Computational Neurodynamics

Topic 3
Numerical Simulation

Pedro Mediano

(Slides: Pedro Mediano & 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
- 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 $y(t+\delta t)$

$$y(t+\delta t) = y(t) + \delta t\, f(y(t))$$

- 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

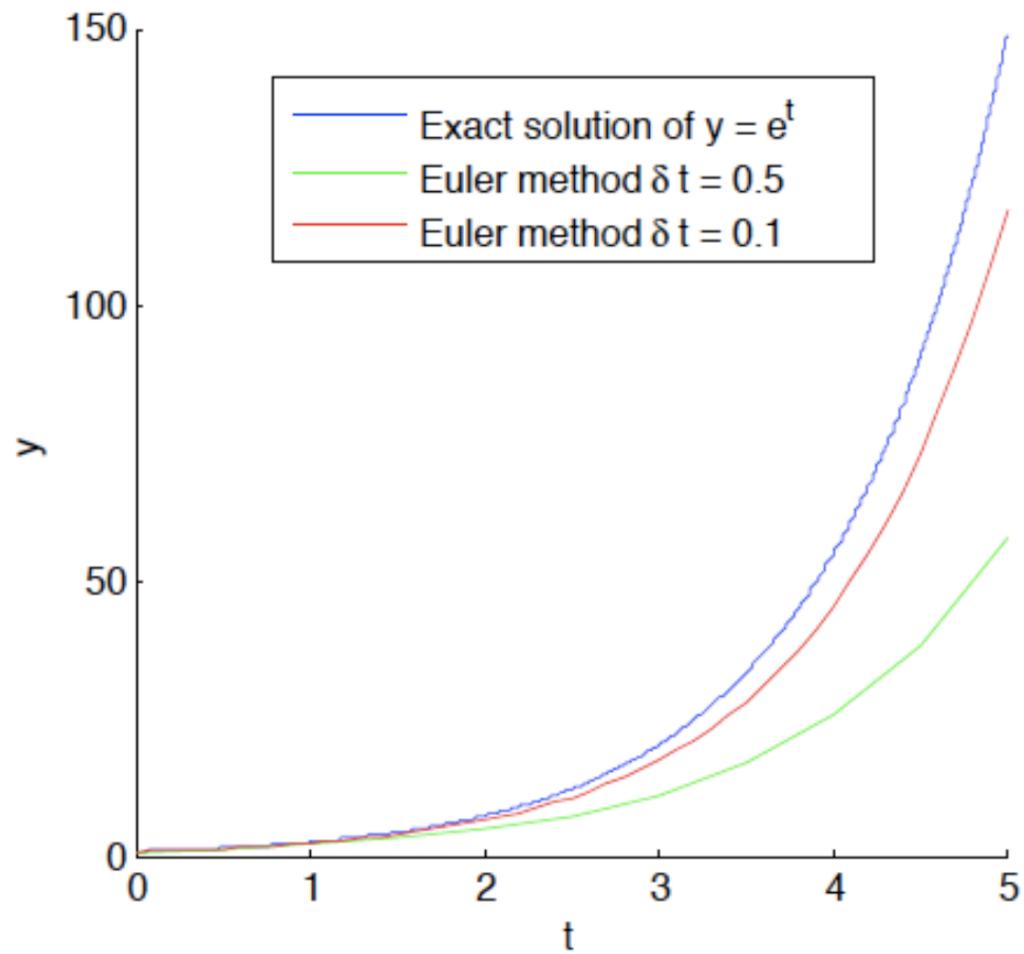
$$y(t) = e^t$$

for which we have

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

giving

$$y(t + \delta t) = y(t) + \delta t\, y(t)$$



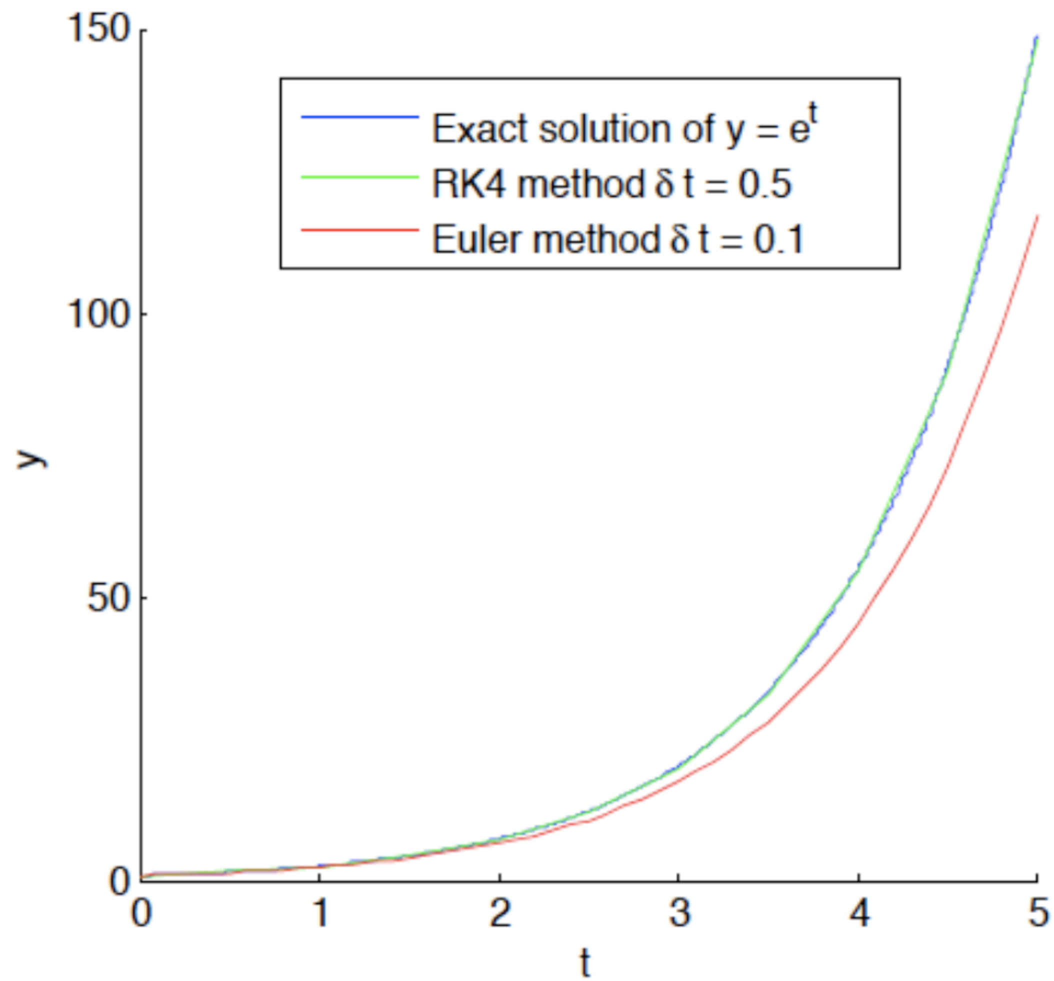
Euler Python Code

```
import numpy as np

# Define parameters
f = lambda t, y: y # ODE
dt = 0.1 # Step size
T = 5 # Simulation duration
t = np.arange(0, T + dt, dt) # Numerical grid
y0 = 1 # Initial Condition

# Explicit Euler Method
y = np.zeros(len(t))
y[0] = y0

for i in range(0, len(t) - 1):
    y[i + 1] = y[i] + dt*f(t[i], y[i])
```

- This is a simple Python script implementing the Euler method
- First, variables are defined and arrays pre-allocated
- Then, the solution is computed iteratively

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

$$k_1 = f\big(y(t)\big)$$
$$k_2 = f\left(y(t) + \tfrac{1}{2}\delta t\, k_1\right)$$
$$k_3 = f\left(y(t) + \tfrac{1}{2}\delta t\, k_2\right)$$
$$k_4 = f(y(t) + \delta t\, k_3)$$

- Then we have

$$y(t+\delta t) = y(t) + \frac{1}{6}\delta t(k_1 + 2k_2 + 2k_3 + k_4)$$

Related Reading

Press, W. et al. (2007). *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press.