Differential Equations — First Order Equations

Contents

1 Theory ......................................................... 2
  1.1 Mathematics ........................................... 2
  1.2 Numerics ............................................... 2
    1.2.1 Euler’s Method .................................... 2
    1.2.2 Example ........................................... 4
    1.2.3 Other Methods .................................... 7

2 Scilab ....................................................... 8
1 Theory

1.1 Mathematics

A first order differential equation is an equation of the form

\[ \frac{dy}{dt} = f(t, y). \] (1)

Once an initial value

\[ y(t_0) = y_0 \]

is specified, we have an initial value problem. An initial value problem has a unique solution \( y(t) \) giving \( y \) as a function of \( t \). Another way of saying the same thing is that for each point \( (t_0, y_0) \) of the \( t-y \) plane there is a unique solution of equation (1) passing through that point.

Geometrically, the differential equation can be interpreted as saying that the solution curve through a point \( (t, y) \) has slope \( f(t, y) \) at that point.

1.2 Numerics

1.2.1 Euler’s Method

The simplest numerical method for solving initial value problems for differential equations is Euler’s method. Suppose we have an initial value problem

\[ \frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0. \]
Choose a step-size $\Delta t$, then, by Taylor’s approximation

$$y(t_0 + \Delta t) \approx y(t_0) + \frac{dy}{dt}(t_0) \Delta t$$

The differential equation says $dy/dt = f(t, y)$ so that

$$y(t_0 + \Delta t) \approx y_0 + f(t_0, y_0) \Delta t$$

Denote the approximate solution obtained in this way by $\bar{y}(t)$, then Euler’s method can be written

$$\bar{y}(t_0 + \Delta t) = y_0 + f(t_0, y_0) \Delta t$$

Euler’s method has a simple geometric interpretation – we approximate the solution curve through a point $(t_0, y_0)$ by its tangent which we know has the slope $f(y_0, t_0)$.

We have seen how one step of Euler’s method works. The same procedure can be repeated over any number of steps: start with the initial condition

$$\bar{y}(t_0) = y_0$$
and then apply Euler’s method to obtain the approximations
\[
\begin{align*}
\tilde{y}(t_0 + \Delta t) &= \tilde{y}(t_0) + f(t_0, \tilde{y}(t_0)) \Delta t \\
\tilde{y}(t_0 + 2\Delta t) &= \tilde{y}(t_0 + \Delta t) + f(t_0 + \Delta t, \tilde{y}(t_0 + \Delta t)) \Delta t \\
\tilde{y}(t_0 + 3\Delta t) &= \tilde{y}(t_0 + 2\Delta t) + f(t_0 + 2\Delta t, \tilde{y}(t_0 + 2\Delta t)) \Delta t \\
\tilde{y}(t_0 + 4\Delta t) &= \tilde{y}(t_0 + 3\Delta t) + f(t_0 + 3\Delta t, \tilde{y}(t_0 + 3\Delta t)) \Delta t \\
&\vdots
\end{align*}
\]

1.2.2 Example

We will use as our example the differential equation
\[
\frac{dy}{dt} = -2ty
\]
This equation can be solved by separation of variables to give the general solution
\[
y(t) = Ae^{-t^2}
\]
The initial value problem
\[
\frac{dy}{dt} = -2ty, \quad y(0) = 1
\]
then has the solution
\[
y(t) = e^{-t^2}
\]
We will write a Scilab function euler(f, t0, y0, dt, n) to approximate the solution of
\[
\frac{dt}{dt} = f(t, y)
\]
using Euler’s method with initial conditions \(t = t0, y = y0\), and time-step dt over n steps.

function [t,y] = euler(f, t0, y0, dt, n)
  t = zeros(n+1,1)
  y = zeros(n+1,1)
  t(1) = t0
  y(1) = y0
  for i = 1:n
\[ t(i+1) = t(i) + dt \\
y(i+1) = y(i) + f(t(i), y(i)) \cdot dt \]
end
endfunction

Applying euler to our example:

-->function y = func(t, y)
--> y = - 2*t*y
-->endfunction

-->[t, y] = euler(func, 0, 1, 0.01, 200);

-->plot2d(t, y)

[Graph image]

We can compare the computed solution to the exact solution:

-->err = y - \text{exp}(-t.*t);

-->plot2d(t, err)
We see that the error is small enough, of the order of \(10^{-3}\), for the computed and exact solutions to be barely distinguishable on a graph. The shape of the error curve is not significant.

[Aside: the general shape of the error curve can be simply explained. From the graphical interpretation of Euler's method it follows that Euler's method overestimates the solution when the slope of the solution is increasing (as in the diagrams) and underestimates the solution when the slope of the solution is decreasing. The change from one type of behaviour to the other will occur near a point of inflection, which, in the exact solution to our example occurs at \(x = 1/\sqrt{2}\). There are also other types of error to consider as well to give a complete account.]

Euler's method does always perform as well as it does in the example above. Here is an example where it gives wildly inaccurate results:

```matlab
-->[t,y]=euler(func,0,1,1,10);
```

```matlab
-->y'
ans =

    column 1 to 9

        ! 1. 1. - 1. 3. - 15. 105. - 945. 10395. - 135135. !

    column 10 to 11
```
The important change here was the step size. It is clear from the derivation of Euler's method that the accuracy should increase as the step size is decreased, but this example shows that if the step size is too large then totally inaccurate results can be obtained.

1.2.3 Other Methods

The methods used in practice for the numerical solution of differential equations are much more sophisticated than Euler’s method. The main issue is the trade-off between smaller step sizes to increase accuracy and larger step sizes to increase efficiency, with larger step sizes fewer steps are needed to cover a given range. Here are brief descriptions of the rationale behind some common numerical methods:

1. Runge-Kutta methods: Euler’s method uses only the information at one point \((t, y)\) and takes a step with slope \(f(t, y)\). Runge-Kutta methods, by contrast, evaluate \(f(t, y)\) at a number of points for each step. A simple example of this idea is to evaluate \(f(t, y)\) and \(f(t + \Delta t, y + \Delta y)\), where \(y + \Delta y\) is obtained from Euler’s method, and take a step whose slope is the average of the two values.

2. Implicit methods: A simple example of an implicit method is the backward Euler method. Euler’s formula is

\[
\hat{y}(t_0 + \Delta t) = \hat{y}(t_0) + f(t_0, \hat{y}(t_0))\Delta t
\]

The backward Euler formula is

\[
\hat{y}(t_0 + \Delta t) = \hat{y}(t_0) + f(t_0 + \Delta t, \hat{y}(t_0 + \Delta t))\Delta t
\]

this is an implicit formula for \(\hat{y}(t_0 + \Delta t)\) which needs to be solved numerically before the step can be taken. Implicit methods are typically more stable, in the sense of avoiding the problems associated with large step sizes, than explicit methods. The disadvantage is that a nonlinear equation has to be solved at each step.

3. Multistep methods: These use not only the information at the current point but information at previously computed points in making each step. Since such information is not available at the first step they need other methods to get started. There are explicit and implicit multistep methods. A typical use is to use an explicit method to make a tentative step and then use the value obtained as an initial approximation for an implicit step. Such methods are often referred to as predictor-corrector methods.
2 Scilab

The Scilab function for solving initial value problems is `ode`. Its simplest use is of the form

\[ y = \text{ode}(y0, t0, t, f) \]

Here \( y0 \) and \( t0 \) are the initial values, \( f \) is the right hand side of the differential equation, and \( t \) is a vector of values at which the solution is computed. The solution at these points is returned in the vector \( y \). Don’t confuse the vector \( t \) with steps taken by the solver; the solver itself determines the step sizes it takes and then computes the values of \( y \) at the values specified by \( t \) by interpolation.

Here is our example using `ode`:

\[ \rightarrow t = 0:0.01:2; \]
\[ \rightarrow y = \text{ode}(1, 0, t, \text{func}); \]
\[ \rightarrow \text{plot2d}(t, y) \]

![Graph](image)

Again we can calculate the error:

\[ \rightarrow \text{err} = y - \exp(-t.*t); \]
\[ \rightarrow \text{plot2d}(t, \text{err}) \]
The rough shape of the graph is mostly due to the step size changing during the computation.