Numerical Methods
5633

Lecture 7

Michaelmas Term 2018

Marina Krstic Marinkovic
mmarina@maths.tcd.ie

School of Mathematics
Trinity College Dublin

Numerical methods for ordinary differential equations

- **Initial value problem** (IVP): find a function **y(t)** s.t.

$$\frac{dy}{dt} = f(t, y(t)), \qquad y(t_0) = y_0$$

- IVP: if **t** - seen as time, we can view the equation as modelling a process that moves forward from some initial time t_0
- Function f:$\mathbf{R}^{n+1}$ **—> $\mathbf{R}^n$ (n>0)**
- **initial point t_0:** given scalar value, often **t_0 =0**
- **initial value y_0:** given vector in $\mathbf{R}^n$

- **Goal:** find the unknown function y(t)

$$y'(t) - f(t, y(t)) = 0, \qquad \forall t > t_0, \quad y(t_0) = y_0$$

Numerical methods for ordinary differential equations

- Numerical solution for ODE, Initial Value Problem:
  - Euler's method: forward, backward, midpoint, trapezoid, predictor-corrector
  - Runge-Kutta method (2nd and 4th order)

Numerical methods for ordinary differential equations

➡ Numerical solution for ODE, Initial Value Problem:

- **Euler's method: forward**, backward, midpoint, trapezoid, predictor-corrector
- Runge-Kutta method (2nd and 4th order)

(Forward) Euler's method

$$y_n = y_{n-1} + hf(t_{n-1}, y_{n-1})$$

```
dy.dx=function(x,y){1-y}

h=1/16.0
y0=1
start=0
end=1

ys<-euler(dy.dx, start=0, end=1, h=1/16.0, y0=0)
ys<-euler(dy.dx, start=0, end=1, h=1/32.0, y0=0)
```

```
# R code for the solution of the Initial Value Problem (IVP)
# using Euler's method
#
# Function Euler requires as input:
#           - dy.dx: function defining IVP
#           - h: stepsize, h*nsteps=(end-start)
#           - y0: initial value
#           - start: start of the interval
#           - end: end of the interval
#
# Output    - ys is a solution vector, containing values for
y(tk) after k=0...nstep iterations

euler<-function(dy.dx=function(x,y){},h=1E-7,y0=1,start=0,end=1)
{
    nsteps <- (end-start)/h
    ys <- numeric(nsteps+1)
    ys[1] <- y0
    for (i in 1:nsteps) {
        x <- start + (i-1)*h
        ys[i+1] <- ys[i] + h*dy.dx(x,ys[i])
    }
    ys
}
```

Error in Euler's method

$n=h^{-1}$	$E_n(y'+y=1,\ y(0)=0)$	$E_n(y'=y,\ y(0)=1)$
16	0.118053E-01	0.803533E-01
32	0.582415E-02	0.412917E-01
64	0.289292E-02	0.209369E-01
128	0.144173E-02	0.105428E-01
256	0.719686E-03	0.529020E-02
512	0.359550E-03	0.264983E-02
1024	0.179702E-03	0.132610E-02

Numerical methods for ordinary differential equations

- Numerical solution for ODE, Initial Value Problem:
  - Euler's method: forward, backward, midpoint, **trapezoid, predictor-corrector**
  - Runge-Kutta method (2nd and 4th order)

Trapezoid rule predictor-corrector method

$$\begin{aligned}\bar{y}_{n+1} =& y_n + hf(t_n, y_n)\\ \bar{y}_{n+1} =& y_n + \frac{1}{2}h\big[f(t_{n+1}, \bar{y}_{n+1}) + f(t_n, y_n)\big]\end{aligned}$$

➡ Algorithm for the Trapezoid Predictor Corrector:

```
1. input t0,y0,h,n
2. f=function(t0,y0){. . .}
3. for k in 1:n
        {
        # First, predict
        f0=f(t0,y0)
        ybar=y0+h*f0
        # Next, correct
        y0=y0+0.5*h*(f0+f(t+h,ybar))
        # Update for next pass through the loop
        y0=y
        t0=t
        }
```

Numerical methods for ordinary differential equations

- Numerical solution for ODE, Initial Value Problem:
  - Euler's method: forward, backward, midpoint, trapezoid, predictor-corrector
  - **Runge-Kutta method** (**2nd** and 4th **order**)

Runge-Kutta method

- Predictor-corrector written as a single recursion:

$$\boxed{y_{n+1} = y_n + \frac{1}{2}h\big[f(t_{n+1}, y_n + hf(t_n, y_n)) + f(t_n, y_n)\big]}$$

- Allow for an arbitrary average of the two slopes:

$$\boxed{y_{n+1} = y_n + c_1 hf(t_n, y_n) + c_2 hf(t_n + \alpha h, y_n + \beta hf(t_n, y_n))}$$

- Truncation error is **O(h²)** if (see derivation done in class):

$$\boxed{1 - c_1 - c_2 = 0; \qquad \frac{1}{2} - c_2\alpha = 0; \qquad \frac{1}{2} - c_2\beta = 0}$$

- One possible solution (trapezoid rule predictor-corrector):

$$\boxed{c_1 = c_2 = \frac{1}{2}; \qquad \alpha = \beta = 1}$$

Numerical methods for ordinary differential equations

- Numerical solution for ODE, Initial Value Problem:
  - Euler's method: forward, backward, midpoint, trapezoid, predictor-corrector
  - **Runge-Kutta method** (2nd and **4th order**)

Runge-Kutta method

- Algorithm for the fourth-order Runge-Kutta:

```
1. input t0,y0,h
2. for k in 1:n
            {
            t=t0+k*h
            t2=t0+0.5h
            v1=f(t0,y0)
            v2=f(t2,y0+0.5*h*v1)
            v3=f(t2,y0+0.5*h*v2)
            v4=f(t,y0+h*v3)
            y=(h/6.0)*(v1+2.0*(v2+v3)+v4)
            y0=y
            t0=t
            }
```

- Implemented in R together with many other methods for ODE: https://CRAN.R-project.org/package=deSolve