Lecture MATLAB Scip-continuation course, ODE-systems

Systems of differential equations

Heikki Apiola and juha Kuortti

February 19, 2018

Aalto University
juha.kuortti@aalto.fi, heikki.apiola@aalto.fi

Numerical methods of ODE systems

MATLAB's ODE-solvers

- MATLAB has a remarkable collection of ODE-solvers. Using the correct ODE solver can save you lots of time and give more accurate results. Start at `» help ode45`.
  - `ode23`

    Low-order solver. Use when integrating over small intervals or when accuracy is less important than speed.
  - `ode45`

    Higher-order solver. High accuracy and reasonable speed. Most commonly used.
  - `ode15s`

    "Stiff" ODE solver, use when the diff. eq's have time constants that vary by orders of magnitude. (Eg. some chemical reactions)
  - `» help ode45` shows 4 more solvers + auxiliary functions and examples.

ODE solvers: standars syntax

- To use standard options and variable time step

  `» [T,Y]= ode45(@myODE,[0,10],[1;0])`
- Inputs:
  - `@myODE`: **Function handle**, defines the differential equation. This function takes input `(t,y)` and returns the dy-(column vector) of derivatives.
  - `[0 10]`: **Time-interval**, initial and final t-values.
  - `[1;0]`: **Initial condition** for each equation (2 eq's in this case), column vector.
- Outputs:
  - `T` contains the time points.
  - Each column of `Y` contains the corresponding values of the numerical approximations of the solution functions.
- A call `» ode45(@myODE,[0,10],[1;0])` without output arguments results in a plot of the solution functions.

ODE solvers, adaptivity, custom options

- ODE solvers use variable time steps, this is called adaptivity: Rapid change needs smaller steps for a given error tolerance. The ODE-algorithm chooses the time steps.
- Often one wants to compute the values at a user specified set of points. The natural way is to use spline interpolation. Actually, MATLAB does it in a more efficient and accurate way: `doc ode45`, read ahead.
- Instead of the time interval `[a b]`, one can call like

  ```
  » tpoints=0:0.001:0.5;
  » [T,Y]=ode45(odefun,tpoints,[0;1]);
  ```

  This has little effect to the adaptivity process, but it gives the values at `tpoints`, which is also returned at `T`.

Viewing results

- To solve and plot the results of an ODE:

  `»[T,Y]=ode45(F,[0 0.5],[0;1]);` $y_1(0) = 0, y_2(0) = 1$

  `» plot(T,Y(:,1),'k','LineWidth',1.5,...`

  `T,Y(:,2),'r','LineWidth',1.5);`

  Or just: `» plot(T,Y,'LineWidth',1.5);`

  `» legend('y1','y2')`

  `» xlabel('Time (s)')`

  `» ylabel('Amount (g)')`

  `» title('Population or chemical or ..')`

  `» figure` Open a new graphics window

  `» plot(Y(:,1),Y(:,2))` For autonomous

  `» title('Phase plane')`

Optional arguments, tolerances, stats,...

The function `odeset` generates an "options" structure, example:

```
opts=odeset('reltol',1e-5,'abstol',1e-8,'stats','on')
```

creates a structure with fields: (Just a few lines are shown.)

```
AbsTol: 1.0000e-08
Events: []
RelTol: 1.0000e-05
Stats: 'on'
...
```

A call `[T,Y]=odexx(F,tspan,y0,opts);` uses these values as extra arguments.

The **default tolerances** are `'reltol',1e-3,'abstol',1e-6`.

Solution structure

```
sol = ode45(odefun,[t0 tf],y0...) returns a structure
    that can be used with deval to evaluate the solution
    at any points between t0 and tf.
```

This could be useful in case we had had a heavy computation with solving an ODE, and want to evaluate the solution on several sets of t-values without having to solve the ODE again.
Of course spline-interpolation can also be used, but assumably, sol-structure with `deval` is more efficient and accurate.
One more useage would be to use sol-structure with `deval` to define a function of t that could be used for instance with `fminsearch` or `fzero`

Examples of passing parameters

Using sol-structure for optimization object function:

```
function d=solfun(sol,t,y0)
z=deval(sol,t);
z=z([1 3],:);
d=norm(z-y0);
```

The objective function for minimization could be defined as:

```
y0=[1.2;0];
% sol assigned above, or here
objf=@(t) solfun(sol,t,y0)
mindist=fminsearc(objf,...)
```

“Later we will see more elegant ways”, it’s now

```
function ydot = rabfox(t,y)
alpha=0.01  % Local variable
ydot = [2*y(1)-alpha*y(1)*y(2)
        -y(2)+alpha*y(1)*y(2)];
```

Instead we can have `alpha` as a parameter set outside:

```
function ydot = rabfox(t,y,alpha)
ydot = [2*y(1)-alpha*y(1)*y(2)
        -y(2)+alpha*y(1)*y(2)];
>> alpha=0.01;
>> odefun=@(t,y)rabfox(t,y,alpha)
```

(Here we could have defined `rabfox` as a function handle as the local parameter is no longer there.)

Two-body problem

Satellite and earth:

Newton's equation of motion:

$$m\vec{r}'' = -\gamma \frac{M\, m\vec{e}_r}{|\vec{r}|^2}.$$

Taking $M\gamma = 1$, we get in coordinate form:

$$\begin{cases} x''(t) = -\frac{x(t)}{(x(t)^2+y(t)^2)^{3/2}} \\ y''(t) = -\frac{y(t)}{(x(t)^2+y(t)^2)^{3/2}} \end{cases}$$

Let: $u_1 = x, u_2 = x', u_3 = y, u_4 = y'$.

Stop here for a moment, write the odefun!

$$\begin{cases} u_1' = u_2 \\ u_2' = x'' = -\frac{u_1}{(u_1^2+u_3^2)^{3/2}} \\ u_3' = u_4 \\ u_4' = y'' = -\frac{u_3}{(u_1^2+u_3^2)^{3/2}} \end{cases}$$

```
function du = Kepler(t,u)
% Odefunction for 2-body problem
r3 = (u(1)^2 + u(3)^2)^(3/2);
du = [  u(2)      ;
       -u(1)/r3   ;
        u(4)      ;
       -u(3)/r3] ;
```

Load Kepler.m and runKepler.m

Example of stiffness

Quote from Moler's book:

- "Stiffness is a subtle, difficult, and important concept in the numerical solution of ordinary differential equations. It depends on the differential equation, the initial conditions, and the numerical method. Dictionary definitions of the word "stiff" involve terms like "not easily bent," "rigid," and "stubborn." We are concerned with a computational version of these properties.
- *A problem is stiff if the solution being sought varies slowly, but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results.*

A model of flame propagation provides an example: (By *Larry Shampine*) If you light a match, the ball of flame grows rapidly until it reaches a critical size. Then it remains at that size because the amount of oxygen being consumed by the combustion in the interior of the ball balances the amount available through the surface. The simple model is

$$y' = y^2 - y^3, y(0) = \delta, 0 \leq t \leq 2/\delta$$

y(t) represents the radius of the ball.
Let's have a look at: stiff.m