Solving Differential Equations Numerically

Solving DE's

We have seen in earlier slides, that the only type of differential equations that we need to consider, are differential equations of form:

$$\overline{y}'(t) = \overline{F}(\ \overline{y}(t)\ ).$$

These are first-order, autonomous differential equations.

One may either call it 'system of differential equations' or a vector-valued differential equation.

Euler's Method

```
vector y = y0;       // Initial value.
double h = ( t1 - t0 ) / 1000;

double t = t0;
while( t + h < t1 )
{
   y += h * F(y);
   t += h;
}
y += ( t1 - t ) * F(y);
t = t1;
```

What is the accurracy of this method?

Heun's Method

First define

```
vector H( vector y, double h )
{
   vector y_star = y + h * F(y);
   return (h/2) * ( F(y) + F(y_star );
}
```

The function $H(y, h)$ first makes a guess for $y(t + h)$ using Euler's method.

It uses the average of F in the guessed value for $y(t + h)$ and F in the present $y(t)$, to obtain the final estimation for $y(t + h)$.

Heun's Method (2)

```
vector y = y0;       // Initial value.
double h = ( t1 - t0 ) / 1000;

double t = t0;
while( t + h < t1 )
{
   y = H( y, h );
   t += h;
}

y = H( y, t1 - t );
t = t1;
   // Now y is the (approximated) solution y1.
```

What is the accurracy of this method?

Runge-Kutta Methods

The general form of a Runge-Kutta method is as follows: Assume that we already know $\overline{y}(t)$ and that we want to compute an approximation for $\overline{y}(t+h)$.

We first compute a sequence of values $\overline{k}_1, \ldots, \overline{k}_n$, where each $\overline{k}_i$ is an approximation for $\overline{y}(t+c_k h)$, for some $c_k \in \mathcal{R}$.

At each level $(i+1)$, the previous estimations $\overline{k}_1, \ldots, \overline{k}_i$ are used to compute the next estimation $\overline{k}_{i+1}$.

The sequence starts with

$$\overline{k}_1 = \overline{F}(\ \overline{y}(t)\ ).$$

Then, for each i $(1 \leq i < n)$, we compute

$$\overline{k}_{i+1} = \overline{F}(\ \overline{y}(t) + h\ (A_{i+1,1}\overline{k}_1 + A_{i+1,2}\overline{k}_2 + \cdots + A_{i+1,i}\overline{k}_i)\ ).$$

Runge-Kutta Methods (2)

At the end, we compute:

$$\overline{y}(t+h) = \overline{y}(t) + h.(b_1\overline{k}_1 + b_2\overline{k}_2 + \cdots + b_n\overline{k}_n).$$

It is easily seen that $c_1 = 0$, and

$$c_{i+1} = A_{i+1,1} + \cdots A_{i+1,i}.$$

Also,

$$b_1 + b_2 + \cdots + b_n = 1.$$

(In order to see this, replace $\overline{k}_i$ by $\overline{F}(\ \overline{y}(t)\ ) = \overline{y}'(t)\ )$.

Butcher Tableaux

A Runge-Kutta method with n stages is determined by the coefficients $A_{i,j}$, the positions c_i, and the final weights $b_1, \ldots, b_n$.

The coefficients are usually written in a tableau as follows, which is called Butcher Tableau (after J.C. Butcher).

$$
\begin{array}{c|ccccc}
c_1 & & & & & \\
c_2 & A_{2,1} & & & & \\
c_3 & A_{3,1} & A_{3,2} & & & \\
\ldots & \ldots & \ldots & & & \\
c_n & A_{n,1} & A_{n,2} & \cdots & A_{n-1} & \\
\hline
 & b_1 & b_2 & \cdots & b_{n-1} & b_n
\end{array}
$$

Order 1

The simplest Runge-Kutta method (the Euler Method) is defined by the following tableau:

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$$

It corresponds to the computation:

$$\overline{k}_1 := \overline{F}(\ \overline{y}(t)\ ),$$

$$\overline{y}(t+h) = \overline{y}(t) + h\ \overline{k}_1.$$

RK21 (Heun's Method)

Heun's method is defined by the following tableau:

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

It corresponds to the computation:

$$\overline{k}_1 := \overline{F}(\ \overline{y}(t)\ ),$$

$$\overline{k}_2 := \overline{F}(\ \overline{y}(t) + h.\overline{k}_1\ ),$$

$$\overline{y}(t+h) = \overline{y}(t) + h.(\frac{1}{2}\overline{k}_1 + \frac{1}{2}\overline{k}_2).$$

The order (for a single step) is 3.

RK41

The best-known method, which is usually called the Runge-Kutta method, is defined by the following tableau:

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Its order (for a single step) is 5.

Above order 5, the number of stages grows quicker than the order.

RK5

The following method has order (in a single step) 6. It has 6 stages, two more than RK41.

0						
$\frac{1}{4}$	$\frac{1}{4}$					
$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$				
$\frac{1}{2}$	0	0	$\frac{1}{2}$			
$\frac{3}{4}$	$\frac{3}{16}$	$-\frac{3}{8}$	$\frac{3}{8}$	$\frac{9}{16}$		
1	$-\frac{3}{7}$	$\frac{8}{7}$	$\frac{6}{7}$	$-\frac{12}{7}$	$\frac{8}{7}$	
	$\frac{7}{90}$	0	$\frac{32}{90}$	$\frac{12}{90}$	$\frac{32}{90}$	$\frac{7}{90}$

Proving Order Properties of Runge-Kutta Methods

Proving the order properties of RK methods is rather hard.

The idea of the proof is not hard:

1. Approximate $\overline{F}$ in $\overline{y}(t)$ by a Taylor polynomial.
2. Using the Taylor polynomial for $\overline{F}$, also express $\overline{y}(t+h)$ as Taylor polynomial.
3. Using an abstract Butcher tableau, give a Taylor polynomial for the estimation $\overline{y}^*(t+h)$.
4. Try to make as many coefficients of $\overline{y}(t+h)$ and $\overline{y}^*(t+h)$ equal, by finding proper values in the Butcher tableau. The more coefficients become equal, the better the order.

Proving Order Properties of Runge-Kutta Methods

The previous plan works fine for orders 1,2,3. For higher orders, it results in serious disaster.

The polynomials get so big that no computer algebra system can handle them.

Martin Kutta (1867-1944, born in Pitschen, now Byczyna) and John.C. Butcher (1933-) used symmetries (and combinatorics) to derive the order conditions. They checked in how many ways each chain of partial derivatives can be realized in the expressions. Even then, the order conditions become too big to solve.

For higher-orders, no optimal (stage as low as possible) methods are known.

Examples

I give an example, using the catenary. Its differential equation is:

$$y''(x) = \lambda.\sqrt{1 + ((y'(x))^2}.$$

Its exact solution is defined by

$$y(x) = \frac{1}{\lambda}\cosh(\lambda x) = \frac{e^{\lambda x} + e^{-\lambda x}}{2\lambda}.$$

It can easily shown that

$$\cosh^2(x) - \sinh^2(x) = 1,$$

and that

$$\sinh'(x) = \cosh(x), \quad \cosh'(x) = \sinh(x).$$

We first have to make the different equation first-order. First define pairs over $\mathcal{R} \times \mathcal{R}$. Call the first component u and the second component v. Define $\overline{p}(\overline{x}) = (y(x), y'(x))$. The differential equation becomes

$$\begin{cases} \overline{p}'_v(x) &= \overline{p}_w(x) \\ \overline{p}'_w(x) &= \lambda.\sqrt{1 + (\overline{p}_w(x))^2}. \end{cases}$$

It is first-order and autonomous. The exact solutions have form

$$\begin{cases} \overline{p}_v(x) &= \frac{1}{\lambda}\cosh(\lambda x) \\ \overline{p}_w(x) &= \sinh(\lambda x) \end{cases}$$

In order to use Runge-Kutta methods, implement

```
class pair
{
   double v;
   double w;
}
```

and the operators

```
pair operator + ( pair&, pair& );
pair operator * ( double, pair );
```

Problems with Runge Kutta Methods

RK methods are very useful for static analysis, performance analysis, orbit computations etc. Unfortunately, they are not always suitable for real time simulation:

1. F has to be sufficiently often differentiable. (Otherwise, it the Taylor approximation does not approach the function fast enough, and the RK method is not as accurate as it should be.) Many systems in real life involve state changes which makes them not differentiable often enough.

2. In practice, one does not have access to all state variables. One needs to be able to compute $\overline{F}(\overline{x})$ for many different, guessed values of $\overline{x}$. In practice, the state $\overline{x}$ may be spread through different objects, partially inside private variables of these objects and therefore not accessible. From the point of software engineering, it is probably impossible to use RK methods on

real, big objects.

3. On many systems, passing from t to $t + h$ may involve a state change that is hard to reverse. Using an RK method involves evaluating F repeatedly at different positions.

4. User input is usually erratic and tends to be not differentiable.