LECTURE 21

Runge-Kutta Methods

In the preceding lecture we discussed the Euler method; a fairly simple iterative algorithm for determining the solution of an intial value problem

$$\frac{dx}{dt} = F(t,x) \quad , \quad x(t_0) = x_0 \ . \tag{21.1}$$

The key idea was to interprete 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 that the formal definition

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

does actually provide the most accurate numerical procedure for computing derivatives. For

$$\frac{dx}{dt} = \frac{x(t+h)-x(t)}{h} + \mathcal{O}(h)$$

but a more accurate formula would be

$$\frac{dx}{dt} = \frac{4}{3h}\left(x(t+h/2)-x(t-h/2)\right) - \frac{1}{6h}\left(x(t+h)-x(t)\right) + \mathcal{O}\left(h^4\right)$$

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 $x(t+h)$:

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

From the differential equation we have

$$\begin{aligned} x'(t) &= F \\ x''(t) &= F_t + F_x F \\ x'''(t) &= F_{tt} + F_{tx}F + \left(F_{xt}+F_{xx}F\right)F + (F_t+F_xF)F_x \end{aligned}$$

And so the Taylor series for $x(t+h)$ can be written

$$x(t+h) = x(t) + hF(t,x) + \frac{h^2}{2}\left(F_t(t,x) + F_x(t,x)F(t,x)\right) + \mathcal{O}(h^3) \tag{21.2}$$

$$= x(t) + \frac{1}{2}hF(t,x) + \frac{1}{2}h\left(F(t,x) + hF_t(t,x) + hF(t,x)F_x(t,h)\right) + \mathcal{O}(h^3) \tag{21.3}$$

Now

$$F(t+h, x+hF(t,h)) = F(t,x) + hF_t(t,x) + F_x(t,h)\left(hF(t,h)\right) + \mathcal{O}(h^2)$$

and so we can rewrite (??) as

$$x(t+h) = x(t) + \frac{h}{2}F(t,x) + \frac{h}{2}F\left(t+h, x+hF(t,x)\right) + \mathcal{O}\left(h^3\right)$$

or

$$x(t+h) = x(t) + \frac{1}{2}\left(F_1 + F_2\right) \tag{21.4}$$

where

$$F_1 = hF(t,x) \tag{21.5}$$

$$F_2 = hF\left(t+h, x+F_1\right) \tag{21.6}$$

We thus arrive at the following algorithm for computing a solution to the intial value problem (21.1):

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

   This method is known as **Heun's method** or the **second order Runge-Kutta method**.

Higher order Runge-Kutta methods are also possible; however, they are very tedius to derive. Here is the formula for the **classical fourth-order Runge-Kutta method**:

$$x(t+h) = x(t) + \frac{1}{6}\left(F_1 + 2F_2 + 2F_3 + F_4\right)$$

where

$$\begin{aligned} F_1 &= hF(t,x) \\ F_2 &= hF\left(t + \frac{1}{2}h, x + \frac{1}{2}F_1\right) \\ F_3 &= hF\left(t + \frac{1}{2}h, x + \frac{1}{2}F_2\right) \\ F_4 &= hF\left(t+h, x+F_3\right) \end{aligned}$$

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

$$\frac{dx}{dt} = -\frac{x^2+t^2}{2xt} \quad , \quad x(1) = 1 \tag{21.7}$$

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;
od:
```

The exact solution to (21.7) is

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

1. 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

$$x(t_0 + h) = x(t_0) + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4) + \mathcal{O}(h^5)$$

where

$$\begin{aligned} F_1 &= hf(t_0, x_0) \\ F_2 &= hf\left(t_0 + \frac{1}{2}h, x_0 + \frac{1}{2}F_1\right) \\ F_3 &= hf\left(t_0 + \frac{1}{2}h, x_0 + \frac{1}{2}F_2\right) \\ F_4 &= hf(t_0 + h, x_0 + F_3) \end{aligned}$$

Thus, the local truncation error (the error induced for each successive stage of the iterated algorithm) will behave like

$$err = Ch^5$$

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 $\frac{1}{2}h$)

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

Substracting these two equations we obtain

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

or

$$\text{local truncation error } = Ch^5 = \frac{u-v}{1-h^{-4}} \approx u - v$$

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**.