Everything you need to know about Initial Value Problem Solvers

Fernando Alonso-Marroquin
(http://physics.uq.edu.au/people/fernando)
email:fernando@esscc.uq.edu.au

September 23, 2008

1 What is an initial value problem?

An initial value problem (IVP) is a differential equation

$$y'(t) = f(t, y(t)) \quad \text{with} \quad f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n \tag{1}$$

together with the initial condition

$$y(t_0) = y_0 \tag{2}$$

Initial value problems include high order differential equations. The derivatives of second or higher order are considered elements of the vector y.

2 Does the solution of the IVP exist?

2.1 Definition: Lipschitz Function

A real valued function $f\colon D \subseteq \mathbb{R} \to \mathbb{R}$ is called *Lipschitz continuous* or is said to satisfy a *Lipschitz condition* if there exists a constant $K \geq 0$ such that for all x_1, x_2 in D

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2| \tag{3}$$

The inequality is (trivially) satisfied if $x_1 = x_2$. Otherwise, for $x_1 \neq x_2$, one can equivalently define a function to be *Lipschitz* if and only if $\frac{|f(x_1)-f(x_2)|}{|x_1-x_2|} \leq K$ that is, iff the slopes of all secant lines are bounded.

2.2 Picard - Lindelöf Theorem

Consider the initial value problem

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0, \quad t \in [t_0 - \alpha, t_0 + \alpha]. \tag{4}$$

Suppose f is bounded,Lipschitz continuous in y, and continuous in t. Then, for some value $\epsilon > 0$, there exists a unique solution $y(t)$ to the initial value problem within the range $[t_0 - \epsilon, t_0 + \epsilon]$.

3 What is an IVP solver?

A Solver for the initial value problem is an approximation of $y(t)$ given by

$$w_{n+k} = \Psi(t_{n+k-1}; w_n, w_{n+1}, \ldots, w_{n+k-1}; h). \tag{5}$$

where w_n represents and approximation of $y_n = y(t_0 + nh)$, h is the step. A particular case is the one-step solver

$$w_{n+1} = \Psi(t_n, w_n, h). \tag{6}$$

3.1 Euler Method

The Euler methods (Euler, 1768) is based on the approximation for the derivative of $y(t)$:

$$y'(t) = \frac{y(t+h) - y(t)}{h} + O(h). \tag{7}$$

Replacing Eq. (1) into this equation leads to:

$$y(t+h) = y(t) + hf(t, y) + O(h^2) \tag{8}$$

This equation allows to approximate y in $t + h$ in therm of its value in t:

$$w_{n+1} = w_n + hf(t_n, w_n); \quad w_1 = y_0; \tag{9}$$

3.2 Backward Euler

The backward Euler method, instead of 7, we use the approximation for $y'(t + h)$:

$$y'(t+h) = \frac{y(t+h) - y(t)}{h} + O(h). \tag{10}$$

Replacing Eq. (1) into this equation leads to:

$$y(t+h) = y(t) + hf(t+h, y(t+h)) + O(h^2) \tag{11}$$

From this Equation we get the following approximate solution for the IVP:

$$w_{n+1} = w_n + hf(t_{n+1}, w_{n+1}); \quad w_1 = y_0; \tag{12}$$

The trapezoidal method, that is, is higher order implicit method, which is obtained from the Taylor Series approximation

$$y(t+h) = y(t) + \frac{1}{2}h\left(f(t, y) + f(t+h, y+h)\right) + 0(h^3), \tag{13}$$

which leads to the implicit relationship

$$w_{n+1} = w_n + \frac{1}{2}h\left(f(t_n, w_n) + f(t_{n+1}, w_{n+1})\right); \quad w_1 = y_0; \tag{14}$$

Applying this method instead of Euler's method leads to more accurate results. The backward Euler method and the trapezoid methods are implicit method, meaning that we have to solve an equation to find w_{n+1}. One can uses the fixed point iteration of the Newton-Raphson method to solve this equation. Why do we have to do such complications? We will find the answer to this question when we introduce the stiff equations.

3.3 Heun's method

This method is also known as explicit trapezoid method. It corresponds to an improvement of the Euler equations

$$\begin{aligned} k_1 &= f(t_n, w_n) \qquad (15)\\ k_2 &= f(t_n + h, w_n + hk_1)\\ w_{n+1} &= w_n + \frac{h}{2}(k_1 + k_2) \qquad (16)\\ w_1 &= y_0 \end{aligned}$$

This relation is obtained from by replacing Eq. (9) in the right side of Eq. (14).

3.4 Runge-Kutta methods

The *Runge - Kutta* methods correspond to a family of one-step solvers. The basic idea of the Runge-Kutta method is find a discrete expression to integrate $y(t)$ from t to $t+h$. This integral is estimated as the average of $f(t, y)$ in a certain amount of points in the interval $[t, t + h]$. Explicit Runge-Kutta methods follow from the approximation:

$$y(t+h) = y(t) + \int_t^{t+h} f(t, y(t))dt = h\sum_{i=1}^{p} b_i k_i + O(h^{p+1}) \qquad (17)$$

$$k_i = f\left(t + c_i h, y(t) + h\sum_{j=1}^{p} a_{ij}k_j\right). \qquad (18)$$

Where b_i, c_i and a_{ij} are chosen to satisfy (17). The most popular of this family is the four order Runge-Kutta method:

$$\begin{aligned} k_1 &= f(t_n, w_n) \qquad (19)\\ k_2 &= f\left(t_n + \frac{h}{2}, w_n + \frac{h}{2}k_1\right)\\ k_3 &= f\left(t_n + \frac{h}{2}, w_n + \frac{h}{2}k_2\right)\\ k_4 &= f(t_n + h, w_n + hk_3) \qquad (20)\\ w_{n+1} &= w_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)\\ w_1 &= y_0. \end{aligned}$$

This method is derived using the Simpson's Integration rule. In general, the order p Runge-kutta methods require p calculations of $f(t, y)$ per step, the truncation error (defined below) is of order $p + 1$ and the global error (defined below) is the order p.

3.5 Multi-step methods

Another possibility to improve the Euler method is to use not only the previously computed value to determine the new one, but to make the solution depend on more past values. This

is possible by using numerical differentiation formulas. These formulas lead to the so-called *multi-step methods*.
The simplest two step method is obtained from the centered difference formula:

$$y'(t) = \frac{y(t+h) - y(t-h)}{2h} + O(h^2). \tag{21}$$

which leads to

$$y(t+h) = y(t-h) + 2hf(t,y) + 0(h^3) \tag{22}$$

From this equation we can derive the second order, multi-step method

$$\begin{aligned} w_{n+1} &= w_{n-1} + 2hf(t_n, w_n) && (23)\\ w_0 &= y_0 \\ w_1 &= y_0 + hf(t_0, y_0) && (24) \end{aligned}$$

This solver, however exhibes instabilities, as it leads for some IVP's to a uncontrollable growth of error. Stable multi-step methods need to be implicit, or semi-implicit, which means that the implicit term is resolved by guessing the value for w_{n+1} from an explicit solver.
For example, the explicit Adam-Bashforth formula

$$w_{n+1} = w_n + h\left(\tfrac{3}{2}f(t_n, w_n) - \tfrac{1}{2}f(t_{n-1}, w_{n-1})\right) \tag{25}$$

can be used to *guess* a value which is then used into a implicit method, like for example, the implicit trapezoid method. This leads to an stable multi-step method, often called predictor-corrector (or PECE = Predictor-Evaluation-Corrector-Evaluation) method, which is known as-the Adams-Bashforth-Moulton method:

$$\begin{aligned} w^*_{n+1} &= w_n + h\left(\tfrac{3}{2}f(t_n, w_n) - \tfrac{1}{2}f(t_{n-1}, w_{n-1})\right) && (26)\\ w_{n+1} &= w_n + \tfrac{1}{2}h\big(f(t_{n+1}, w^*_{n+1}) + f(t_n, y_n)\big) && (27) \end{aligned}$$

Since this is a two-steps method, the second term w_2 is calculated using a one-step method. After the second step, the explicit predictor formula is first used to compute a tentative value. It is substituted in the right hand side of the implicit corrector formula and the formula evaluated to get a better approximation to w_{n+1}. This process is repeated until the implicit formula is satisfied well enough or the step is deemed a failure. If the step is a failure, is reduced to improve the rate of convergence and the accuracy of the prediction, and the program again tries to take a step. Simple iteration is the standard way of evaluating implicit formulas for the non-stiff problems introduced bellow. If a predetermined, fixed number of iterations is done, the resulting method is called a predictor-corrector formula.

3.6 Multi-derivative methods

Some classes of alternative methods are the multi-derivative method , which calculates not only $y(t)$ but but also its derivatives until a certain order. This class include the so-called *Gear's predictor-corrector methods*. These algorithms constitute a commonly used class of methods in

problems where the calculation of $f(t, y)$ is expensive. Expensive force calculations are often in molecular dynamics (i.e. simulations of many particles interacting). The Gear predictor-corrector methods consists of three steps:

- **Predictor**. From the state variables $y(t)$ and their time derivatives up to a certain order p, all known at time t, one *predicts* the same quantities at time by means of a Taylor expansion. Among these quantities are, of course, the derivative $y'_p(t)$.
- **Force evaluation**. $f(t, y_p)$ is computed in terms of the predicted values. The resulting values will be in general different from the *predicted* one $y'_p(t)$. The difference between the two constitutes an *error signal*.
- **Corrector**. This error signal is used to *correct* state $y(t)$ and their derivatives. All the corrections are proportional to the error signal, the coefficient of proportionality being a *magic number* determined to maximize the stability of the algorithm.

Details of these algorithms can be found in *M.P. Allen and D.J. Tildesley. Computer Simulation of Liquids. Oxford University Press, New York, 1989.*

3.7 Step-variable solvers

One of the advantages of the Runge-Kutta Methods is the possibility to extend them to the adaptative step solver. They consists of two embedded methods, one of high order than the other. They are call embedded because the one of the highest order used the calculation of the lowest one. The embedded Runge-Kutta pairs provide a control of the global error by evaluating according to an imposed tolerance T, (where $0 < T < 1$). In each iteration the step is modified by the following procedure:

- Calculate the solution w_{n+1} and w^*_{n+1} using the two methods.
- Estimate the discrepancy between both approximations as $E = |w_{n+1} - w^*_{n+1}|/|w^*_{n+1}|$
- if the discrepancy is too large ($E < T$), then reduce the time step and recalculate w_{n+1} and w^*_{n+1}.
- if the discrepancy is one significant digit lower than the tolerance $E < 0.1T$, then time step is increased and it is used for the next iteration.
- if the discrepancy is moderately small ($0.1T < E < T$), the step is used for the next iteration.

Matlab includes the solver *ode23*, which corresponds to order 2/order 3 embedded Runge-Kutta pairs. The matlab command *ode45* used an order 4/order 5 embedded pair. More details of these solver can be found in Section 8.

4 Is the solver convergent?

The solver of the initial value problem is said to be convergent if the numerical solution approaches the exact solution as the step size h goes to 0. More precisely, we require that for every ordinary differential equation with a Lipschitz function f and every $t > 0$,

$$\lim_{h \to 0+} \max_{n=0,1,\ldots,\lfloor t/h \rfloor} \|w_n - y_n\| = 0. \tag{28}$$

The convergence is the first condition a solver must satisfy.

5 How accurate is the solver?

The precision of the solver is measured from the error of the numerical solution. The *local error* of the method is the error committed by one step of the method. That is, it is the difference between the result given by the method, assuming that no error was made in earlier steps, and the exact solution:

$$e_{n+k}(h) = |\Psi(t_{n+k-1}; y(t_n), y(t_{n+1}), \ldots, y(t_{n+k-1}); h) - y(t_{n+k})|. \tag{29}$$

In the special case of one step solvers, the local error is given by;

$$e_{n+1}(h) = |\Psi(t_n; y(t_n), h) - y(t_n + h)|. \tag{30}$$

The method has *order* p if $e_{n+k}(h) = O(h^{p+1})$ as $h \to 0$. The method is *consistent* if it has an order greater than 0. The Euler method had order 1, so it is consistent. Most methods being used in practice attain higher order. Consistency is a necessary condition for convergence.
The local error differs from the *global error*, which is the accumulated error until a time $t = to{+}nh$

$$e(t, h) = |w_n(h) - y(t)| \tag{31}$$

The global error of a p order one-step method is $O(h^p)$; in particular, such a method is convergent. This statement is not necessarily true for multi-step methods.

6 Is the solver stable?

When apply a numerical method to initial value problems, Some calculations might damp out approximation errors that occur; others might magnify such errors. Calculations that do not magnify approximation errors are called numerically stable. One desirable property of solver is that is be *robust*. That means, it exhibits numerical stability for a wide range of initial value solver. For some differential equations, application Euler method, or explicit Runge-Kutta methods, or multi-step methods (e.g., Adams-Bashforth methods) exhibit instability in the solutions, though other methods may produce stable solutions. This "difficult behavior" in the equation (which may not necessarily be complex itself) is described as stiffness, and is often caused by the presence of different time scales in the underlying problem.
A differential equation is called stiff when its numerical solution becomes instable unless the step size is taken extremely small. There is no a precise definition of stiffness, but the main idea is that the equation includes some terms that can lead to rapid variation in the solution in certain time intervals.

6.1 Example

Consider the initial value problem $y'(t) = \lambda y(t)$, where $y(0) = 1$. The exact solution is $y(t) = e^{\lambda t}$, and clearly for $\lambda < 0$ $y(t) \to 0$ as $t \to \infty$. We want the numerical solutions to exhibit the same behavior. However, For various numerical integrators applied on the equation we do not have this condition. For Example, the explicit Euler method applied to this problem leads to:

$$w_{n+1} = (1 + h\lambda)w_n \tag{32}$$

First, Euler's method with $\lambda = -15$ a step size of $h = 0.25$ oscillates wildly and quickly become much larger that the actual solution. After halving the step size and re-running the integration with $h = 0.125$, the resulting solution oscillates about zero, and by no means represents the exact solution. In general, the numerical solution will be stable only when $|1 + h\lambda| < 1$, or $0 < h < -2/\lambda$, so that as $|\lambda|$ is large h must be taken extremely small.
Applying implicit methods gives a much better results for this kind of problem. The simplest implicit method, the backward euler, leads to

$$w_{n+1} = w_n + hf(t_{n+1}, w_{n+1}) \tag{33}$$

In general, this is aa non-linear equation must be solved for w_{n+1} using either fixed point interaction or Newton-Raphson Method. But for our example $y'(t) = \lambda y(t)$ we get the following recursion :

$$w_{n+1} = \frac{1}{1 - h\lambda} w_n \tag{34}$$

The solution will always decay for any $\lambda < 0$. So, the solution may be inaccurate, but it will never blow up. In fact, if h is very large, the solution will be damped even more rapidly. That is, the method pushes the decaying solution prematurely towards its steady-state value of 0. Thus, backward Euler is unconditionally stable for any equation with decaying exponential solutions, whereas forward Euler is stable conditioned on restricting h.

6.2 Characterization of Stiff equations

Certain types of problems can be characterized as stiff:

- Problems of the form $y' = ky + f(t)$ where $|k|$ is large
- Systems of equations in the form $\mathbf{y}' = K\mathbf{y} + \mathbf{f}(t)$, where K is a the maximal eigenvalue of the square matrix K is large.
- Systems of the form $\mathbf{y}' = \mathbf{f}(\mathbf{y}, t)$ where the maximal eigenvalue of Jacobian $\partial\mathbf{f}/\partial\mathbf{y}$ is large.

Some initial value problems are not stiff initially. they only becomes stiff as the solution approaches steady state. For example, the differential equation for ignition problems $y' = y^2 - y^3$, with $y(0) = \epsilon$ and $0 < t < 2/\epsilon$, where ϵ is small, becomes *rigid* when the solution approach to the steady-state $y(t) = 1$. Near this value the solution increases or decreases rapidly toward that solution. We should point out that *rapidly* here is with respect to an unusually long time scale $(0 < t < 1/\epsilon)$.

6.3 A-stability

The behavior of numerical methods on stiff problems can be analyzed by applying these methods to the test equation

$$y' = \lambda y \quad y(0) = 1 \quad with \quad \lambda \in C. \tag{35}$$

The solution of this equation is $y(t) = e^{\lambda t}$. This solution approaches zero as $t \to \infty$ when $Re(\lambda) < 0$. If the numerical method also exhibits this behavior, then the method is said to be A-stable. A-stable methods do not exhibit the instability problems of stiff problems as described in the motivating example above.
Euler, Explicit Trapezoid, and Runge-Kutta methods are not A-stable. Implicit Euler and Implicit Trapezoid methods are A-Stable. Explicit multi-step methods can never be A-stable. Implicit multi-step methods can only be A-stable if their order is at most 2. The Adams-Bashforth-Moulton method is an example of a multi-step A-stable method.

7 What is the efficiency of the Solver?

Given two solvers, the most efficient one is the one that require less numerical operations to obtain the same global error. Euler method is less inefficient than the explicit trapezoid method because the step required to get accurate solutions is much lower. Runge-Kutta methods of order 3 or more are more efficient than the explicit trapezoid method. For problems where the calculation of $f(t, y)$ is computational expensive, a Runge-Kutta method of order p may become less efficient than multi-step or Gear's predictor-corrector methods of the same order. In those cases is preferred to store the values of f for the previous steps (or story high order derivatives of t)and to use them in a multi-step, or the Gear's predictor-corrector method. Step-variable solvers are in general more efficient, and safer, than the corresponding constant-step solvers.
For stiff problems, step-variable explicit solver may become inefficient as the time steps after the regimes of fast change may remain too small. In these cases it more efficient to use implicit or semi-implicit solvers. It is important to clarify that stiffness is an efficiency issue. If we weren't concerned with how much time a computation takes, we wouldn't be concerned about stiffness. Non-stiff methods (like ode45) can solve stiff problems; they just take a long time to do it.

8 In-build solvers in Matlab

Matlab contains several variable-step solvers. These solvers decrease the simulation step size to increase accuracy when a system's continuous states are changing rapidly and increase the step size to save simulation time when a system's states are changing slowly.

- ode45 is based on an explicit Runge-Kutta $4-5$ embedded formula, the Dormand-Prince pair. It is a one-step solver; that is, in computing w_n, it needs only the solution at the immediately preceding time point, w_{n-1}. In general, ode45 is the best solver to apply as a first try for most problems.

- ode23 is also based on an explicit Runge-Kutta 2-3 pair of Bogacki and Shampine. It can be more efficient than ode45 at crude tolerances and in the presence of mild stiffness. ode23 is a one-step solver.

- ode113 is a variable-order Adams-Bashforth-Moulton PECE solver. It can be more efficient than ode45 at stringent tolerances. ode113 is a multistep solver; that is, it normally needs the solutions at several preceding time points to compute the current solution.
- ode15s is a variable order solver based on the numerical differentiation formulas (NDFs). These are related to but are more efficient than the backward differentiation formulas, BDFs (also known as Gear's method). Like ode113, ode15s is a multistep method solver. If you suspect that a problem is stiff, or if ode45 failed or was very inefficient, try ode15s.
- ode23s is based on a modified Rosenbrock formula of order 2. This formula is a embedded semi-explicit 4-5 Runge-Kutta PECE Pair. Because it is a one-step solver, it can be more efficient than ode15s at crude tolerances. It can solve some kinds of stiff problems for which ode15s is not effective.

For more information about these solvers, see Shampine, L. F., Numerical Solution of Ordinary Differential Equations, Chapman & Hall, 1994.