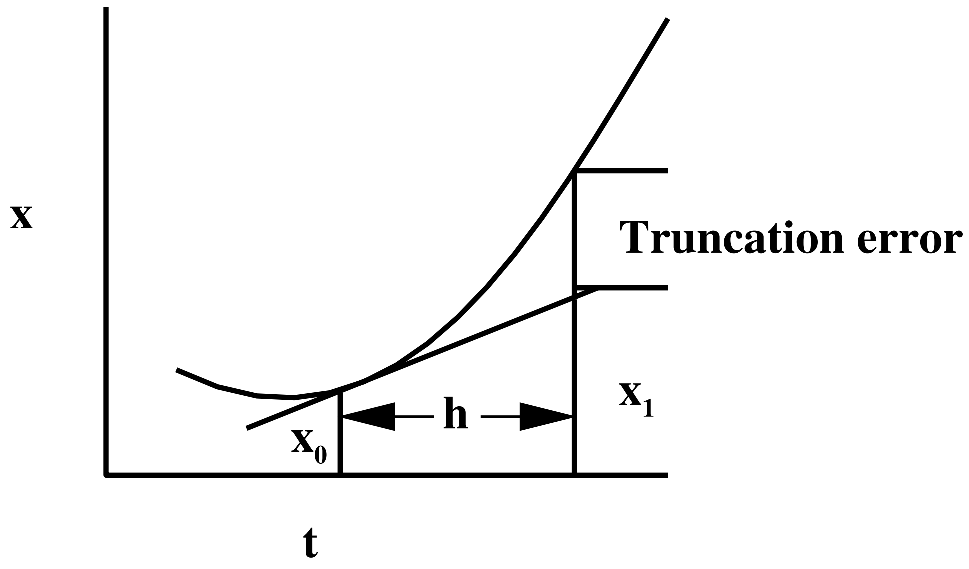

Figure 1.2: Geometric intrepretation of Euler's method.

In order to actually get a numerical solution, we need to come up with and implement an algorithm which solves the equations [6].

Algorithm 1.1 Euler's Method

Function: Solves a first order o.d.e. by Euler's method

Output: x the solution at time t

Inputs: x_0 and t_0, t_f, N (number of subintervals)

Define: the function $f(x,t)$ (the rhs of the differential equation)

Begin

$t \leftarrow t_0, x \leftarrow x_0$

$h \leftarrow (t_f - t_0)/N$

For i $\leftarrow$ 1 to N Do

$x \leftarrow x + hf(x,t)$

$t \leftarrow t_0 + ih$

Output: x, t

End

We then turn our algorithm into computer code, debug it and run our simulation. But what about validating our results? Try solving the coffee cup problem subject to the conditions, $T_s = 22°C, T_0 = 83°C, r = 0.1$.

Once we've implemented our algorithm for Euler's method, we need to ask ourselves some questions regarding the accuracy of the algorithm, for example how can we decide what Δt should be and how do you know if our answer is correct?

One useful characterization of a particular numerical method is its *order* [6]. In terms of approximating a solution to an initial value problem, we define the method to be of order n if there is a constant C, independent of the stepsize h such that

$$|x(t_i) - \xi_i| < Ch^n \qquad i = 1, \ldots, N \tag{1.12}$$

We can estimate the order of Euler's method by looking at a Taylor's series expansion for the original differential equation

$$x(t_{i+1}) = x(t_i) + hx'(t_i) + \frac{x''(\tau_i)}{2!}h^2 \tag{1.13}$$

where $\tau_i \in (t_i, t_{i+1})$. We can rearrange this as

$$x(t_{i+1}) - [x(t_i) + hx'(t_i)] = \frac{x''(\tau_i)}{2!}h^2 \tag{1.14}$$

such that the left hand side is the description of Euler's method. The quantity on the left hand side is called the *local truncation error.* This tells us that each step of Euler's method has a local truncation error proportional to h^2. Such errors can compound (since we are using approximate, rather than exact data). For instance, obtaining an Euler approximation requires $N = (t_f - t_0)/h$ steps. Thus the error of approximation of $x(t_i)$ will be proportional to $h^2(1/h) = h$. Therefore, Euler's method will be of *order one.*

If we apply Euler's method to the differential equation, $dx/dt = (x - 20)$ we see from Figure 1.3 that the Euler method requires a small step size h to provide reasonable levels of accuracy. If the initial value problem needs to be solved for a large time interval, then the method becomes very expensive to compute. In addition, the errors made at each step can accumulate.

Stability

When solving a problem numerically, we wish to ensure that whatever method we use is numerically stable. We've noted that the error for each step of an Euler's method is of order h^2, this is the so-called *local truncation error.* However, we are mostly interested, not in the error of a single step, but the accumulation of such errors over a sequence of steps during the entire computation. Let's analyze Euler's method for stability. Consider the o.d.e.

$$\frac{dx}{dt} = Kx. \tag{1.15}$$

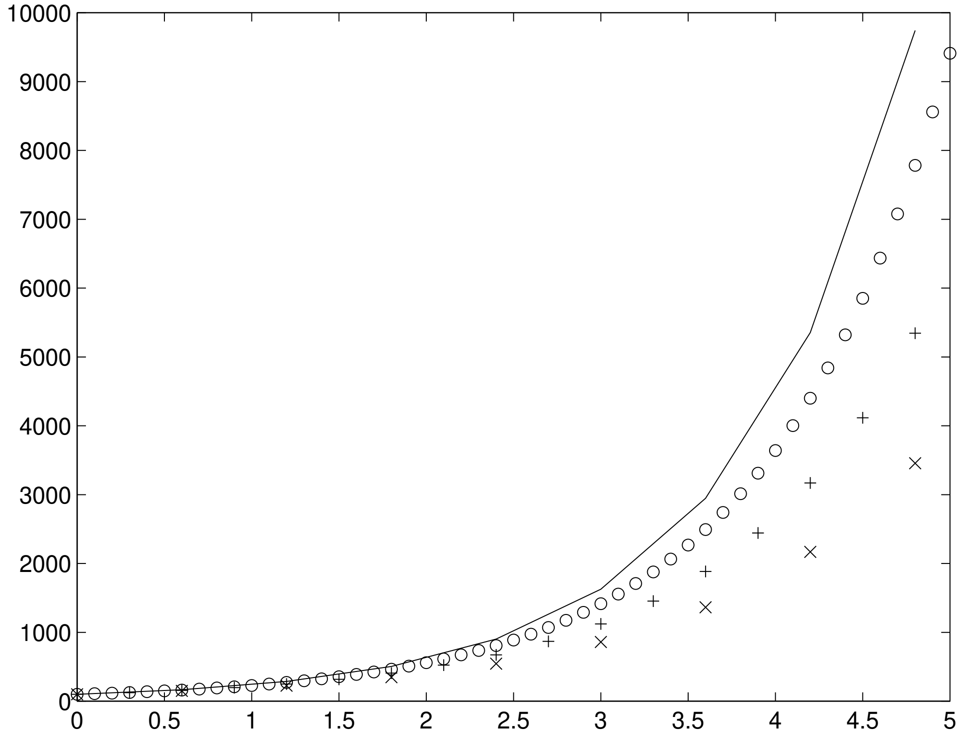

Figure 1.3: Solution of the initial value problem with $x = 100$ at $t = 0$. The exact solution is given by the solid line. Approximate solutions are shown for $h = 0.1, 0.3, 0.6$.

Since $f(x,t) = Kx$ our Euler method is

$$x_{i+1} = x_i + hKx_i. \tag{1.16}$$

If we use 1.16 repeatedly and assume no errors, we will obtain

$$x_{i+1} = (1 + hK)^{i+1} x_0. \tag{1.17}$$

For small enough values of the step size h, we can show that this equation converges to the true solution. Let's now perturb our initial value x_0 by some value e_0 such that $x^p = x_0 - e_0$. Substitution into equation 1.17 yields

$$x^p_{i+1} = (1 + hK)^{i+1} x^p = x_{i+1} - (1 + hK)^{i+1} e_0. \tag{1.18}$$

Therefore, the initial error e_0 will be magnified if $\|1 + hK\| \geq 1$. This means that the errors will accumlulate and may dominate the solution. This is the condition of *instability*. If, on the other hand, $\|1 + hK\| < 1$, then the error dies away and the method is said to be *absolutely stable*. Rearranging the stability condition leads to

$$-2 < hK < 0 \tag{1.19}$$

Notice that the Euler method is not absolutely stable for any positive value of K. This may be too demanding for our particular application, so we might be content to accept the fact that the error does not increase directly proportional to the x values. This is the condition of *relative stability.*

Equation 1.19 can be generalized as

$$-2 < h\frac{\partial f}{\partial x} < 0 \tag{1.20}$$

since $h > 0$, this implies that the quantity $\frac{\partial f}{\partial x}$ must be negative for absolute stability.

Figures 1.4 and 1.5 show graphs of absolute error, that is $x_{true} - x_{euler}$. As you can see, even at a fairly small step size, the error is increasing rapidly.

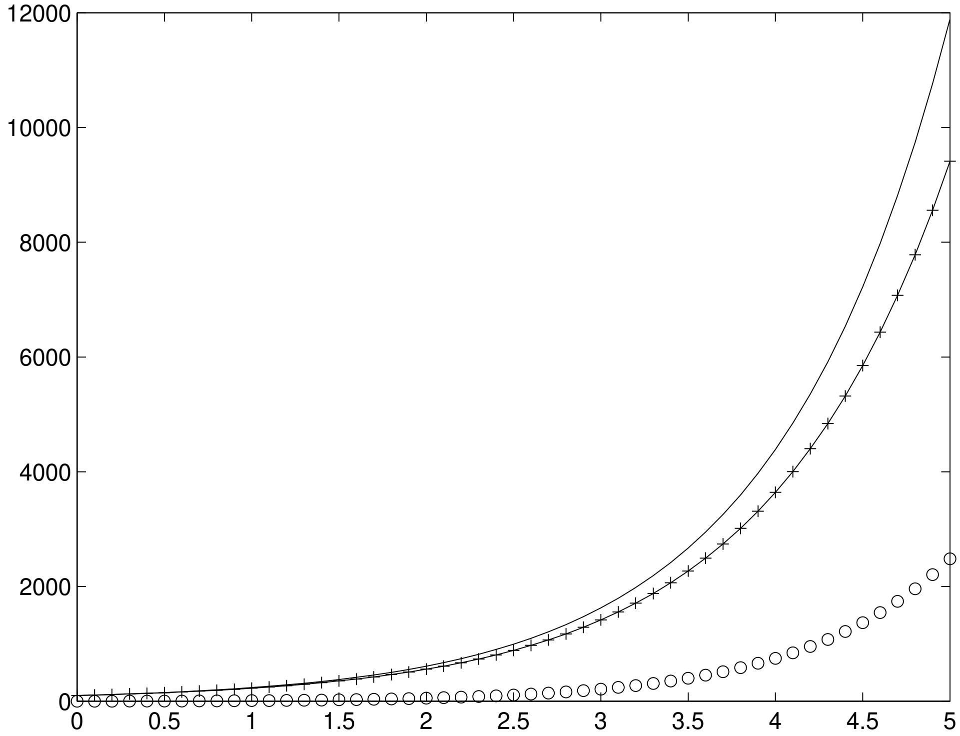

Figure 1.4: Solution of the initial value problem with $x = 100$ at $t = 0$. The exact solution is given by the solid line. The approximate solution at $h = 0.1$ is shown as '+' and the absolute error is shown as 'o'.

So now I've demonstrated how poor an approximation method the Euler method can be for certain problems. A reasonable question would be, "is there something better?" Well, if there were not anything better, there wouldn't be much of a point of continuing this course. There are a large number of variants upon Euler's method as well as several other methods that are higher order and have great stability properties. Here's an example of the Euler-trapezoidal method. The method can be expressed formally as:

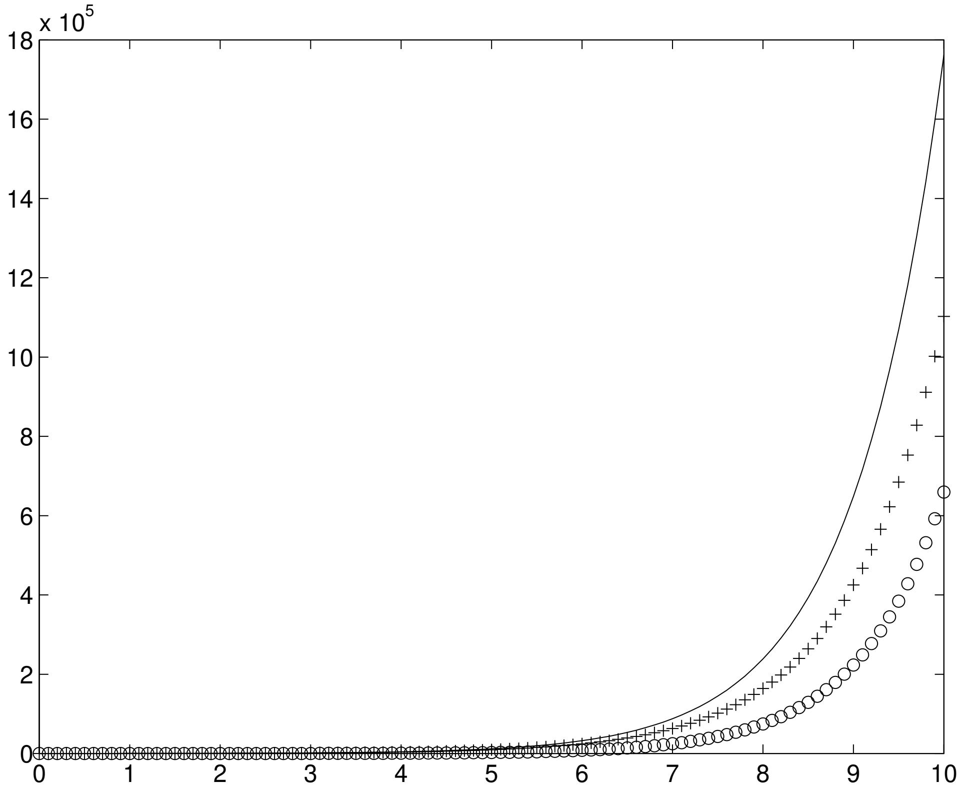

Figure 1.5: Solution of the initial value problem with $x = 100$ at $t = 0$. The exact solution is given by the solid line. The approximate solution at $h = 0.1$ is shown as '+' and the absolute error is shown as 'o'.

1. Start with n set at zero where n indicates the number of steps taken.

2. Calculate $x_{i+1}^{(1)} = x_i + hf(x_i, t_i)$.

3. Calculate $f(x^{(1)_{i+1},t_i})$ where $t_{i+1} = t_i + h$.

4. For $k = 1, 2, \ldots,$ calculate $x_{i+1}^{(k+1)} = x_i + hf(x_{i+1}^{(k)}, t_{i+1}) + f(x_i, t_i)/2$

At step 4, when the difference between successive values of x_{i+1} is sufficiently small (less than 0.05, for example), increment n by 1 and repeat steps 2, 3, and 4.

If we apply a similar error analysis to the Euler-trapezoidal method as well did to the Euler method, we find that the influence of the error term that involves e_0 will die away if its multiplier is less than unity in magnitude, i.e.

$$|(1 + \frac{hK}{2})/(1 - \frac{hK}{2})| < 1. \tag{1.21}$$

If K is negative for all h, the method is absolutely stable. For positive K, it is not absolutely stable for any h. Figure 1.6 shows that the Euler-trapezoidal method converges much more rapidly than the Euler method. Note that even the $h = 0.6$ solution gives a good result. Figure 1.7 shows the abolute error for the Euler-trapezoidal method. Note the difference

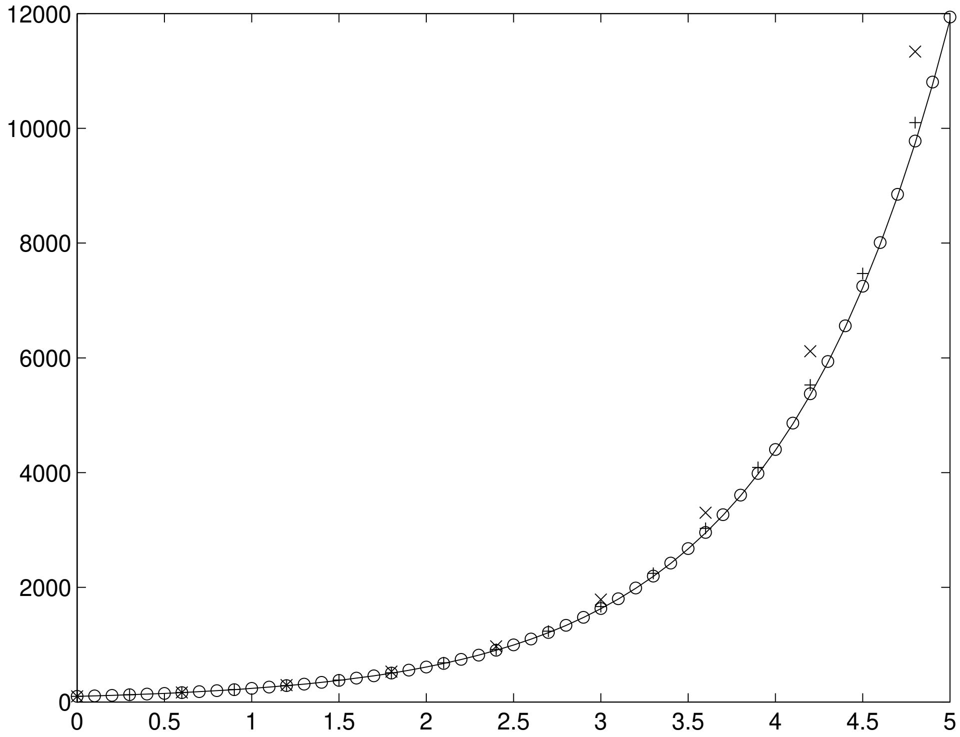

Figure 1.6: Solution of the initial value problem with $x = 100$ at $t = 0$ using the Euler-trapezoidal method. The exact solution is given by the solid line. Approximate solutions are given by $h = .6$ shown as 'x', $h = .3$ shown as '+', and $h = .1$ shown as 'o'.

between 1.7 and Figures 1.4 and 1.5. The Euler-trapezoidal method is clearly more accurate than the Euler method. I note that the Euler-trapezoidal method has a local truncation error of order h^3, while the Euler method has a local truncation error of h^2. Another consideration, however, is that of efficiency, i.e. how much time does it take to obtain a particular level of accuracy. For the Euler-trapezoidal method, one can focus on step (4) in the algorithm. If a large number of iterations are needed, then the method is fairly slow. For certain problems this is the case, however, for our example problem, we only needed two iterations.

While, the Euler-trapezoidal method was a higher order method than the simplist Euler method, there are still many other methods with even higher order. In the Euler method we extrapolate by using only a single smapling of the whole slope function and this gave us an order one method. In the Runge-Kutta methods, we sample the function at several (well chosen) points and use the information to contruct the slope function. The Runge-Kutta methods are fourth-order and have a local truncation error of order h^5 and a global error of order h^4.

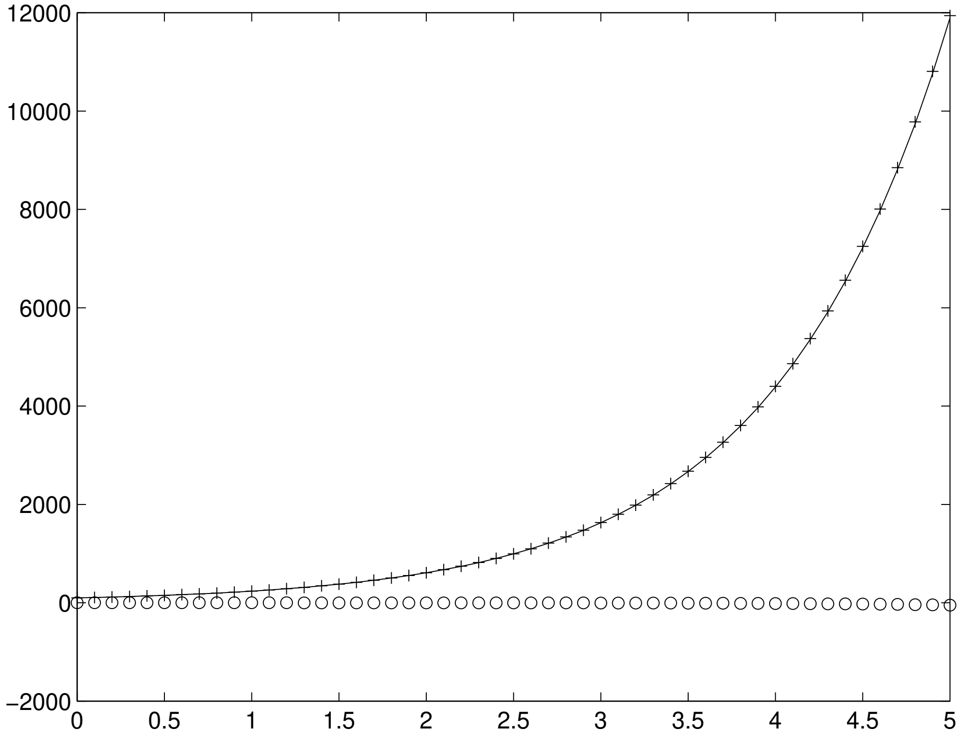

Figure 1.7: Solution of the initial value problem with $x = 100$ at $t = 0$ using the Euler-trapezoidal method. The exact solution is given by the solid line. The approximate solution at $h = 0.1$ is shown as '+' and the absolute error is shown as 'o'.

The Euler method evaluated f once per step, let's sample f twice per step and see what happens. First we make sure that the second evaluation depends on the first:

$$k_1 = hf(t_n, y_n) \tag{1.22}$$

$$k_2 = hf(t_n + \alpha h, y_n + \beta k_1) \tag{1.23}$$

$$y_{n+1} = y_n + \alpha k_1 + bk_2 \tag{1.24}$$

The goal is to choose the parameters α, β, a, and b so that the local truncation error is $O(h^3)$. Expanding $y(t_n + 1)$ in a Taylor's series expansion, we have:

$$y(t_n + 1) = y(t_n) + y^{(1)}(t_n)h + y^{(2)}(t_n)\frac{h^2}{2} + O(h^3) \tag{1.25}$$

Since we have that

$$y^{(1)}(t_n) = f(t_n, y_n) \tag{1.26}$$

$$y^{(2)}(t_n) = \frac{\partial f(t_n, y_n)}{\partial t} + \frac{\partial f(t_n, y_n)}{\partial y} f(t_n, y_n) \tag{1.27}$$

with

$$f = f(t_n, y_n) \tag{1.28}$$

$$f_t = \frac{\partial f(t_n, y_n)}{\partial t} \tag{1.29}$$

$$f_y = \frac{\partial f(t_n, y_n)}{\partial y} \tag{1.30}$$

we have

$$y(t_{n+1}) = y(t_n) = fh + (f_t + f_y h)\frac{h^2}{2} + O(h^3) \tag{1.31}$$

If we observe that

$$k_2 = hf(t_n + \alpha h, y_n + \beta k_1) = h(f + \alpha h f_t + \beta k_1 f_y + O(h^3)) \tag{1.32}$$

and so

$$y_{n+1} = y_n + ak_1 + bk_2 = y_n + (a+b)fh + b(\alpha h f_t + \beta f f_y)h^2 + O(h^3) \tag{1.33}$$

For the local truncation error to be $O(h^3)$ the following equation much hold:

$$y(t_{n+1}) - y_{n+1} = O(h^3). \tag{1.34}$$

Therefore we compare terms in equations (1.31 - 1.33) and require that:

$$\begin{aligned} a + b &= 1 \\ 2ba &= 1 \\ 2b\beta &= 1 \end{aligned} \tag{1.35}$$

As you will note, there are an infinite number of solutions to this system. Therefore we choose a simple combintation of $a = b = 1/2$ and $\alpha = \beta = 1$. This gives us the requirement that the local truncation error is $O(h^3)$ and we obtain the second-order Runge-Kutta method:

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + h, y_n + k_1) \\ y_{n+1} &= y_n + (k_1 + k_2)/2 \end{aligned} \tag{1.36}$$

Probably the most famous (and most used) Runge-Kutta method is the classical fourth-order method. For this method, f is sampled four times per step. The method is derived using a Taylor's series expansion is an analagous way as for the second-order RK method. The fourth-order RK method is:

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + h/2, y_n + k_1/2) \\ k_3 &= hf(t_n + h/2, y_n + k_2/2) \\ k_4 &= hf(t_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned} \tag{1.37}$$

1.2.2 Pond Example

Certainly many (perhaps most?) science and engineering problems are expressible as continuous sets of ordinary and/or partial differential equations. On the other hand, we may run into problems which are not amenable to conventional analytical or numerical methods. Such models are usually approached as having a

- feasible solution
- optimal solution
- satisfactory solution
- no solution.

Often, discrete models are used to find "satisfactory solutions" to practical problems.

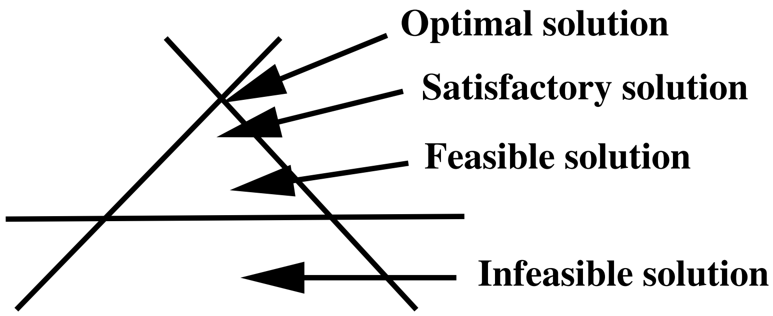

Figure 1.8: Solution space for a discrete system.

Let's assume that our class took a literal *field trip*. The particular field we end up in has a pond within its boundary (let's also assume that we know the boundary of our field to be a 1 acre rectangle). As this is a class on solving scientific and engineering problems, I ask the following question, "what is the area of the pond?" Some students start scurrying to measure the circumference of the pond, while others try other approximation methods. One unusually bright student turns his back to the pond and starts throwing stones in a random fashion (we must emphasize that he must throw the rocks in a truely random way) such that they land within the boundary of the field, either in the pond or not. After tabulating the number of *splashes* he hears versus the total number of stones he has thrown, he comes up with an accurate estimate of the area of the pond. How did he do it? By a discrete Monte Carlo simulation, of course...

More explicitly, let a rectangle be of height H and width $(b-a)$. Let a function $f(x)$ be confined within the rectangle's boundary. To measure the area under the curve, i.e. to evaluate the integral of $f(x)$, we can do the following:

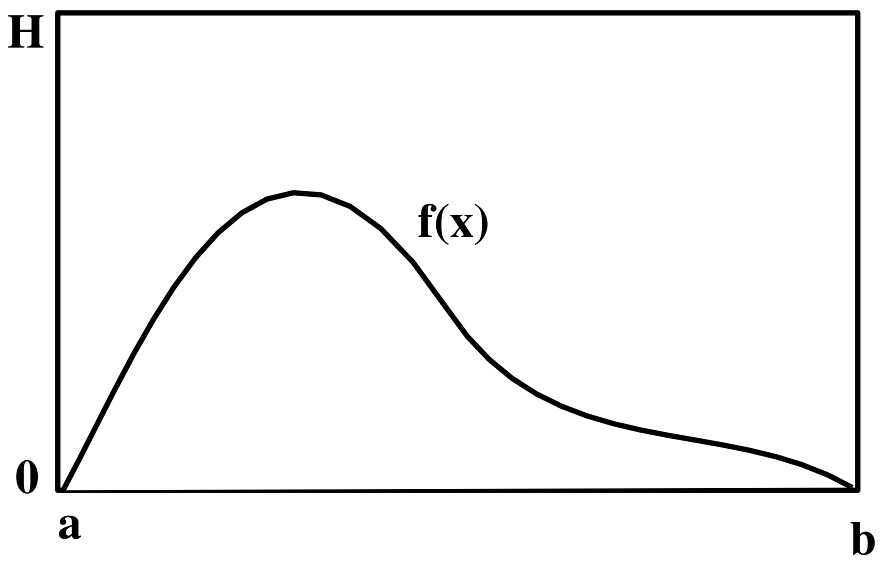

Figure 1.9: Integration of a function $f(x)$ by the Monte Carlo method.

- compute pairs of random numbers x_i and y_i such that $a \leq x_i \leq b$ and $0 \leq y_i \leq H$
- count the fraction of points x_i, y_i which satisfy the condition $y_i \leq f(x_i)$
- an estimate of the integral of $f(x)$ is then made by multiplying the area of the rectangle by the fraction of the points which satisfy the previous condition

As applied to the pond problem, we would estimate the area of the pond as approximately equaling $A\frac{n_s}{n}$, where A is the area of the land, n_s represents the number of splashes and n is the total number of rocks we threw within the boundaries of the land. Our estimate will improve with the more rocks we throw. Seems almost unbelieveable doesn't it? But it works - prove it to yourself by completing the assignment. Monte Carlo simulation techniques are powerful methods for solving multi-dimensional integrals. I note that one area of active research involves investigating the use of *sub-random* sequences rather than standard congruential random number generators to speed up convergence. We'll have more to say about this subject later. For those that can't wait, see the reference regarding Sobol sequences [10, 11, 12].