Computational modeling of the cell cycle

Part 4

Icahn School of Medicine at Mount Sinai

SBCNY

Outline

Implementing the 1993 Novak-Tyson model

Model equations and notation

Practical considerations

MATLAB scripts versus MATLAB functions

Using MATLAB's built-in ODE solvers

Novak-Tyson 1993 cell cycle model

7 ODEs for 7 molecular species

1. $$\frac{d}{dt}[\text{Cyclin}] = k_1 - k_2[\text{Cyclin}] - k_3[\text{Cyclin}][\text{Cdk}]$$

2. $$\frac{d}{dt}[\text{MPF}] = k_3[\text{Cyclin}][\text{Cdk}] - k_2[\text{MPF}] - k_{wee}[\text{MPF}] + k_{25}[\text{preMPF}]$$

3. $$\frac{d}{dt}[\text{preMPF}] = -k_2[\text{preMPF}] + k_{wee}[\text{MPF}] - k_{25}[\text{preMPF}]$$

4. $$\frac{d}{dt}[\text{Cdc25P}] = \frac{k_a[\text{MPF}]([\text{total Cdc25}]-[\text{Cdc25P}])}{K_a + [\text{total Cdc25}]-[\text{Cdc25P}]} - \frac{k_b[\text{PPase}][\text{Cdc25P}]}{K_b + [\text{Cdc25P}]}$$

5. $$\frac{d}{dt}[\text{Wee1P}] = \frac{k_e[\text{MPF}]([\text{total Wee1}]-[\text{Wee1P}])}{K_e + [\text{total Wee1}]-[\text{Wee1P}]} - \frac{k_f[\text{PPase}][\text{Wee1P}]}{K_f + [\text{Wee1P}]}$$

6. $$\frac{d}{dt}[\text{IEP}] = \frac{k_g[\text{MPF}]([\text{total IE}]-[\text{IEP}])}{K_g + [\text{total IE}]-[\text{IEP}]} - \frac{k_h[\text{PPase}][\text{IEP}]}{K_h + [\text{IEP}]}$$

7. $$\frac{d}{dt}[\text{APC}] = \frac{k_c[\text{IEP}]([\text{total APC}]-[\text{APC}])}{K_c + [\text{total APC}]-[\text{APC}]} - \frac{k_d[\text{PPase}][\text{APC}]}{K_d + [\text{APC}]}$$

Sible & Tyson (2007) *Methods* 41:238-247

Novak-Tyson 1993 cell cycle model

Constant values for many model parameters

Maximal rates (usually k_{cat}'s)

```
k1 = 1 ;
k3 = 0.005 ;
ka = 0.02 ;
kb = 0.1 ;
kc = 0.13 ;
kd = 0.13 ;
ke = 0.02 ;
kf = 0.1 ;
kg = 0.02 ;
kh = 0.15 ;
```

Michaelis Constants

```
Ka = 0.1 ;
Kb = 1 ;
Kc = 0.01 ;
Kd = 1 ;
Ke = 1 ;
Kf = 1 ;
Kg = 0.01 ;
Kh = 0.01 ;
```

Total protein concentrations

```
CDK_total = 100 ;
cdc25_total = 5 ;
wee1_total = 1 ;
IE_total = 1 ;
APC_total = 1 ;
PPase = 1 ;
```

Weighting parameters

```
v2_1 = 0.005 ;
v2_2 = 0.25 ;
v25_1 = 0.0085 ;
v25_2 = 0.085 ;
vwee_1 = 0.01 ;
vwee_2 = 1 ;
```

9. $k_{25} = V_{25}'\,([\text{Total Cdc25}] - [\text{Cdc25P}]) + V_{25}''\,[\text{Cdc25P}]$

10. $k_{wee} = V_{wee}'\,[\text{Wee1P}] + V_{wee}''\,([\text{Total Wee1}] - [\text{Wee1P}])$

11. $k_2 = V_2'\,([\text{Total APC}] - [\text{APC}]) + V_2''\,[\text{APC}]$

Model structure to solve an ODE system

Our initial MATLAB script will be structured as follows:

1) Define constants

2) Set time step, simulation time, etc.

3) Set initial conditions

4) A "for" loop to simulate evolution of time

   At each time step:

   write output if needed
   compute dX/dt
   compute X at the next time step

5) Plot and output results

Euler's method example

$$\frac{dx}{dt} = a - bx \qquad\qquad x(t=0) = c$$

Assume a=20, b=2, c=5

We can write simple MATLAB code to solve this numerically

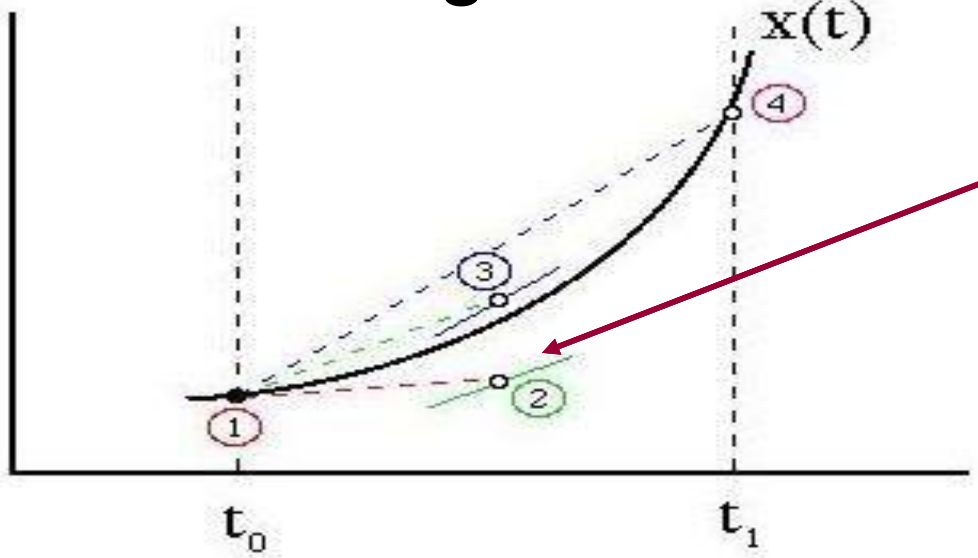
```
a = 20 ;
b = 2 ;
c = 5 ;

dt    = 0.05 ;
tlast = 2 ;

iterations = round(tlast/dt) ;
xall = zeros(iterations,1) ;

x = c ;
for i = 1:iterations
  xall(i) = x ;
  dxdt = a - b*x ;
  x = x + dxdt*dt ;
end % of this time step

time = dt*(0:iterations-1)' ;
figure
plot(time,xall)
```

euler.m

Other numerical methods for ODE systems

Euler died over 200 years ago – since then, improvements to his algorithm have been made

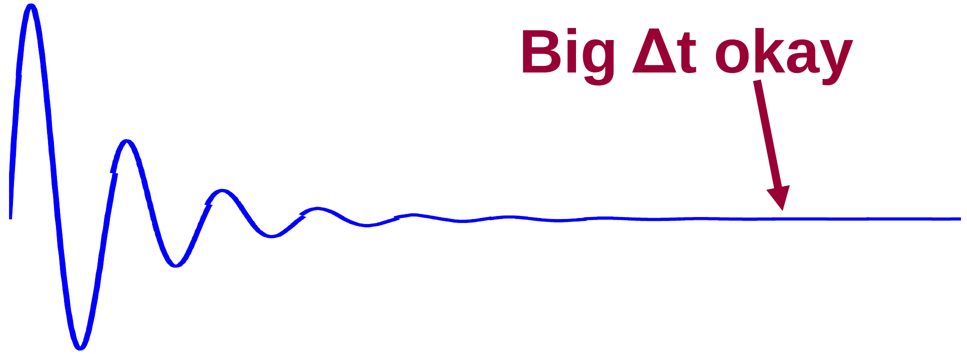
Runge-Kutta method



Also compute dx/dt between t_0 and t_1; use a weighted average

Variable time-step methods

Small Δt required



Big Δt okay

These algorithms are available in MATLAB as the built-in ODE solvers: ode 23, ode45, ode15s, ode23tb, etc.

Model structure to solve an ODE system

We will modify steps (2) and (4) to make the structure:

1) Define constants

2) Set ~~time step~~, simulation time, etc.

3) Set initial conditions

4) Use MATLAB's solvers to integrate the system

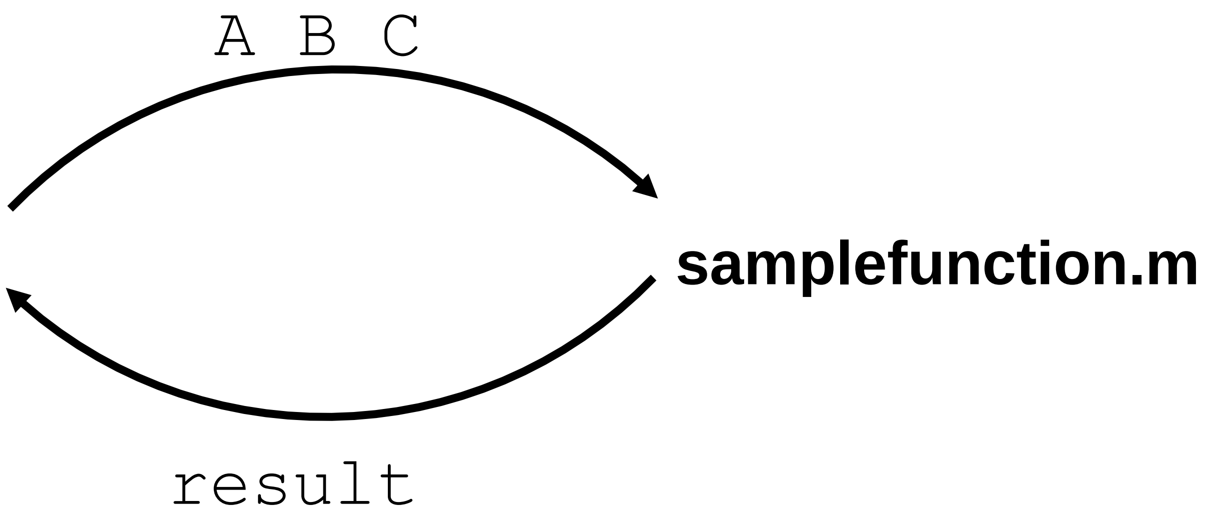
5) Plot and output results

This will require writing a function that, given a collection of state variables, computes the set of derivatives

MATLAB scripts versus functions

Schematic relationship between scripts and functions

```
result = samplefunction(A,B,C) ;
```




```
function a = samplefunction(x,y,z)

  LOTS OF IMPORTANT MATLAB CODE

  a = something ;

return
```

[A,B,C in script are called x,y,z in function, but this is okay]

[a in function will be called result in script, but this is okay]

After function is called, variables defined within function are gone.

MATLAB scripts versus functions

What does this have to do with solving ODEs?

1. **The MATLAB ode solvers, e.g. "ode23" are functions. To use them properly, you need to know what variables to pass to them.**

```
[time,statevars] = ode23(@dydt_novak,[0,tlast],statevar_i) ;
```

2. **To use the MATLAB ode solvers, you must create a function that computes the derivatives of your variables.**

```
function deriv = dydt_novak(t,statevar)
```

Solving ODEs with MATLAB functions

To solve the simple 1-variable ODE:

ode.m

```
global a b;
a = 20 ;
b = 2 ;
c = 5 ;

tlast = 4 ;

x0 = c ;
[t,x] = ode23(@dxdt,[0,tlast],x0) ;

figure
plot(t,x)
```

dxdt.m

```
function deriv = dxdt(t,x)
global a b ;
deriv = a - b*x ;
return
```

This is the general structure has is followed in the two files `novak.m` and `dydt_novak.m` that solve the Novak-Tyson model

Summary

One way to use MATLAB's built-in ODE solvers is to write a script that sets up the model and a function that computes the ODEs.

Parameters are generally defined in the script and used in the function, so it can be helpful to define these as `global` variables.