Notes - Numerical Methods

Numerical methods for simulation and data analysis are widely used in both systems and synthetic biology. In this part of the course you will investigate three different numerical methods:

1 . Solving Differential Equations

2. Solving nonlinear equations

3. Fitting data to models

Methods for solving stochastic models will be dealt with in a separate assignment.

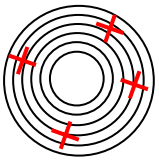
1. Basic Concepts in Numerical Analysis

All numerical methods involve approximations due to either limits in the algorithm or physical limits in the computer hardware. Errors associated with measurements or calculations can be characterized with reference to **accuracy** and **precision**.

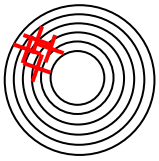
Accuracy refers to how closely a computed or measured value agree with the **true** value.

Precision refers to how close measured or computed values agree with each other after repeated sampling.

Nether Precise NOR accurate:



Precise, but NOT accurate:



Accurate but NOT precise:

Precise AND accurate

Simple Error Definitions

There are a number of ways to describe errors in measurements and calculations. The simplest is the **absolute error**, this is the difference between the measured or calculated value and the true value.

$$\epsilon = |True\ Value - Approximation|$$

A shortcoming of the absolute error is that it doesn't take into account the order of magnitude of the value under consideration. One way to account for the magnitude is to consider instead the **relative error**.

$$\epsilon = \frac{|True\ Value - Approximation|}{|True\ Value|} \times 100$$

Many numerical methods are interative, that is they involve repeating the same calculation many times. In terms of error analysis, two types of error emerge, **local** and **global** errors. The local error is the error introduced during one operation of the iterative process. The global error is the accumulative error over many iterations. Note that the global error is not simply the sum of the local errors due to the nonlinear nature of many problems although often it is assumed to be so because of the difficulties in measuring the global error.

Sources of Errors in Numerical Calculations

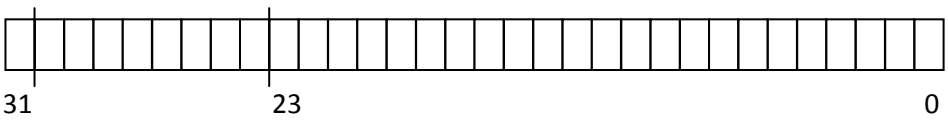
There are at least two sources of errors in numerical calculations:

1. Rounding Errors
2. Truncation Errors

Rounding errors originate from the fact that computers can only represent numbers using a fixed and limited number of significant figures. Thus, numbers such as π or $\sqrt{2}$ cannot be represented exactly in computer memory. The discrepancy introduced by this limitation is call **round-off error.** Even simple addition can result in round-off error. Often computers have the capacity to represent numbers in two different precisions, called **single** and **double** precision.

In **single precision**, 23 bits out of a total of 32 bits are used to represent the significant digits in the number. Of the remaining bits, 8 are used to store the exponent and one bit is used to store the sign.

Bit Organization in a Single Precision Number:



With 23 bits to represent the significant figures, a single precision number can represent data to about seven decimal places. The 8 bit exponent allows scaling in the range 10^{38} and 10^{-38}. Taken together, single precision numbers can range from $\pm 1.175494351 \times 10^{-38}$ to $\pm 3.4028235 \times 10^{38}$. In single precision, π can be represented as 3.141593.

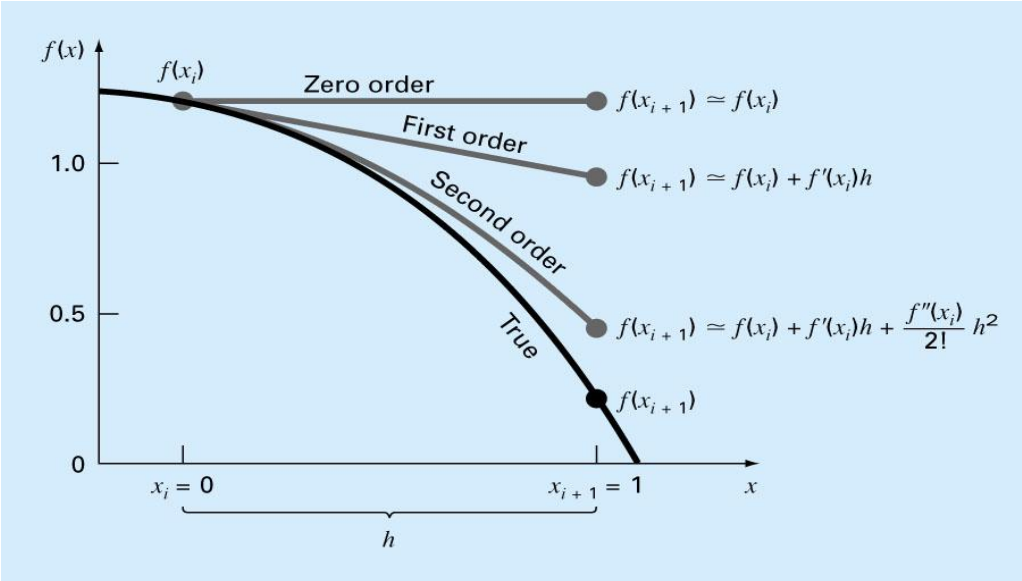
Due to the limited number of significant digits in single precision, most modern computers use **double precision** which uses **64 bits** with **52 digits** used to represent the significant figures. This allows π, for example, to be represented as 3.141592653589793, that is, 16 digits. The full range for number representation using double precision is $\pm 2.2250738585072020 \times 10^{-308}$ to $\pm 1.7976931348623157 \times 10^{308}$.

The use of double precision reduces the effects of rounding error and should be used whenever possible in numerical calculations.

Truncation errors in numerical analysis arise when approximations are used to estimate some quantity. Often a Taylor series is used to approximate a solution which is then truncated. The figure below shows a function $f(x_i)$ being approximated by a Taylor series that has been truncated at different levels. The more terms that are retained in the Taylor series the better the approximation and the smaller the truncation error.

Taylor Series:

$$f(x_{i+1}) = f(x_i) + f'(x_i)\,h + \frac{f''(x_i)}{2!}\,h^2 + \ldots$$



2. Solving Differential Equations Numerically

Consider the simple differential equation:

$$\frac{dy}{dt} = -20\,y$$

Such simple equations can be solved analytically, that is we can obtain an equation that describes how the variable y evolves in time, t. However, in most cases this is not possible; instead we must resort to solving the equation numerically.

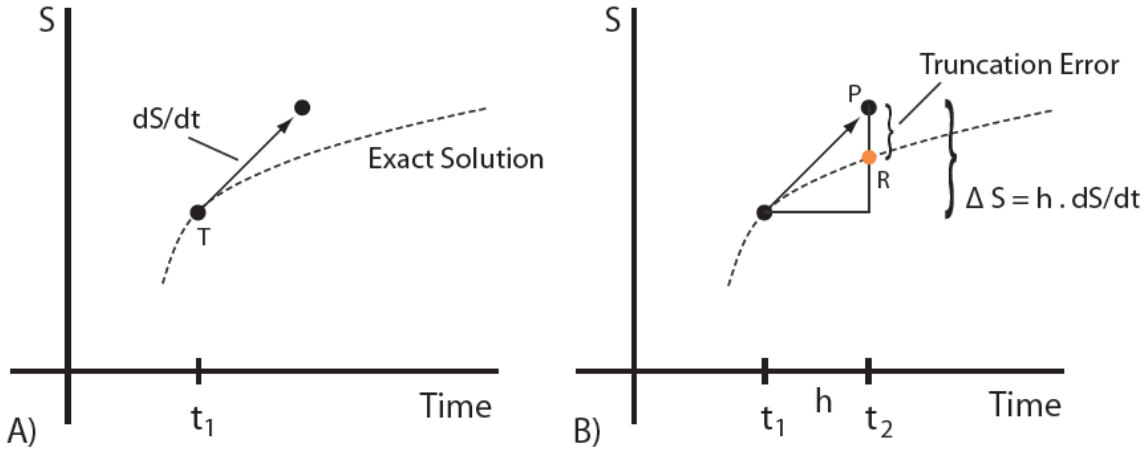
The simplest method for computing a solution to a differential equation is called the **Euler Method**. To use the Euler method we need two things, a starting point for the y variable, $y(0)$ and a step size h by which we increment time.

If $y_n$is the value of y at time t_n then the next value of y, y_{n+1}, is given by:

$$y_{n+1} = y_n + h\,\frac{dy_n}{dt}$$

$$t_{n+1} = t_n + h$$

The accuracy of the solution will depend on how small we make the step size, h. If you look at the figure below you will see that the Euler method projects the slope forward and uses it to estimate the next solution point. However, because the exact solution may be nonlinear, the projection will inevitably yield some error, called the truncation error. In panel B), one can see that the smaller we make h, the smaller the truncation error. However, a small h means it will take longer to traverse the solution and if h is too small we can begin to run into round-off errors.



If we have more than one differential equation the pseudo code shown below can be used:

Algorithm 1 Euler Integration Method

$n = \text{Number of State Variables}$
$\text{timeEnd} = 10$
$\text{currentTime} = 0$
$h = \text{stepSize}$
while $t < \text{timeEnd}$ **do**
  for all n **do**
    $k_i = f(y_i)$

  for all n **do**
    $y_i(t+h) = y_i(t) + h\ k_i$

  $\text{currentTime} = \text{currentTime} + h$
end while

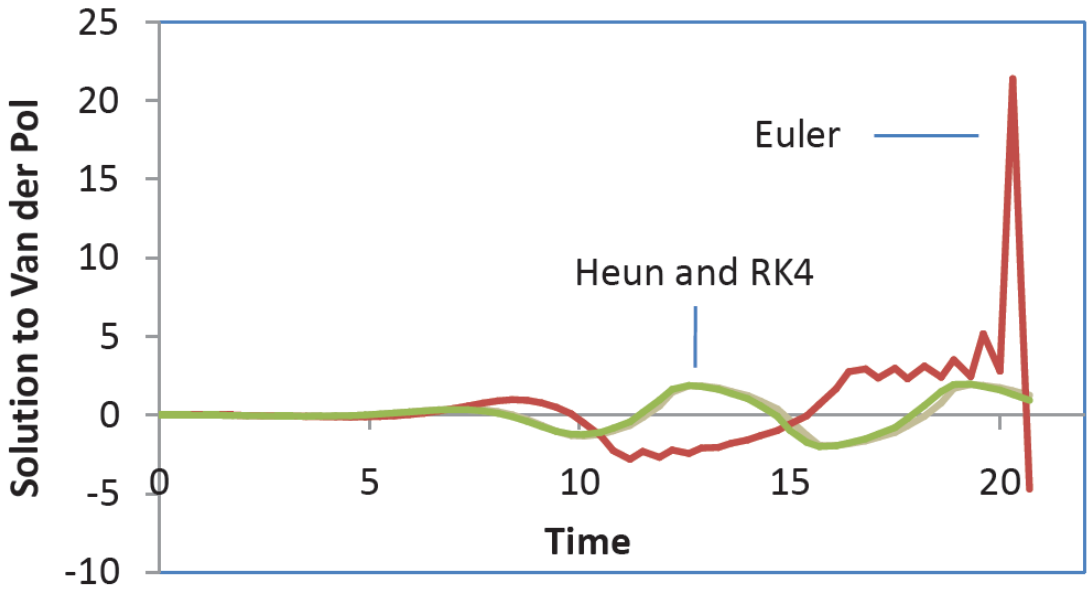
The Euler method, being the simplest, is also the worse method for solving a set of differential equations. For simple problems the approach is quick to implement and will usually generate reasonable solutions, even if the step size has to be made very small.

Due to instability problems, the Euler method is however not often used. Instead a variety of other methods are employed. One of the commonest alternative approaches is the Runge-Kutta 4th order method (RK4). The RK4 method takes a series of slopes from which it constructs a weighted average.

$$k_1 = h\ f(y_n)$$
$$k_2 = h\ f\left(y_n + \frac{k_1}{2}\right)$$
$$k_3 = h\ f\left(y_n + \frac{k_2}{2}\right)$$
$$k_4 = h\ f\left(y_n + k_3\right)$$
$$y(t+h) = y(t) + \frac{1}{6}\left(k_1 + 2\ k_2 + 2\ k_3 + k_4\right)$$
$$t_{n+1} = t_n + h$$

4th Order Runge-Kutta

The figure below compares the simulation results from three different methods, Euler, Heun and RK4.



Comparison of Euler, Heun and RK4 numerical methods at integrating the Van der Pol dynamic system: $dy_1/dt = y_2$ and $dy_2/dt = -y_1 + (1 - y_1 y_1)y_2$. The plots show the evolution of y_1 in time. The RK4 solution is indistinguishable from solutions generated by the best integrators. Step size was set to 0.35.

Stiff Problems

One class of problem, terms stiff problems, is characterized by widely separated time scales. For example one part of a model might be dominated by very fast reactions, while another by slow reactions. In such situations, a numerical integrator must ensure that the step size is small enough to capture the fast dynamics. This results in extremely slow integration times and often failure to integrate the solutions at all. As a result, numerical analysts have developed sophisticated methods to deal with stiff problems. Computer libraries are available that are specifically designed to solve stiff problems, the most common libraries include the **LSODA** series (ODEPACK) and **CVODE**. Most modern software tools use one or both of these to solve systems of differential equations.