Numerical integration of molecular dynamics

Exercise 4, BNF079, 2005

Supervisor: Pontus Melke
(pontus@thep.lu.se, 046-2229079)

Introduction

Biological networks are complex and involves multiple molecular players. It is often impossible to achieve analytical solutions and numerical integration of the systems are necessary to evaluate the dynamical behavior. We assume that we have a system of continous variables representing molecular concentrations, and that the interactions are defined deterministically resulting in a model that can be described by a system of ordinary differential equations (ODEs).

The goal of this exercise is to implement a numerical integrator in perl that can be used to solve systems ODEs given initial values of the variables. A first-order and a second-order solver are to be implemented and tested on two simple molecular networks. The second-order solver is then to be used in the final computer exercise.

Numerical integration

Consider a system of ODEs defined by

$$\frac{dx_i(t)}{dt} = f_i(x_1, ..., x_N), \qquad i = 1, ..., N \tag{1}$$

where the functional form of the time derivatives f_i are known functions of the variables x_i. Together with the initial values of the variables x_i, the future behavior of the system is perfectly defined.

Numerical solvers typically discretize Equation 1 by $dx \to \Delta x$ and $dt \to \Delta t$. Multiplying by Δt leads to the Euler update

$$x(t + \Delta t) = x(t) + \Delta t f(x(t)). \tag{2}$$

which is the first method we are going to implement. A power expansion of x around t shows that the error introduced by this update is of order Δt^2, and the method is called a first-order method (it is correct to $O(\Delta t)$). It is possible to cancel out higher order terms in the power expansion of x around t by evaluating the derivative at points in between t and $t + \Delta t$ and hence achieve more accurate solvers (higher-order) without knowing higher order derivatives of x as a function of t. This is utilized in the mid-point

(or second order Runge-Kutta) step which evaluates the derivative also at the mid-point $t + \frac{1}{2}\Delta t$ resulting in a second-order method

$$k_1 = \Delta t f(x_t) \tag{3}$$

$$k_2 = \Delta t f(x_t + \frac{1}{2}k_1) \tag{4}$$

$$x_{t+\Delta t} = x_t + k_2 \tag{5}$$

where the notation has been shortened by $x(t) \rightarrow x_t$. The algorithm for a step can in word be described by

1. Calculate the derivatives at time t.
2. Do a temporary update to $t + \Delta t$.
3. Calculate the derivatives using the varable values from 2.
4. Take a full step by using the derivatives calculated in 3.

More advanced solvers are out of the scope for this exercise, but more information can for example be found in the book "Numerical Recipes in C".

Molecular models

We will use the very simple molecular systems introduced in the first two examples in the lecture notes for testing the simulators.

$$\begin{aligned} \emptyset &\underset{d_1}{\overset{k_1}{\rightleftharpoons}} X, \\ A + Y &\underset{d_2}{\overset{k_2'}{\rightleftharpoons}} 2Y, \end{aligned} \tag{6}$$

Again, it is assumed that there is a surplus of A and this variable is not updated. Using concentration variables $x_1 = [X]$ and $x_2 = [Y]$ results in the ODEs

$$\frac{dx_1}{dt} = k_1 - d_1 x_1 \tag{7}$$

$$\frac{dx_2}{dt} = k_2 x_2 - d_2 x_2^2 \tag{8}$$

Exercises

In the exercises, the two different solvers are to be implemented. Since these programs are to be used and developed also for the final computer exercise it is good to be somewhat

general in the implementation and e.g. use a subroutine for calculating the derivatives which then can be changed depending on the model to be simulated. The time evoulution of the molecular concentrations are to be plotted during the exercises using e.g. gnuplot (if needed there is a short introduction to gnuplot in Appendix A).

Exercise 1 Implement a perl derivatives subroutine for the functions in Eqs. 7 and 8 It should take an array of variable values as input, and return an array with the calculated derivative values. Define all parameters as local perl variables within the subroutine and set their values to $k_1 = k_2 = d_1 = d_2 = 1.0$.

Exercise 2 Implement a numerical integrator that utilizes the Euler step for updating the variables and calls the derivatives subroutine from Exercise 1. Start time, end time, and time-step size should be defined as perl variables, together with an array holding the initial variable values. The solver should also print the time together with the variable values at each time step.

Exercise 3 Use a start time equal to 0, an end time equal to 10, and initial values $x_1 = 0.0$, and $x_2 = 4.0$. Simulate and plot the result for time-step sizes equal to 0.01, 0.1 and 0.5. Is the solutions accurate?

Exercise 4 Extend the simulator to use a stepper utilizing the mid-point method. Redo the simulations in Exercise 3 with this simulator. Plot the results using both numerical methods in the same plot. Is there any difference between the methods. Do you prefer one in front of the other.

Additional exercises

In the lectures, a protein activation/deactivation cycle has been discussed. Assuming a Michaelis-Menten formalism for the activation/deactivation and a constant enzyme concentration for the deactivation, the resulting differential equation can be written as

$$\frac{dX}{dt} = -\frac{V_1 X}{K_1 + X} + \frac{V_2 I(1-X)}{K_2 + (1-X)} \tag{9}$$

where X is the active form of the protein, I is the 'input' enzyme mediating the activation, and V_1, V_2, K_1, and K_2 are parameters.

Exercise 3 Implement a derivative function for the protein activation/deactivation cycle model, where the 'input' enzyme is assumed to be a constant. Use $X = 0$ as initial condition, parameter values $V_1 = V_2 = 1.0$ and $K_1 = K_2 = K = \{0.1, 10.0\}$. For each K-value, run the system until equilibrium for a number of values for the input concentration I. Can you relate to the result presented in the lecture notes.

The methodology of using substeps to gain accuracy in the ODE solvers can be extended. A fourth order Runge-Kutta solver utilize a step of the form

$$k_1 = \Delta t f(x_t) \tag{10}$$
$$k_2 = \Delta t f(x_t + \frac{k_1}{2}) \tag{11}$$
$$k_3 = \Delta t f(x_t + \frac{k_2}{2}) \tag{12}$$
$$k_4 = \Delta t f(x_t + k_3) \tag{13}$$
$$x_{t+\Delta t} = x_t + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(\Delta t^5) \tag{14}$$

Exercise 4 Create a forth order Runge-Kutta solver and simulate some of the earlier networks.

A Gnuplot

Typically, it is easiest to save all plot commands within a file and then use *gnuplot file* te execute the plotting commands. Here follows some examples of useful plotting commands in gnuplot.

Setting a postscript file as output:

```
set term postscript
set out ‘‘file.ps’’
```

Defining and plotting a function:

```
f(x,a) = x+a
plot f(x,1) title ‘‘a=1’’, f(x,2) title ‘‘a=2’’
```

Plotting columns in a file (columns 2 and 3 in file *file.data*):

```
plot ‘‘file.data’’ using 2:3
```

Setting ranges and labels for the axes:
`set xrange [0:1]`
`set yrange [0:*]` (only lower bound set)
`set xlabel ‘‘time’’`