Computational Neurodynamics

Topic 3
Numerical Simulation

Murray Shanahan

Overview

- The Euler method
- The Runge-Kutta method

Simulating Neurons

- Given a mathematical model of a neuron's behaviour expressed as a set of ordinary differential equations (like the Hodgkin-Huxley model), we can use numerical methods to simulate the neuron's temporal dynamics by computer
- The Hodgkin-Huxley model is computationally expensive. Shortly we'll look at some simpler models with better computational properties
- But first we need to make a short excursion into numerical methods

Numerical Simulation

- Suppose we are given an ordinary differential equation (ODE) of the form

$$\frac{dy}{dt} = f(y)$$

and the initial value of y

- Now we want to compute the value of y as it changes over time
- This is an example of an *initial value problem*

The Euler Method 1

- Let $y(t)$ denote the value of y at time t
- Given $y(t)$, we can approximate the value of $y(t+\delta t)$

$$y(t+\delta t) \approx y(t) + \delta t . f(y(t))$$

- By repeatedly applying this formula we can plot the approximate trajectory of y over time
- This is known as the Euler method of numerical simulation
- But its accuracy is very sensitive to the step size δt

The Euler Method 2

- Here the Euler method is applied to compute

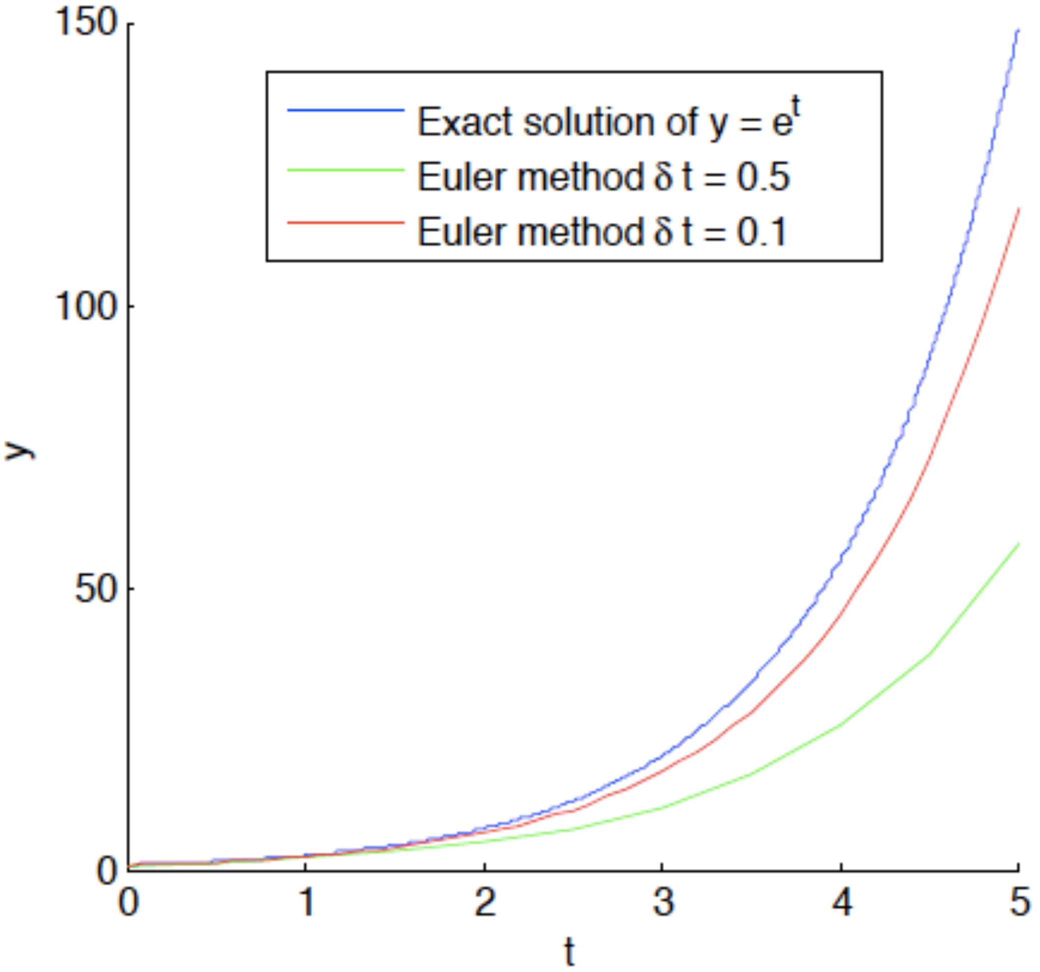
$$y = e^t$$

for which we have

$$\frac{dy}{dt} = y$$

giving

$$y(t+\delta t) \approx y(t) + \delta t \,.\, y(t)$$



Euler Matlab Code 1

```
function EulerDemo
% Solves the ODE dy/dt=y (exact solution: y(t)=exp(t)), by numerical
% simulation using the Euler method, for two different step sizes

dt1 = 0.5; % large step size
dt2 = 0.1; % smaller step size

% Time points
Tmin = 0;
Tmax = 5;
T1 = Tmin:0.001:Tmax;
T2 = Tmin:dt1:Tmax;
T3 = Tmin:dt2:Tmax;

% Exact solution
y1(1) = exp(Tmin); % initial value
for t = 1:length(T1)-1
   y1(t+1) = exp(T1(t+1));
end
```

- This is the associated Matlab code
- Two step sizes are compared $\delta t = 0.5$ and $\delta t = 0.1$
- First, the exact solution `y1(t)` is computed

Euler Matlab Code 2

- Then approximations are computed using the Euler method
- `y2(t)` uses `dt1` $= 0.5$
- `y3(t)` uses `dt2` $= 0.1$
- Finally all three series of values are plotted
- As the plots show, there is a big improvement in accuracy for the smaller step size

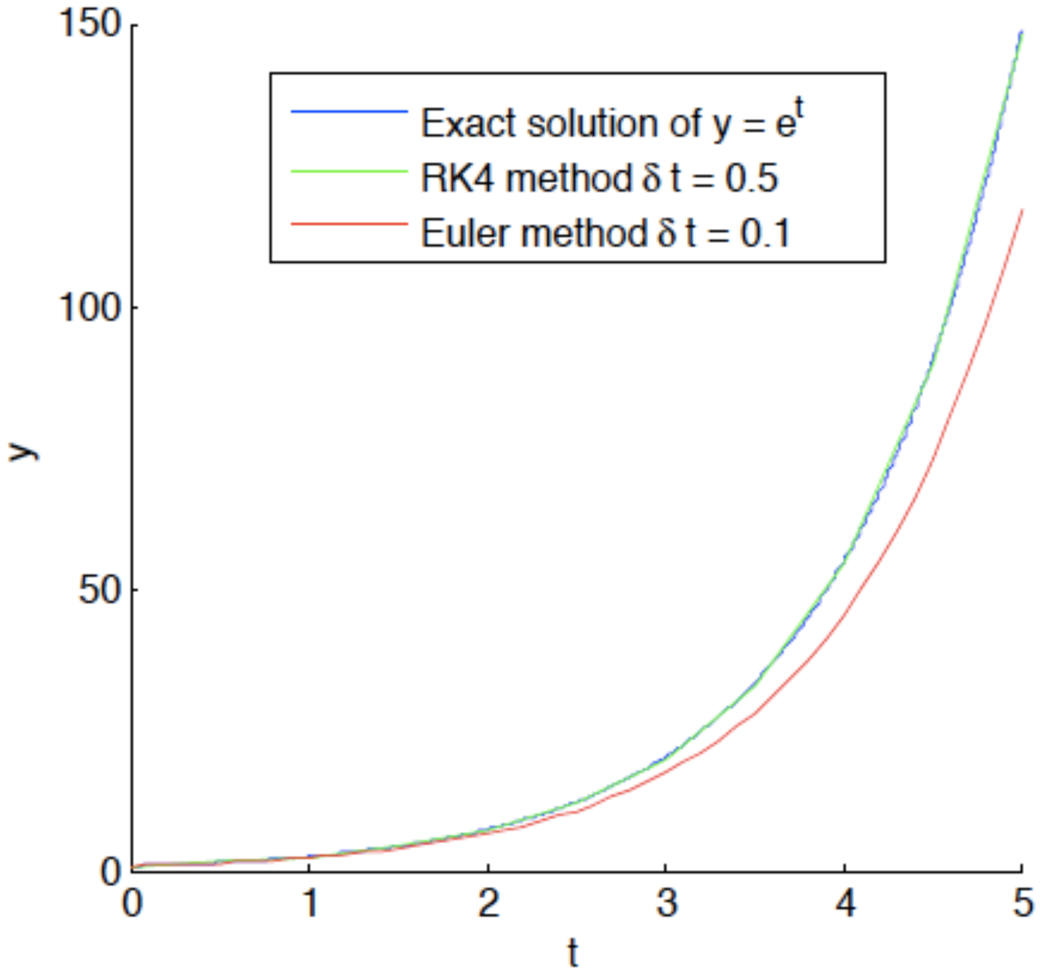
```
% Euler method with large step size
y2(1) = exp(Tmin); % initial value
for t = 1:length(T2)-1
   y2(t+1) = y2(t) + dt1*y2(t);
end

% Euler method with smaller step size
y3(1) = exp(Tmin); % initial value
for t = 1:length(T3)-1
   y3(t+1) = y3(t) + dt2*y3(t);
end

% Plot the results
hold on
plot(T1,y1,'Color','Blue')
plot(T2,y2,'Color','Green')
plot(T3,y3,'Color','Red')
```

The Runge-Kutta Method 1

- So one way to improve accuracy with the Euler method is to use a small step size
- But a computationally more efficient option is the Runge-Kutta method
- Here we see the 4th order Runge-Kutta method (RK4) used to approximate $y=e^t$



The Runge-Kutta Method 2

- Given $y(t)$, we can approximate the value of $y(t+\delta t)$ using the 4th order Runge-Kutta method by first computing

$$k_1 = f(y(t))$$
$$k_2 = f(y(t) + \tfrac{1}{2}\delta t.k_1)$$
$$k_3 = f(y(t) + \tfrac{1}{2}\delta t.k_2)$$
$$k_4 = f(y(t) + \delta t.k_3)$$

- Then we have

$$y(t+\delta t) \approx y(t) + \tfrac{1}{6}\delta t.(k_1 + 2k_2 + 2k_3 + k_4)$$

RK4 Matlab Code

- Here's the main loop of the Matlab code for approximating $y = e^t$ (so $f(y)=y$ making it rather easy) with RK4, as used to generate the earlier plot

```
% Runge-Kutta 4 method with large step size
y2(1) = exp(Tmin); % initial value
for t = 1:length(T2)-1

   k1 = y2(t);
   k2 = y2(t) + 0.5*dt1*k1;
   k3 = y2(t) + 0.5*dt1*k2;
   k4 = y2(t) + dt1*k3;

   y2(t+1) = y2(t) + dt1*(k1+2*k2+2*k3+k4)/6;

end
```