LECTURE 9

Numerical Methods for ODEs, II

1. Runge-Kutta Methods

In Lecture 5, we discussed the Euler method; a fairly simple iterative algorithm for determining the solution of an initial value problem

\[ \frac{dx}{dt} = F(t, x) , \quad x(t_0) = x_0 . \]

The key idea was to interpret the \( F(x, t) \) as the slope \( m \) of the best straight line fit to the graph of a solution at the point \((t, x)\). Knowing the slope of the solution curve at \((t_0, x_0)\) we could get to another (approximate) point on the solution curve by following the best straight-line-fit to a point \((t_1, x_1) = (t_0 + \Delta t, x_0 + m_0 \Delta t)\), where \(m_0 = F(t_0, x_0)\). And then we could repeat this process to find a third point \((t_2, x_2) = (t_1 + \Delta t, x_1 + m_1 \Delta t)\), and so on. Iterating this process \(n\) times gives us a set of \(n + 1\) values \(x_i = x(t_i)\) for an approximate solution on the interval \([t_0, t_0 + n\Delta t]\).

Now recall from our discussion of the numerical procedures for calculating derivatives (Lecture 6) that the formal definition

\[ \frac{dx}{dt} = \lim_{h \to 0} \frac{x(t+h) - x(t)}{h} \]

does not actually provide the most accurate numerical procedure for computing derivatives. For while

\[ \frac{dx}{dt} = \frac{x(t+h) - x(t)}{h} + O(h) \]

a more accurate formula would be

\[ \frac{dx}{dt} = \frac{4}{3h} (x(t+h/2) - x(t-h/2)) - \frac{1}{6h} (x(t+h) - x(t-h)) + O(h^4) \]

and even more accurate formulas were possible using Richardson Extrapolations of higher order.

In a similar vein, we shall now seek to improve on the Euler method. Let us begin with the Taylor series for \(x(t+h)\):

\[ x(t+h) = x(t) + hx'(t) + \frac{h^2}{2} x''(t) + \frac{h^3}{6} x'''(t) + O(h^4) \]

From the differential equation we have (by differentiating the differential equation and applying the two-variable chain rule)

\[ \frac{d^2 x}{dt^2} = \frac{d}{dt} \frac{dx}{dt} = \frac{d}{dt} F(t, x) = \frac{\partial F}{\partial t} (t, x) \frac{dt}{dt} + \frac{\partial F}{\partial x} \frac{dx}{dt} = F_t (t, x) + F_x (t, x) F(t, x) \]

And so the Taylor series for \(x(t+h)\)

\[ x(t+h) = x(t) + hx'(t) + \frac{h^2}{2} x''(t) + O(h^3) \]
can be expressed in terms of $F(t, x)$ and its derivatives
\[ x(t + h) = x(t) + h F(t, x) + \frac{h^2}{2} \left( F_t(t, x) + F_x(t, x) F(t, x) \right) + \mathcal{O}(h^3) \]
which after regrouping terms is the same as
\[ x(t + h) = x(t) + \frac{1}{2} h F(t, x) + \frac{1}{2} h \left( F_t(t, x) + h F_t(t, x) + h F(t, x) F_x(t, h) \right) + \mathcal{O}(h^3) \]
Now, regarding $F(t + h, x + h F(t, x))$ as a function of $h$, we have the following Taylor expansion
\[ F(t + h, x + h F(t, x)) = F(t, x) + h F_t(t, x) + F_x(t, h) (h F(t, h)) + \mathcal{O}(h^2) \]
Now you see the purpose of the regrouping in (2); the tail end of (2) coincides with $\frac{1}{2} h$ times the right hand side of (3). This allows us to write
\[ x(t + h) = x(t) + \frac{h}{2} F(t, x) + \frac{h}{2} F(t + h, x + h F(t, x)) + \mathcal{O}(h^3) \]
or
\[ x(t + h) \approx x(t) + \frac{h}{2} (f_1 + f_2) + \mathcal{O}(h^3) \]
where
\[ f_1 \equiv F(t, x) \]
\[ f_2 \equiv F(t + h, x + f_1) \]
Notice that formulas (4), (5), and (5) allow us to use initial data $x(t_0) = x_0$, to compute an approximate value for $x(t_0 + h)$ in three steps:

(i) compute $f_1 = F(t_0, x_0)$
(ii) compute $f_2 = F(t_0 + h, x + f_1)$
(iii) compute $x(t + h) = x_0 + \frac{h}{2} (f_1 + f_2)$

We thus arrive at the following algorithm for computing a solution to the initial value problem (1):

1. Partition the solution interval $[a, b]$ into $n$ subintervals:
   \[ \Delta t = \frac{b - a}{n} \]
   \[ t_k = a + k \Delta t \]
2. Set $x_0$ equal to $x(a)$ and then for $k$ from 0 to $n - 1$ calculate
   \[ f_{1,k} = \Delta t F(t_k, x_k) \]
   \[ f_{2,k} = \Delta t F(t_{k} + \Delta t, x_{k} + \Delta t f_{1,k}) \]
   \[ x_{k+1} = x_k + \frac{1}{2} (f_{1,k} + f_{2,k}) \]

This method is known as Heun’s method or the second order Runge-Kutta method (3).

Higher order Runge-Kutta methods are also possible; however, they are very tedious to derive. Here is the formula for the classical fourth-order Runge-Kutta method:
\[ x(t + h) = x(t) + \frac{h}{6} (f_1 + 2 f_2 + 2 f_3 + f_4) + \mathcal{O}(h^4) \]
where

\[
\begin{align*}
f_1 &= F(t, x) \\
f_2 &= F\left(t + \frac{1}{2} h, x + \frac{1}{2} f_1 \right) \\
f_3 &= F\left(t + \frac{1}{2} h, x + \frac{1}{2} f_2 \right) \\
f_4 &= F(t + h, x + f_3) \\
\end{align*}
\]

Below is a Maple program that implements the fourth order Runge-Kutta method to solve

(7) \[ \frac{dx}{dt} = -\frac{x^2 + t^2}{2xt}, \quad x(1) = 1 \]
on the interval [1, 2].

```
F := (x,t) -> -(x^2+t^2)/(2*x*t) ;
n := 100;
t[0] := 1.0;
x[0] := 1.0;
h := 1.0/n
for i from 0 to n-1 do
    f1 := evalf(h*F(t[i],x[i]));
    f2 := evalf(h*F(t[i]+h/2,x[i]+f1/2));
    f3 := evalf(h*F(t[i] + h/2,x[i]+f2/2));
    f4 := evalf(h*F(t[i]+h,x[i]+f3));
    t[i+1] := t[i] + h;
x[i+1] := x[i] + (f1 + 2*f2 + 2*f3 + f4)/6;
end do:
```

The exact solution to (7) is

\[ x(t) = \sqrt{\frac{1}{3} \left( \frac{4}{t} - t^3 \right)} \]

2. Error Analysis for the Runge-Kutta Method

Recall from the preceding lecture the formula underlying the fourth order Runge-Kutta Method: if \( x(t) \) is a solution to

\[ \frac{dx}{dt} = F(t, x) \]

then

\[ x(t_0 + h) = x(t_0) + \frac{1}{6} \left( f_1 + 2f_2 + 2f_3 + f_4 \right) + O(h^5) \]
\[ f_1 = hF(t_0, x_0) \]
\[ f_2 = hF \left( t_0 + \frac{1}{2}h, x_0 + \frac{1}{2}f_1 \right) \]
\[ f_3 = hF \left( t_0 + \frac{1}{2}h, x_0 + \frac{1}{2}f_2 \right) \]
\[ f_4 = hF \left( t_0 + h, x_0 + f_3 \right) \]

Thus, the term \( O(h^5) \) is the “local truncation error”; it corresponds to the error induced for each successive stage of the iterated algorithm; at each stage the difference between the computed value and the actual value will be of the form

\[ err = Ch^5 \]

for some constant \( C \). Here \( C \) is a number independent of \( h \), but dependent on \( t_0 \) and the fourth derivative of the exact solution \( \tilde{x}(t) \) at \( t_0 \) (the constant factor in the error term corresponding to truncating the Taylor series for \( x(t_0 + h) \) about \( t_0 \) at order \( h^4 \)). To estimate \( Ch^5 \) we shall assume that the constant \( C \) does not change much as \( t \) varies from \( t_0 \) to \( t_0 + h \).

Let \( u \) be the approximate solution to \( \tilde{x}(t) \) at \( t_0 + h \) obtained by carrying out a one-step fourth order Runge-Kutta approximation:

\[ \tilde{x}(t) = u + Ch^5 \]

Let \( v \) be the approximate solution to \( \tilde{x}(t) \) at \( t_0 + h \) obtained by carrying out a two-step fourth order Runge-Kutta approximation with step sizes of \( \frac{1}{2}h \)

\[ \tilde{x}(t) = v + 2C \left( \frac{h}{2} \right)^5 \]

Subtracting these two equations we obtain

\[ 0 = u - v + C \left( 1 - 2^{-4} \right) h^5 \]

or

\[ \text{local truncation error} = Ch^5 = \frac{u - v}{1 - 2^{-4}} \approx u - v \]

In a computer program that uses a Runge-Kutta method, this local truncation error can be easily monitored, by occasionally computing \( |u - v| \) as the program runs through its iterative loop. Indeed, if this error rises above a given threshold, one can readjust the step size \( h \) on the fly to restore a tolerable degree of accuracy. Programs that uses algorithms of this type are known as adaptive Runge-Kutta methods.

3. Systems of First Order ODEs

It turns out the that the Runge-Kutta method just describes is easily extendable to the situation of a system of first order ODEs. Indeed, consider such a system expressed in matrix notation:

\[ \frac{dx}{dt} = F(t, x) = \begin{pmatrix} f_1(t, x) \\ \vdots \\ f_n(t, x) \end{pmatrix} \]

Then partitioning the interval in question as \( t_0, t_1 = t_0 + h, t_2 = t_0 + 2h, \ldots \) and setting

\[ x_k \approx x(t_k) \]

and following the derivation of the Runge-Kutta formula above one arrives at the following recursive formula for \( x_{k+1} \)

\[ x_{k+1} = x_k + \left( \frac{h}{6} \right) (f_{1,k} + 2f_{2,k} + 2f_{3,k} + f_{4,k}) \]
where
\[
\begin{align*}
f_{1,k} &= F(t_k, x_k) \\
f_{2,k} &= F\left(t_k + \frac{h}{2}, x_k + \frac{h}{2} f_{1,k}\right) \\
f_{3,k} &= F\left(t_k + \frac{h}{2}, x_k + \frac{h}{2} f_{2,k}\right) \\
f_{4,k} &= F(t_k + h, x_k + h f_{3,k})
\end{align*}
\]

4. Multistep Methods

Here we will develop another method for improving the accuracy of numerical solutions of first order ODEs. Like the original Euler method begin with a first order ODE
\[
\frac{dx}{dt} = F(t, x)
\]
and a systematic partition of the interval upon which we want to know \(x(t)\):
\[
\begin{align*}
t_0 &= t_0 \\
t_1 &= t_0 + \Delta t \\
&\quad \vdots \\
t_n &= t_0 + n\Delta t
\end{align*}
\]
Recall the Fundamental Theorem of Calculus
\[
x(t_{n+1}) - x(t_n) = \int_{t_n}^{t_{n+1}} \frac{dx}{dt}(t) \, dt
\]
The basic idea behind the Adams method is approximate \(\frac{dx}{dt}(t)\) by a polynomial \(P(t)\); which in turn can be easily integrated to yield a formula
\[
x(t_{n+1}) = x(t_n) + \int_{t_n}^{t_{n+1}} P(t) \, dt
\]
Suppose, for example, we wish to use a polynomial
\[
P_1(t) = At + B
\]
of degree 1 to approximate \(\frac{dx}{dt} = F(t, x)\). We need two conditions on \(P_1\) to fix the coefficients \(A\) and \(B\). Assuming that previous \(x(t_n)\) and \(x(t_{n-1})\) have already know, we can require
\[
\begin{align*}
P_1(t_{n-1}) &= f_{n-1} \\
P_1(t) &= f_n
\end{align*}
\]
More explicitly, we require
\[
\begin{align*}
f_{n-1} &= P_1(t_{n-1}) = At_{n-1} + B \\
f_n &= P_1(t_n) = At_n + B
\end{align*}
\]
Solving for \(A\) and \(B\) yields
\[
\begin{align*}
A &= \frac{f_n - f_{n-1}}{t_n - t_{n-1}} = \frac{f_n - f_{n-1}}{h} \\
B &= \frac{f_{n-1} t_n - f_n t_{n-1}}{t_n - t_{n-1}} = \frac{f_{n-1} t_n - f_n t_{n-1}}{h}
\end{align*}
\]
where \(h = t_n - t_{n-1}\).
We can now write
\[ x(t_{n+1}) = x(t_n) + \int_{t_n}^{t_{n+1}} (At + B) \, dt \]
\[ = x(t_n) + \frac{A}{2} (t_{n+1}^2 - t_n^2) + B(t_{n+1} - t_n) \]
\[ = x(t_n) + \frac{1}{2} \left( \frac{f_{n-1} - f_n}{h} \right) (t_{n+1} - t_n) (t_{n+1} + t_n) + \left( \frac{f_{n-1} t_n - f_n t_{n-1}}{h} \right) (t_{n+1} - t_n) \]
\[ = x(t_n) + \frac{1}{2} (f_n - f_{n-1}) (2t_n + h) + (f_{n-1} t_n - f_n (t_n - h)) \]
\[ = x(t_n) + \frac{3}{2} h f_n - \frac{1}{2} h f_{n-1} \]
(note that, in the second to the last line, the terms involving \( t_n \) all cancel).