

1 Introduction

This is a step-by-step guide to perform *in silico* experiments and analysis on E-CELL 3 with a simulation model for *Drosophila* circadian rhythm. If you need detailed information about the simulation using E-CELL , please see the user's manual.

Note

Instructions given in this handout are written for Windows version of E-CELL 3. However, E-CELL 3 is originally developed and used on Linux platform. We recommend to use Linux version to get full function of the software.

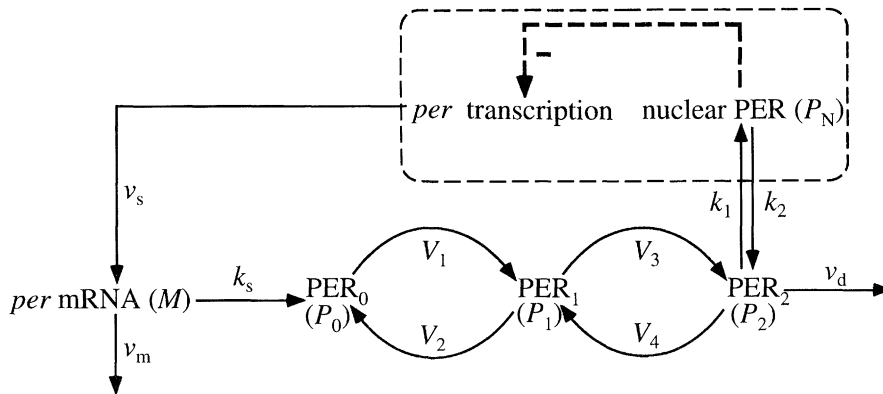
About the circadian rhythm model(Goldbeter1995)

This model was proposed by Dr. Alfred Goldbeter in 1995(Goldbeter, A., A model for circadian oscillations in the *Drosophila* period protein(PER). *Proc. R. Soc. Lond. B. Biol. Sci.* **261**:319-324, 1995). The main characteristics of the model is the negative feedback of *per* gene expression by its product, PER protein.

Reactions in this model is described by ordinary differential equations based on reaction kinetics. By numerical integration, or simulation of the model, we can predict states of the system at each time points.

Reaction scheme and kinetics are described below. You can come back to this section occasionally for reference.

Scheme of the model



(Goldbeter, A. *Proc. R. Soc. Lond. B. Biol. Sci.* **261**:319-324, 1995) Figure.1

1. Cytosolic *per* mRNA(denoted by M) is synthesized in the nucleus, transfers to the cytosol, where it is accumulated and degraded(Equation(1)).

- M transfer(accumulation) maximum rate: v_s
- M degradation maximum rate: v_m , Michaelis constant: K_m

2. PER protein(denoted by P_0) is synthesized proportionally to M (Equation(2)).
 - Rate: k_s
3. PER has three states of phosphorylation: unphosphorylated(P_0), mono-phosphorylated(P_1) and bisphosphorylated(P_2). Phosphorylation reactions of each state of the proteins occur sequentially from P_0, P_1 to P_2 , as dephosphorylation occur reversely (Equation(2)(3)(4)).
 - Vmax for P_0 phosphorylation: V_1 , Michaelis constant: K_1
 - Vmax for P_1 dephosphorylation: V_2 , Michaelis constant: K_2
 - Vmax for P_1 phosphorylation: V_3 , Michaelis constant: K_3
 - Vmax for P_2 dephosphorylation: V_4 , Michaelis constant: K_4
4. Fully phosphorylated form of PER(P_2) is then degraded, whereas some of them transfer to the nucleus (to be P_N) (Equation(4)(5)).
 - P_2 degradation Vmax: v_d , Michaelis constant: K_d
 - Rate for transfer of P_2 to nucleus(to be P_N): k_1
 - Rate for relocation of P_N to cytoplasm(to be P_2): k_2
5. The negative feedback exerted by P_N on *per* transcription is described by the Hill type equation (Equation(1)).
 - Hill constant: n , threshold value for the threshold of repression: K_I

Equations

$$\frac{dM}{dt} = v_s \frac{K_I^n}{K_I^n + P_N^n} - v_m \frac{M}{K_m + M} \quad (1)$$

$$\frac{dP_0}{dt} = k_s M - V_1 \frac{P_0}{K_1 + P_0} + V_2 \frac{P_1}{K_2 + P_1} \quad (2)$$

$$\frac{dP_1}{dt} = V_1 \frac{P_0}{K_1 + P_0} - V_2 \frac{P_1}{K_2 + P_1} - V_3 \frac{P_1}{K_3 + P_1} + V_4 \frac{P_2}{K_4 + P_2} \quad (3)$$

$$\frac{dP_2}{dt} = V_3 \frac{P_1}{K_3 + P_1} - V_4 \frac{P_2}{K_4 + P_2} - k_1 P_2 + k_2 P_N - v_d \frac{P_2}{K_d + P_2} \quad (4)$$

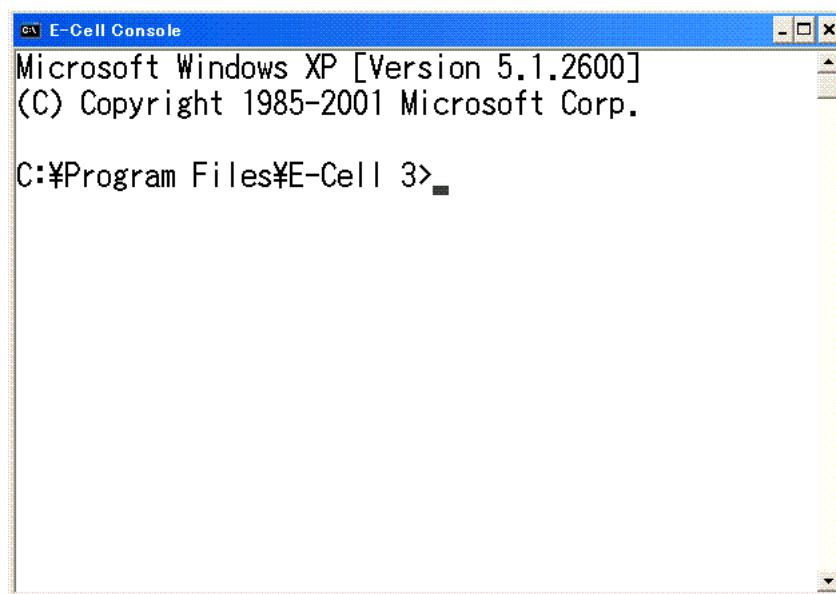
$$\frac{dP_N}{dt} = k_1 P_2 - k_2 P_N \quad (5)$$

2 Hands-on 1: Run simulation and save simulated time course

The purpose of this section is to introduce the most basic functions of the E-CELL software. You will invoke a single simulation for the circadian rhythm model and see saved time-course data.

1. Copy tutorial material files to your PC.
 - (a) Doubleclick 'My Computer'.
 - (b) Move to C:\Program Files\E-Cell3 folder by doubleclicking folder icons.
 - (c) Copy 'tutorial' folder in the USB flash memory into C:\Program Files\E-Cell3 folder.
2. Doubleclick 'E-Cell console' icon on Desktop.

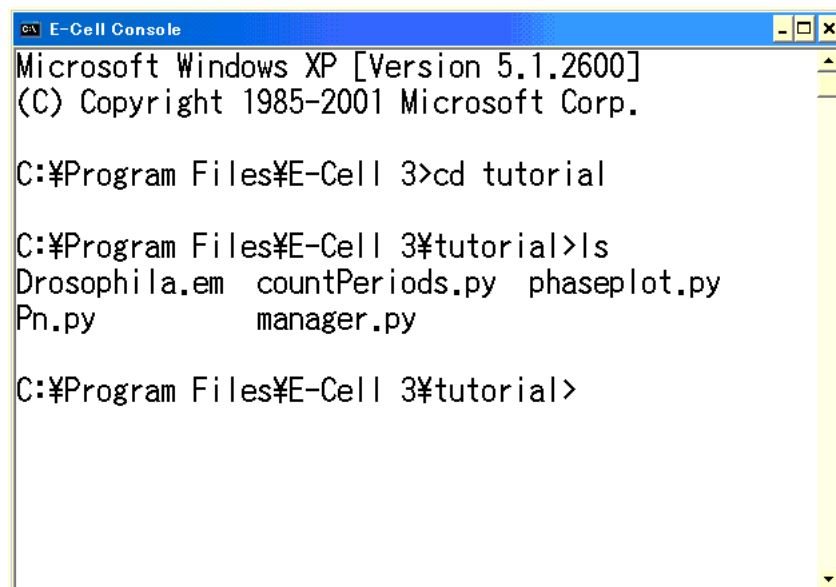
'E-Cell console' window will appear.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Program Files\E-Cell 3>
```

3. Move to the folder you extracted downloaded file on the E-CELL console by typing:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

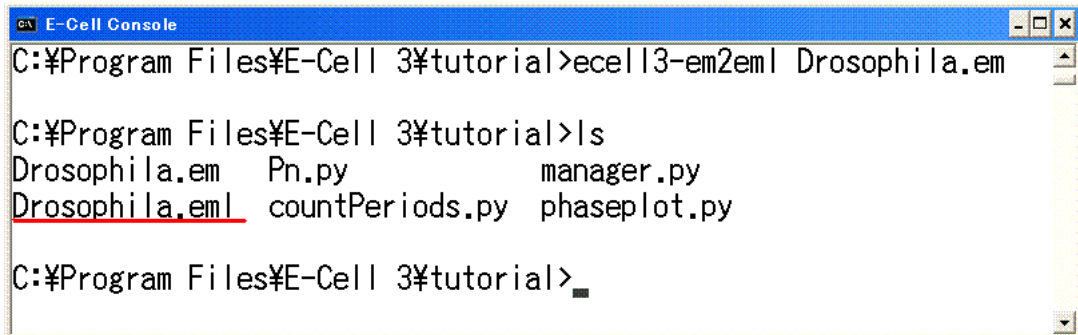
C:\Program Files\E-Cell 3>cd tutorial

C:\Program Files\E-Cell 3\tutorial>ls
Drosophila.em  countPeriods.py  phaseplot.py
Pn.py          manager.py

C:\Program Files\E-Cell 3\tutorial>
```

Drosophila.em is a E-CELL model file (with extension .em), the file type users usually edit with text editor.

4. Convert the model file (Drosophila.em).



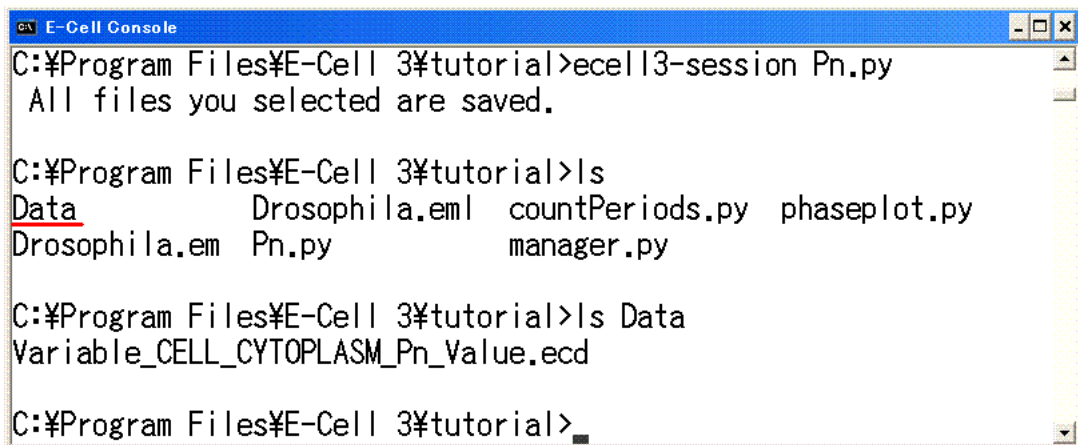
```
C:\Program Files\E-Cell 3\tutorial>ecell3-em2eml Drosophila.em

C:\Program Files\E-Cell 3\tutorial>ls
Drosophila.em  Pn.py          manager.py
Drosophila.eml countPeriods.py phaseplot.py

C:\Program Files\E-Cell 3\tutorial>
```

The file 'Drosophila.eml' will appear. This is a converted E-CELL model file (with extension .eml) that is loadable on E-CELL 3.

5. Run simulation.



```
C:\Program Files\E-Cell 3\tutorial>ecell3-session Pn.py
All files you selected are saved.

C:\Program Files\E-Cell 3\tutorial>ls
Data          Drosophila.eml countPeriods.py phaseplot.py
Drosophila.em Pn.py          manager.py

C:\Program Files\E-Cell 3\tutorial>ls Data
Variable_CELL_CYTOPLASM_Pn_Value.ecd

C:\Program Files\E-Cell 3\tutorial>
```

One simulation is called 'session' on E-CELL . The command 'ecell3-session' invokes single session according to the instruction written in 'Pn.py' file.

This script file is called 'E-CELL session script (ESS) file' which is written using the Python programming language. (Python language is easy to learn, object-oriented programming/scripting language, often mentioned as one of the best way to learn computer programming in general.)

An ESS is used for automating procedures for a single simulation session.

When the simulation finished, simulated data will be saved under 'Data' folder.

6. Graph the simulated time-course.

- (a) Open any spreadsheet software you usually use, like Microsoft Excel.

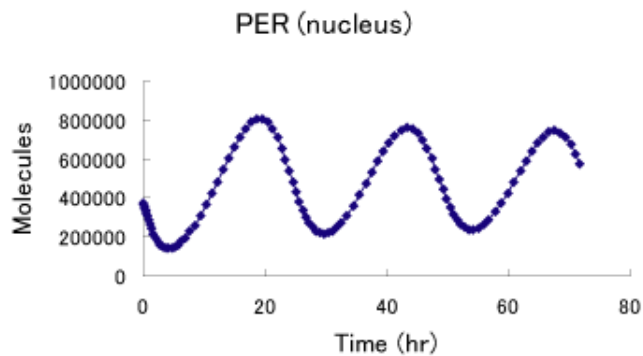
- (b) Select 'File' → 'Open':

```
C:\Program Files\E-Cell3\tutorial\Data\Variable_Pn_Value.ecd'
```

	A	B	C	D	E
1	#DATA: Variable:/CELL/CYT OPLASM:Pn:Value				
2	#SIZE: 2 105				
3	#LABEL: t	value	avg	min	max
4	#NOTE:				
5	#				
6	#-----				
7	0.00E+00	3.74E+05			
8	1.00E-03	3.74E+05			
9	2.00E-03	3.74E+05			
10	1.00E-02	3.73E+05			
11	7.40E-02	3.68E+05			
12	1.82E-01	3.59E+05			
13	2.89E-01	3.49E+05			
14	3.97E-01	3.38E+05			
15	5.37E-01	3.24E+05			
16	6.78E-01	3.10E+05			
17	8.70E-01	2.91E+05			
18	1.11E+00	2.69E+05			
19	1.41E+00	2.43E+05			
20	1.72E+00	2.19E+05			
21	2.09E+00	1.95E+05			
22	2.45E+00	1.76E+05			
23	2.82E+00	1.62E+05			
24	3.30E+00	1.49E+05			
25	3.78E+00	1.43E+05			
26	4.25E+00	1.42E+05			
27	4.73E+00	1.45E+05			

E-CELL uses the ECD (E-CELL Data) file format to store simulation results. ECD is a plain text of tab separated values file in which time course data of a simulation object is written as tab separated values. It can be handled by plotting softwares like 'gnuplot', or spreadsheet softwares like Excel, mathematical softwares like 'GNU Octave'.

(c) Graph a time-course of PER in nucleus (called 'Pn' in the model).



Note

The model used here is included in E-CELL 3 package. On your computer, you can find some more sample models in the folder below:

C:\Program Files\E-Cell3\share\doc\sample

3 Taking a glance at E-CELL session script

Here we will look over what is written in the ESS file 'Pn.py' to see basics of E-CELL Session Script. The purpose of this section is to introduce basic scripting methods for automating a single simulation.

The file we loaded from 'ecell3-session' in the previous section is called 'E-CELL Session Script (ESS)'. ESS file is written using the Python code. It defines how a simulation is to be executed.

This function is useful when conducting *in silico* experiments as protocols can be automated and each procedure logged. The logged procedure can then be reused for repeating the experiment and for reproducing the same results.

Session scripts are particularly important when multiple simulations need to be run and analyzed. This is the case for parameter estimation, where several simulations are necessary and perturbations are applied to several variables. Depending on the state of that variable, perturbation may need to be repeated. Using the scripting methods, the execution of all these processes can be automated.

To automate a simulation using ESS, you will need:

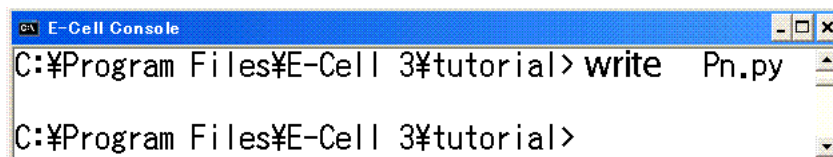
EML: An E-Cell Model (ex. Drosophila.eml)
ESS: A script for E-Cell Session (ex. Pn.py)

Note that the suffix for an E-CELL session script is '.py'.

Although ESS manages only one session, the E-Cell Session Manager (ESM) is capable of managing multiple session scripts. Unfortunately, since E-CELL 3 is basically developed and used on Linux platform, ESM is not yet implemented on Windows version. Alternatively, we will manage to do batch simulation using a simple script later in this tutorial.

Open sample ESS script

Now let's see what is written in the ESS script we used. This is a very simple example. To write more complex script, please see E-CELL user's manual Chapter 5. Open 'Pn.py' by text editor you usually use. For example, in the E-CELL console you are working, type:



```
C:\Program Files\E-Cell 3\tutorial> write Pn.py
C:\Program Files\E-Cell 3\tutorial>
```

If it doesn't work, please check if you are on the tutorial folder:

```
C:\Program Files\E-Cell3\tutorial
```

Pn.py

```
loadModel ( 'Drosophila.eml' )
# Load 'model_filename'.
# The loadable format is '.eml'(XML format of E-CELL model file).

Pn_logger = createLoggerStub( 'Variable:/:Pn:Value' )
Pn_logger.create()
# Commands to make an object that records amount of Pn molecules
# over time(called 'Logger').

run( 72 )
# Run simulation for 72 seconds.

stop()
# Stop simulation.

saveLogeerData()
# Save data recorded on Logger objects into datafiles.
```

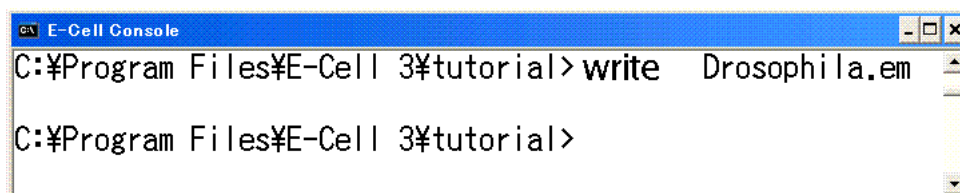
Note

The lines starts with # are comment lines that doesn't have any effect as a python code.

4 Taking a glance at E-CELL model file

Now let's look into 'Drosophila.em' which contains information of the model. The model file we loaded using 'Pn.py' script is 'Drosophila.eml'. It is a file converted from Drosophila.em, which we usually edit using a text editor. To see more information about how to write .em, please see E-CELL user's manual Chapter 4.

Open 'Drosophila.em' by text editor you usually use. For example, in the E-CELL console you are working, type:



```
E-Cell Console
C:\Program Files\E-Cell 3\tutorial>write Drosophila.em
C:\Program Files\E-Cell 3\tutorial>
```

Stepper statement

Stepper The stepper class specifies the simulation algorithm to be used for calculation. Users can select from steppers such as generic ordinary differential equation solvers, differential algebraic equation solvers. Multi-stepper simulations may also be conducted where more than one Stepper object is used.

```
Stepper ODEStepper( DE )
# defining a Stepper in the model
{
# no property
}
```

System statement

System The system class defines the overall structure of the model. A System object can contain other systems to form tree-like structures to define locations of variables and processes. For example, in the case of cell components, System may be used to define compartments such as the cytoplasm and membrane.

```
System System( / )
{
  StepperID DE;
  # Define that by default, Stepper 'DE' is used for numerical
  integration # of Processes(the objects correspond to reactions in
  the model)
  # in this System.

  Variable Variable( SIZE ) { Value @(VOLUME); }
  # SIZE is a special Variable(the objects usually correspond to
  # substances in the model) that indicates system volume.
}
```

Note

You can also define other compartment adding 'System' statements. Only one System is defined in this model for simplicity. Substances (called 'Variable' in .em files) in Figure.1 and all the reactions (called 'Process' in .em files) are put into this System ('/', the root System). These Variables and Processes are written within the System statement block. Note that the closed parenthesis of '/' System statement is put on the last line of the file.

Variable statement

You can describe molecular species in the model by name and the initial value.

Variable A Variable object holds a scalar real-number value. For example, if a variable represents a substrate in a metabolic pathway, the value can be defined in terms of molecular concentration or quantity.

```
Variable Variable( M ){
  Value @(3.61328202E-01 * N_A * VOLUME);
}

# The initial value of M is 3.61328202e-1 mol/liter.
# In E-CELL , 'Value' corresponds to number of molecules.
# Here, molar concentration of 'M' is converted
# to number of molecules (value) by multiplying
# Avogadro number and the Volume.
# Note that parameter 'N_A' and 'VOLUME' is defined
# at the first line of the .em file.

# You can also use concentration(unit:mol/liter) for
# Variable statement by typing like:
# MolarConc 3.0e-6;
```

Note

Script lines written in '@()' parenthesis are evaluated when .em files are converted to .eml files.

Process statements

You can describe reactions in the model by name and reaction kinetics. The reaction defined here is equivalent with the first term of equation(1), where M is accumulated.

Process Process objects represent phenomena in the simulation that result in changes in the values of one or more Variable objects. The definition of each process is defined by its Class. Process classes can be created by implementing rate equations, or by using Python or C++ programming languages. Some default classes are also available, such as the MassActionFluxProcess and MichaelisUniUniFluxProcess. The ExpressionFluxProcess used in this example is a special Process that users can kinetic equations directly in .em files.

```

Process ExpressionFluxProcess( R_toy1 ){
  vs 0.76;  # define parameter 'vs' and set its value
  KI 1;    # define parameter 'KI' and set its value

  VariableReferenceList
  [ P0 Variable:/:M 1 ]
  [ C0 Variable:/:Pn ];
  # Define which Variables are related to the reaction.
  # [ Identifier in the kinetic equation path:
  # Variable name:stoichiometric coefficient ]
  # Note that ';' is needed at the end of the list.

  Expression "( ( vs * KI * KI * KI ) / ( KI * KI * KI + ( C0.MolarConc
* C0.MolarConc * C0.MolarConc ) ) * self.getSuperSystem().SizeN_A )";
  # Write kinetic equation in Expression statement.
  # 'C0.MolarConc denotes' concentration of Pn Variable.
  # (The identifier 'C0' is made correspondent with
  # 'Variable:/:Pn' by VariableReferenceList.)
  # In E-CELL , velocity of the Process should be molecules per second.
  # The last element of the Expression is for unit conversion
  # from M/second to molecules/second.
}

```

Note

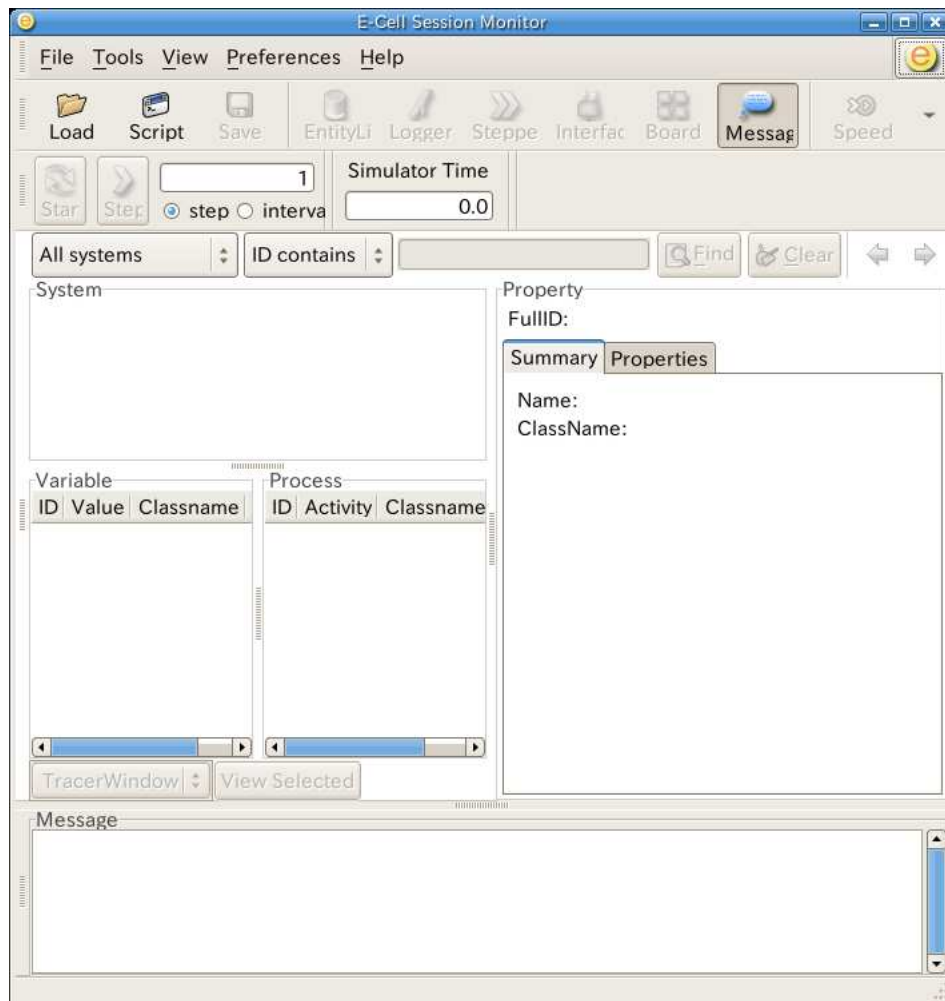
As mentioned before, You can create Process classes by implementing rate equations. You can define the class instances in .em file. In this case, you don't need to write expressions in the .em file, just need to give parameters for the Process instance. Using user-defined Process class will make calculation time of the simulation faster and save your time to write the same expression repeatedly. See user's manual Chapter 6 for detail.

5 Optional hands-on: Using ecell-session-monitor(GUI mode)

You can do the same thing as the previous section with a graphic user interface. With GUI, you can load model, run/stop simulation, make interference if you need, and save time-course data.

GUI mode is rather slow and clumsy way to see model behavior, but it is useful for debugging, when you are building or implementing a model with E-CELL .

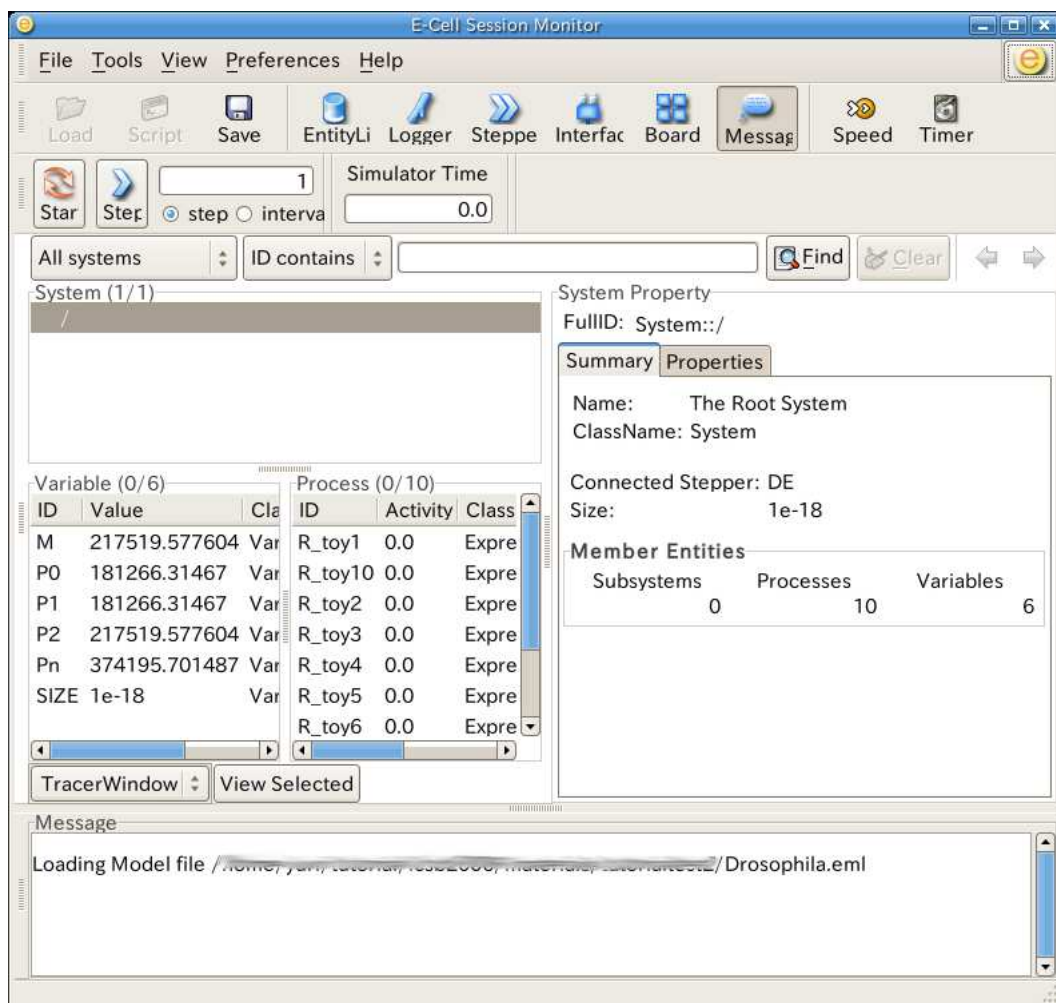
1. To invoke a simulation session with GUI mode, doubleclick 'E-Cell Session Monitor' icon on Desktop.



2. Loading a simulation file.

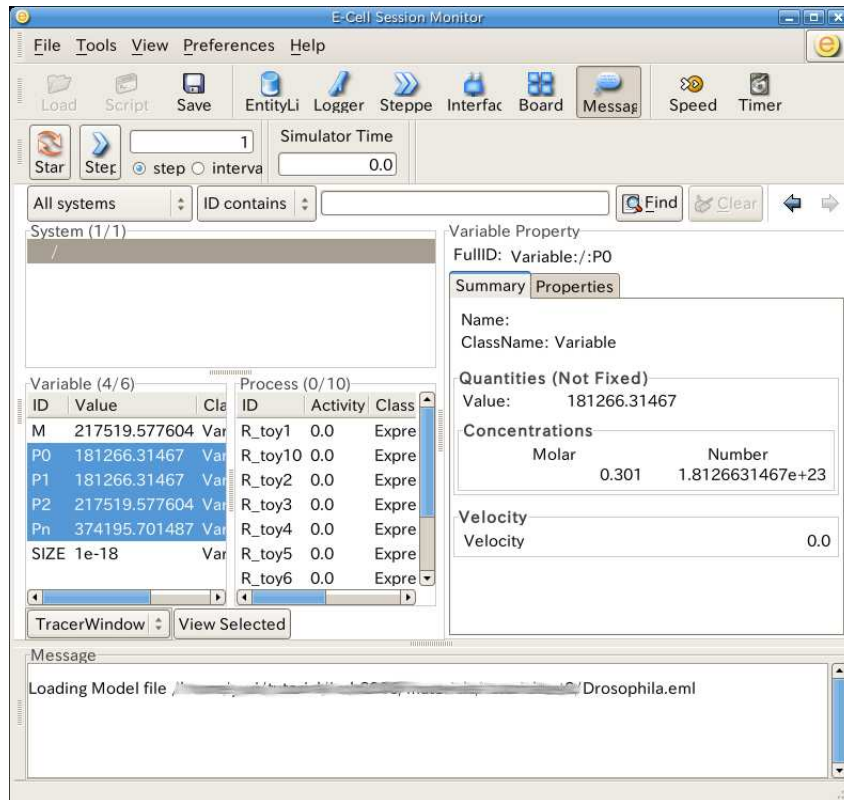
- Click 'Load' button.
- Double-click 'Drosophila.eml' from the list of Files.
- Click 'OK'

The Drosophila model has now been loaded into the simulator. The middle-left pane of the window shows 'EntityList', which lists the Systems, Variables and Processes in the loaded model.

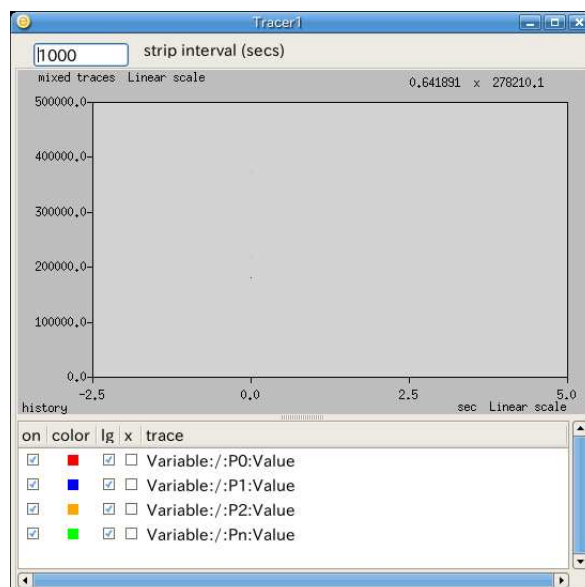


3. Setting a tracer window

- Select the variables to be displayed in the tracer. While pressing the Shift-key, select: P0, P1, P2, Pn

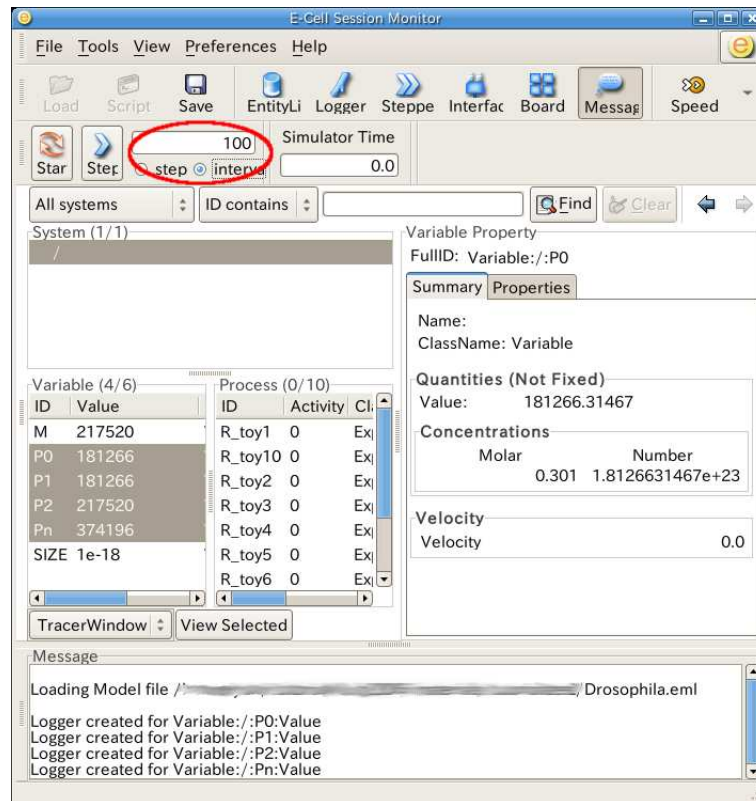


- Click 'View Selected'.
Variables selected in the Entity List Window will be displayed in a Tracer Window.

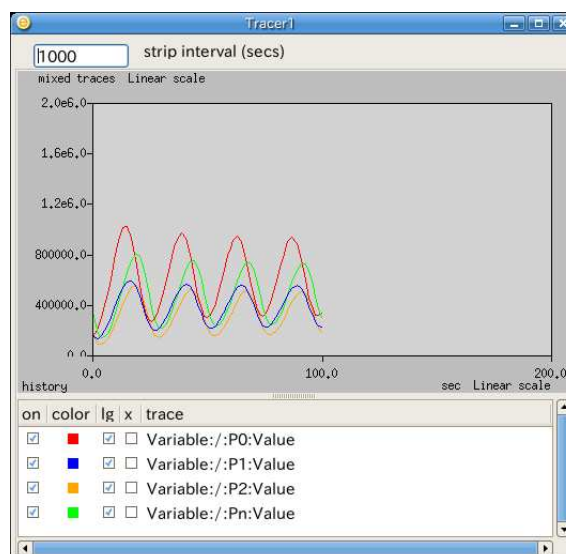


4. Running a simulation

- Click the textfield on the right of 'Start' 'Stop' button and input how long would you like to run the simulation, select 'interval' on the radio button below the textfield. (or you can just hit 'Start' button and press 'Stop' when you like. You can do it repeatedly.)



- Click 'Step' button. It will run simulation for time you assigned.



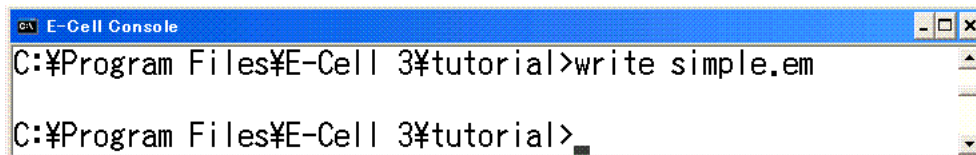
- 5. Close the session monitor window by selecting File → Exit.

6 Hands-on 2: Building the simplest model

You can write a simple model now. Let's work on the hands-on exercise below.

Writing simple.em

1. Open a new file by a text editor to write an E-CELL model file. For example, on E-CELL console, type:



```
E-Cell Console
C:\Program Files\E-Cell 3\tutorial>write simple.em
C:\Program Files\E-Cell 3\tutorial>
```

2. Write definition of one Stepper named 'DE'..

- Stepper name: DE
- Stepper class: ODEStepper

```
#
# simple.em
#

Stepper ODEStepper( DE )
{
    # no property
}
```

3. Write definition of one System named '/'(Root system).

- System name: /
- System class: System
- Default stepper used in the System: DE

```
#
# simple.em
#

Stepper ODEStepper( DE )
{
    # no property
}

System System( / )
{
    StepperID    DE;

}
}
```

4. Write the 'SIZE' Variable in the root System to assign the volume.

- Variable class: Variable
- Variable name: SIZE
- Initial value: 1.0e-15 (unit: liter)


```
#
# simple.em
#

Stepper ODEStepper( DE )
{
    # no property
}

System System( / )
{
    StepperID    DE;

    Variable Variable( SIZE )
    {
        Value    1.0e-15;
    }

}
}
```



5. Write a Variable statement to put a substance 'S' into the root System.

- Variable name: S
- Initial value: 1.0×10^6 (unit: particle)

```
#
# simple.em
#
Stepper ODEStepper( DE )
{
    # no property
}

System System( / )
{
    StepperID    DE;

    Variable Variable( SIZE )
    {
        Value    1.0e-15;
    }

    Variable Variable( S )
    {
        Value    1.0e6;
    }
}
```

6. Write a Process statement so that substance 'S' is changed by time depending on the equation below:

- Process class: ExpressionFluxProcess
- Process name: Reaction1
- VariableReferences: [S0: Variable:/:S -1];
- Equation: $\frac{d[S]}{dt} = -k[S]$
- Parameter: $k(0.2\text{sec}^{-1})$

```
#
# simple.em
#

Stepper ODEStepper( DE )
{
    # no property
}

System System( / )
{
    StepperID    DE;

    Variable Variable( SIZE )
    {
        Value    1.0e-15;
    }

    Variable Variable( S )
    {
        Value    1.0e6;
    }

    Process ExpressionFluxProcess( Reaction1 )
    {
        VariableReferenceList    [ S0 Variable:/:S -1 ];

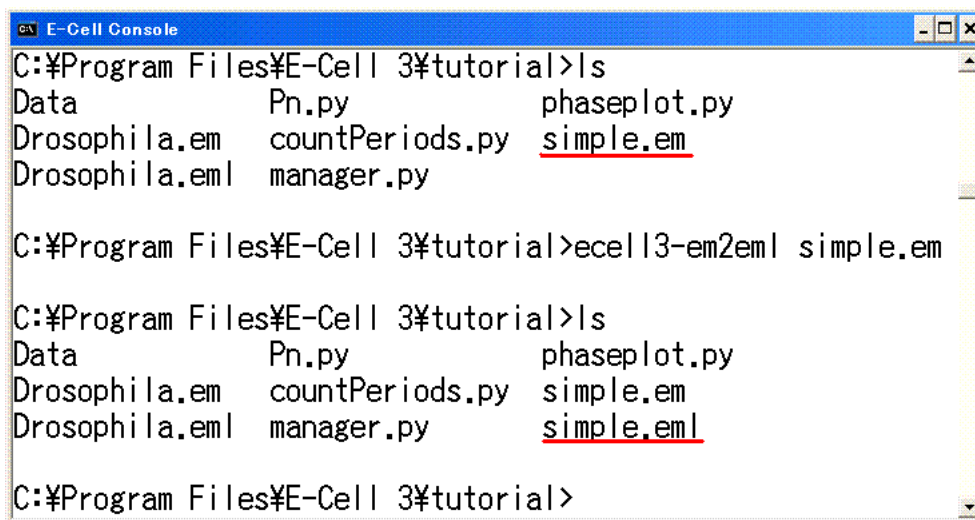
        k        0.2;

        Expression "( k * S0.Value )";
    }
}

}
```

7. Save the file as 'simple.em' and close the file.

- Convert 'simple.em' to .eml format by typing:



```
C:\Program Files\E-Cell 3\tutorial>ls
Data          Pn.py          phaseplot.py
Drosophila.em countPeriods.py simple.em
Drosophila.eml manager.py

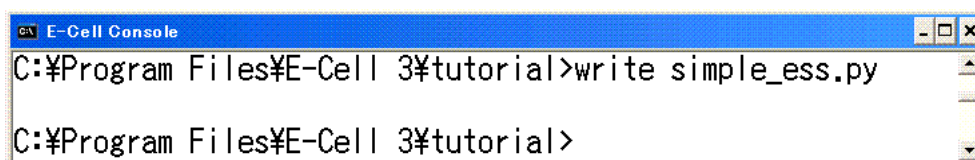
C:\Program Files\E-Cell 3\tutorial>ecell3-em2eml simple.em

C:\Program Files\E-Cell 3\tutorial>ls
Data          Pn.py          phaseplot.py
Drosophila.em countPeriods.py simple.em
Drosophila.eml manager.py    simple.eml

C:\Program Files\E-Cell 3\tutorial>
```

Writing simple ESS

- Open a new file named 'simple_ess.py' by a text editor to write an E-CELL session script. For example, on E-CELL console, type:



```
C:\Program Files\E-Cell 3\tutorial>write simple_ess.py

C:\Program Files\E-Cell 3\tutorial>
```

- Write a 'loadModel' statement to load 'simple.eml'.

```
loadModel( 'simple.eml' )
```

- Create Logger for defined Variable 'S'.

```
loadModel( 'simple.eml' )

S_logger = createLoggerStub( 'Variable:/:S:Value' )
S_logger.create()
```

4. Write a line to run 30 seconds simulation.

```
loadModel( 'simple.eml' )

S_logger = createLoggerStub( 'Variable://S:Value' )
S_logger.create()

run( 30 )
stop()
```

5. Write a line to save time-course data of Variable 'S'.

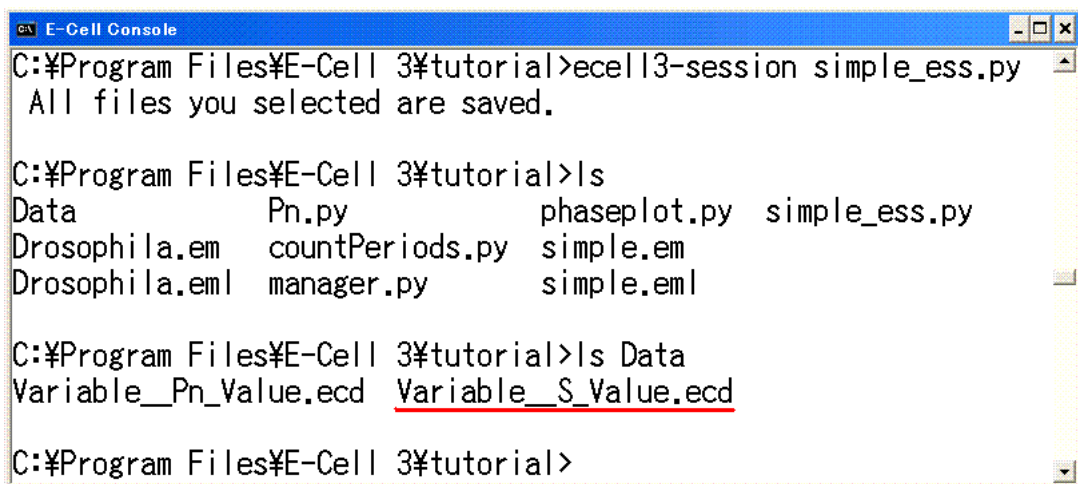
```
loadModel( 'simple.eml' )

S_logger = createLoggerStub( 'Variable://S:Value' )
S_logger.create()

run( 30 )
stop()

saveLoggerData()
```

6. Save and close the file.
7. Run simulation using the session script.



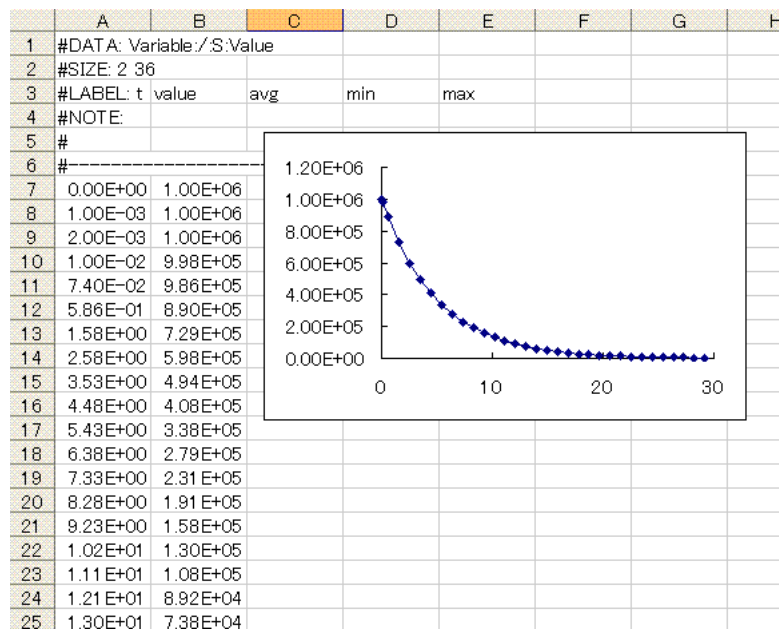
```
E-Cell Console
C:\Program Files\E-Cell 3\tutorial>ecell3-session simple_ess.py
All files you selected are saved.

C:\Program Files\E-Cell 3\tutorial>ls
Data          Pn.py          phaseplot.py  simple_ess.py
Drosophila.em countPeriods.py simple.em
Drosophila.eml manager.py     simple.eml

C:\Program Files\E-Cell 3\tutorial>ls Data
Variable_Pn_Value.ecd Variable_S_Value.ecd

C:\Program Files\E-Cell 3\tutorial>
```

8. Graph time-course of Variable 'S' by saved simulation data.



7 Stub: operating Variables and Processes

In this section, you will be introduced to a new object type called Stub. Using Stub object, you can get the amount of each Variable objects, activities of Process instance, and set these values during the simulation at any time point. The explanation in detail is in Chapter 5 of the E-CELL user's manual.

Here we show an example of Stub usage with reference to a E-CELL session script 'phaseplot.py', which can be used to make a phase-space plot.

Open the file typing:

```

E-Cell Console
C:\Program Files\E-Cell 3\tutorial>write phaseplot.py
C:\Program Files\E-Cell 3\tutorial>

```

phaseplot.py

```
loadModel( 'Drosophila.eml' )
# Load Drosophila model.

Pn_Stub = createEntityStub( 'Variable://Pn' )
# Stub for Variable Pn(PER in the nucleus).

M_Stub = createEntityStub( 'Variable://M' )
# Stub for Variable M(per mRNA).

R_toy10_Stub = createEntityStub( 'Process://R_toy10' )
# Stub for R_toy10 reaction (degradation of P2)

Pn_Stub.setProperty( 'Value', 6.21367e-1 )
# Setting Value of Pn to 6.21367e-1.
# It's just a demonstration to show the usage of 'setProperty' method.

R_toy10_Stub.setProperty( 'vd', 0.95 )
# It sets value of parameter vd in R_toy10 Process to 0.95.
# It's just a demonstration to show the usage of 'setProperty' method.

while getCurrentTime() < 72.0:
# Repeat quoted statements until 72 second in simulation time.
    Pn = Pn_Stub.getProperty( 'Value' )
    # value Pn at the timepoint.

    M = M_Stub.getProperty( 'Value' )
    # value of M at the timepoint.

    R_toy10 = R_toy10_Stub.getProperty( 'Activity' )
    # activity of R_toy10 at the timepoint.

    print Pn, '\t', M, '\t', R_toy10
    # print value of Pn, M, R_toy10 separated by tab characters.

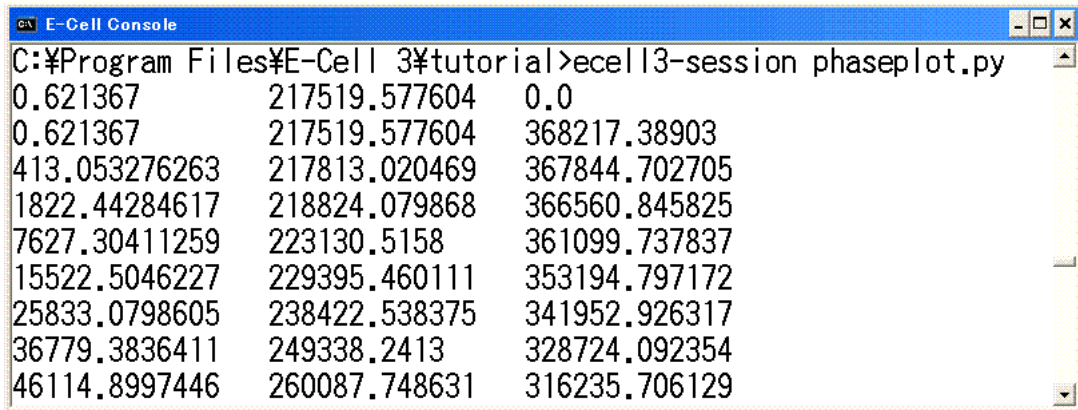
    step()
    # run simulation for one step(in default, 1ms).

stop()
# Stop the simulation.
```

Running phaseplot.py

Then run a simulation using this ESS script.

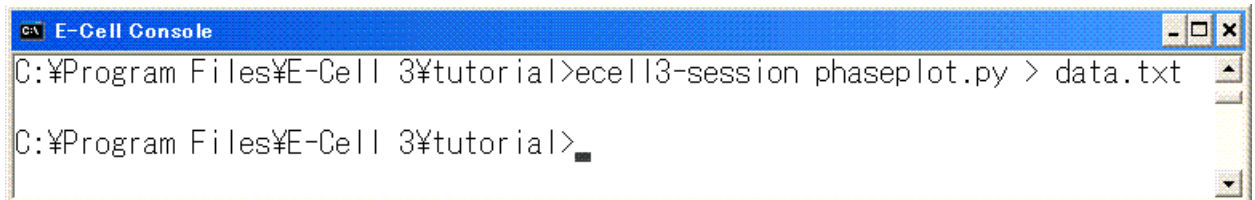
1. Invoke ecell3-session command and load phaseplot.py.



```
C:\Program Files\E-Cell 3\tutorial>ecell3-session phaseplot.py
0.621367      217519.577604    0.0
0.621367      217519.577604    368217.38903
413.053276263 217813.020469    367844.702705
1822.44284617 218824.079868    366560.845825
7627.30411259 223130.5158      361099.737837
15522.5046227 229395.460111    353194.797172
25833.0798605 238422.538375    341952.926317
36779.3836411 249338.2413      328724.092354
46114.8997446 260087.748631    316235.706129
```

'phaseplot.py' prints a line which contains tab separated values of Pn value, M value and R_{toy10} activity at each simulation step.

2. Save these lines by typing:

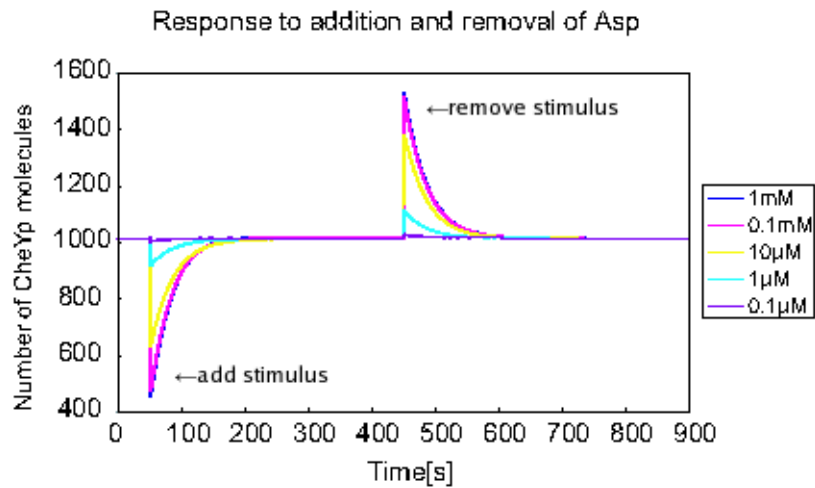


```
C:\Program Files\E-Cell 3\tutorial>ecell3-session phaseplot.py > data.txt
C:\Program Files\E-Cell 3\tutorial>
```

3. Make a phase-space plot by 'data.txt' with any software you usually use to graph. For example:
 - (a) Open any spreadsheet software you usually use, like Microsoft Excel.
 - (b) Select 'File' → 'Open':
C:\'Program Files\E-Cell3\tutorial\data.txt'
 - (c) Plot.

Note: Interference during the simulation

One of the features of E-CELL that is not mentioned directly in this tutorial is interference during the simulation. By writing ESS, you can easily change properties of simulated objects during the simulation. This can be useful in many cases. For example, in case you need to analyse dynamic properties of the model system responding to stimulation like below:

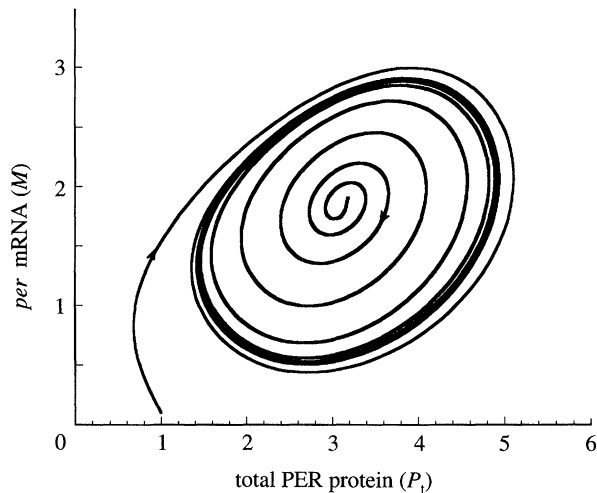


Simulated time courses of bacterial chemotaxis signal protein, phospho-CheY

Instructions about these usage of E-CELL are in Chapter1, 4 of the E-CELL tutorial book.

8 Hands-on 2: Conducting similar experiment with Goldbeter1995, Figure 3

In this section, you will do the similar experiment described in Goldbeter1995, Figure3 using Stubs introduced in the previous section.



(Goldbeter, A. *Proc. R. Soc. Lond. B. Biol. Sci.* **261**:319-324, 1995) Figure.3

1. Copy 'phaseplot.py' to a file named 'fig3.py' and open 'fig3.py' with a text editor.

```
E-Cell Console
C:\Program Files\E-Cell 3\tutorial>cp phaseplot.py fig3.py
C:\Program Files\E-Cell 3\tutorial>write fig3.py
C:\Program Files\E-Cell 3\tutorial>
```

2. Modify the script to print values of P_t and M separated by a tab character at each logging step of the simulation. As Goldbeter1995, Figure3 plots P_t (sum of P_0 , P_1 , P_2), this parameter should be defined. You can define P_t in the ESS file using Stubs for P_0 , P_1 , and P_2 . To compare the graph you made with Figure.3 in the paper, write 'Pn', 'P0', 'P1', 'P2' parameters in the script to get 'MolarConc' instead of 'Value' from defined EntityStubs.

- New Stubs need to be defined: Stubs for Variable P0, P1, P2

```

loadModel( 'Drosophila.eml' )

Pn_Stub = createEntityStub( 'Variable://Pn' )
P0_Stub = createEntityStub( 'Variable://P0' )
P1_Stub = createEntityStub( 'Variable://P1' )
P2_Stub = createEntityStub( 'Variable://P2' )
M_Stub = createEntityStub( 'Variable://M' )
R_toy10_Stub = createEntityStub( 'Process://R_toy10' )

Pn_Stub.setProperty( 'Value', 6.21367e-1 )
R_toy10_Stub.setProperty( 'vd', 0.95 )

while getCurrentTime() < 72.0:
    Pn = Pn_Stub.getProperty( 'Value' )
    M = M_Stub.getProperty( 'Value' )
    R_toy10 = R_toy10_Stub.getProperty( 'Activity' )
    print Pn, '%t', M, '%t', R_toy10
    step()

stop()

```

- Stubs that can be deleted: the Stub for R_toy10

```

loadModel( 'Drosophila.eml' )

Pn_Stub = createEntityStub( 'Variable://Pn' )
P0_Stub = createEntityStub( 'Variable://P0' )
P1_Stub = createEntityStub( 'Variable://P1' )
P2_Stub = createEntityStub( 'Variable://P2' )
M_Stub = createEntityStub( 'Variable://M' )
R_toy10_Stub = createEntityStub( 'Process://R_toy10' )

Pn_Stub.setProperty( 'Value', 6.21367e-1 )
R_toy10_Stub.setProperty( 'vd', 0.95 )

while getCurrentTime() < 72.0:
    Pn = Pn_Stub.getProperty( 'Value' )
    M = M_Stub.getProperty( 'Value' )
R_toy10 = R_toy10_Stub.getProperty( 'Activity' )
    print Pn, '%t', M, '%t', R_toy10
    step()

stop()

```

- The attributes need to be got from EntityStubs: MolarConc

```

loadModel( 'Drosophila.eml' )

Pn_Stub = createEntityStub( 'Variable://Pn' )
P0_Stub = createEntityStub( 'Variable://P0' )
P1_Stub = createEntityStub( 'Variable://P1' )
P2_Stub = createEntityStub( 'Variable://P2' )
M_Stub = createEntityStub( 'Variable://M' )

Pn_Stub.setProperty( 'Value', 6.21367e-1 )

while getTime() < 72.0:
    Pn = Pn_Stub.getProperty( 'MolarConc' )
    P0 = P0_Stub.getProperty( 'MolarConc' )
    P1 = P1_Stub.getProperty( 'MolarConc' )
    P2 = P2_Stub.getProperty( 'MolarConc' )
    M = M_Stub.getProperty( 'MolarConc' )
    print Pn, '%t', M
    step()

stop()

```

- A new parameter to be wrote in the script: $P_t(=P_0+P_1+P_2+P_n)$

```

loadModel( 'Drosophila.eml' )

Pn_Stub = createEntityStub( 'Variable://Pn' )
P0_Stub = createEntityStub( 'Variable://P0' )
P1_Stub = createEntityStub( 'Variable://P1' )
P2_Stub = createEntityStub( 'Variable://P2' )
M_Stub = createEntityStub( 'Variable://M' )

Pn_Stub.setProperty( 'Value', 6.21367e-1 )

while getTime() < 72.0:
    Pn = Pn_Stub.getProperty( 'MolarConc' )
    P0 = P0_Stub.getProperty( 'MolarConc' )
    P1 = P1_Stub.getProperty( 'MolarConc' )
    P2 = P2_Stub.getProperty( 'MolarConc' )

    Pt = P0 + P1 + P2 + Pn

    M = M_Stub.getProperty( 'MolarConc' )
    print Pn, '%t', M
    step()

stop()

```

- Data output format: Pt [tab] M

```
loadModel( 'Drosophila.eml' )

Pn_Stub = createEntityStub( 'Variable:/:Pn' )
P0_Stub = createEntityStub( 'Variable:/:P0' )
P1_Stub = createEntityStub( 'Variable:/:P1' )
P2_Stub = createEntityStub( 'Variable:/:P2' )
M_Stub = createEntityStub( 'Variable:/:M' )

Pn_Stub.setProperty( 'Value', 6.21367e-1 )

while getCurrentTime() < 72.0:
    Pn = Pn_Stub.getProperty( 'MolarConc' )
    P0 = P0_Stub.getProperty( 'MolarConc' )
    P1 = P1_Stub.getProperty( 'MolarConc' )
    P2 = P2_Stub.getProperty( 'MolarConc' )

    Pt = P0 + P1 + P2 + Pn

    M = M_Stub.getProperty( 'MolarConc' )
    print Pt, '%t', M
    step()

stop()
```

- Set initial values of the Variables as mentioned in Goldbeter1995, Figure3 legend.

```
loadModel( 'Drosophila.eml' )

Pn_Stub = createEntityStub( 'Variable:/:Pn' )
P0_Stub = createEntityStub( 'Variable:/:P0' )
P1_Stub = createEntityStub( 'Variable:/:P1' )
P2_Stub = createEntityStub( 'Variable:/:P2' )
M_Stub = createEntityStub( 'Variable:/:M' )

step()

Pn_Stub.setProperty( 'MolarConc', 0.25 )
P0_Stub.setProperty( 'MolarConc', 0.25 )
P1_Stub.setProperty( 'MolarConc', 0.25 )
P2_Stub.setProperty( 'MolarConc', 0.25 )
M_Stub.setProperty( 'MolarConc', 0.1 )

while getCurrentTime() < 72.0:
    Pn = Pn_Stub.getProperty( 'MolarConc' )
    P0 = P0_Stub.getProperty( 'MolarConc' )
    P1 = P1_Stub.getProperty( 'MolarConc' )
    P2 = P2_Stub.getProperty( 'MolarConc' )

    Pt = P0 + P1 + P2 + Pn

    M = M_Stub.getProperty( 'MolarConc' )
    print Pt, '%t', M
    step()

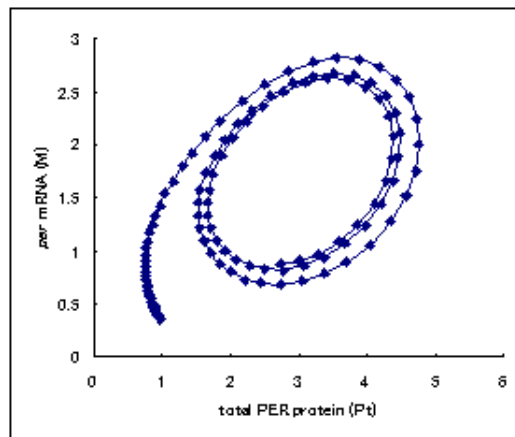
stop()
```

- Run simulation with 'fig3.py', save printed data to 'fig3data.txt', make phase-space plot that is correspondent with Goldbeter1995, Figure3.(see below)

```

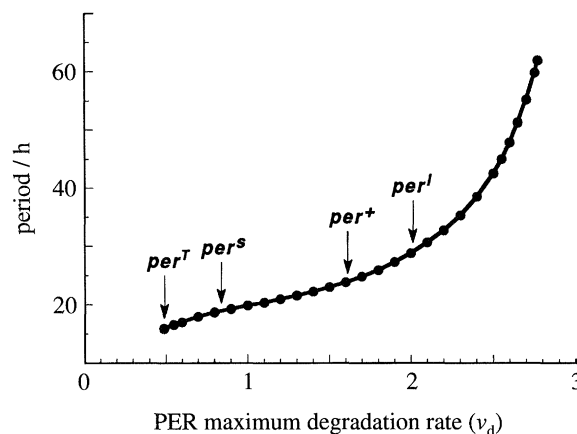
E-Cell Console
C:\Program Files\E-Cell 3\tutorial>ecell3-session fig3.py > fig3data.txt
C:\Program Files\E-Cell 3\tutorial>

```



9 Hands-on 3: Conducting similar experiment with Goldbeter1995, Figure 4

Figure.4 in Goldbeter1995 shows dependence of the period of PER oscillations on the maximum rate of PER degradation(v_d).



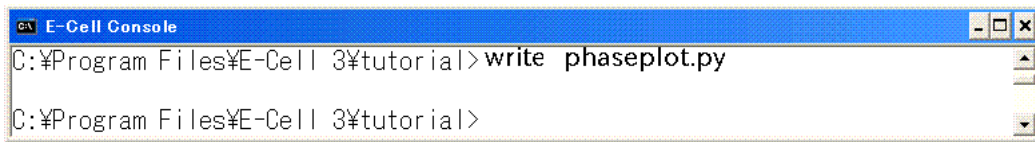
(Goldbeter, A. *Proc. R. Soc. Lond. B. Biol. Sci.* **261**:319-324, 1995) Figure.3

To obtain these data, you need to run simulations changing different value for parameter 'vd' in R_toy10 Process.

There are several ways to do that. Here we will do it by parameterizing the 'vd' value in the session script. One example is shown below.

Defining model parameter on command-line

1. Open 'phaseplot.py' with a text editor.



```
E-Cell Console
C:\Program Files\E-Cell 3\tutorial>write phaseplot.py
C:\Program Files\E-Cell 3\tutorial>
```

2. Make a slight change on a line in 'phaseplot.py' like below:

before:

```
R_toy10_Stub.setProperty( 'vd', 0.95 )
```

modified:

```
R_toy10_Stub.setProperty( 'vd', vd_value )
```

For example, when you need to run simulation with v_d value = 2.0, you can do it by typing:
(You don't actually need to do it now. It will be needed later.)

```
% ecell3-session -Dvd_value=2.0 phaseplot.py
```

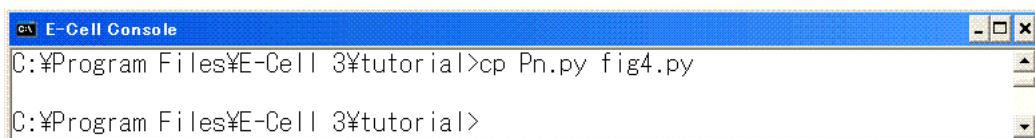
General usage is:

```
% ecell3-session -D[Param1]=[Value1] -D[Param2]=[Value2] · [Session  
script file]
```

Writing a script for the re-experiment

Now, let's move on to do re-experiment of Goldbeter1995, Figure4. What you are going to do here is to run simulations changing v_d value from 0.5 to 2.8 (every 0.05), save simulated time-course of P_n value, count period and plot period versus v_d .

1. Copy 'Pn.py' to 'fig4.py', open it with a text editor.



```
E-Cell Console
C:\Program Files\E-Cell 3\tutorial>cp Pn.py fig4.py
C:\Program Files\E-Cell 3\tutorial>
```

2. Add 'fig4.py' the line which defines EntityStub for 'Process://R_toy10' as 'R_toy10_Stub'.

```
loadModel( 'Drosophila.eml' )
Pn_logger = createLoggerStub( 'Variable://Pn:Value' )
Pn_logger.create()

R_toy10_Stub = createEntityStub( 'Process://R_toy10' )

run( 72 )
stop()
saveLoggerData()
```

3. Add a line to set parameter 'vd' of 'R_toy10_Stub' to 'vd_value', to set 'vd_value' afterwards from command-line.

```
loadModel( 'Drosophila.eml' )
Pn_logger = createLoggerStub( 'Variable://Pn:Value' )
Pn_logger.create()

R_toy10_Stub = createEntityStub( 'Process://R_toy10' )
R_toy10_Stub.setProperty( 'vd', vd_value )

run( 72 )
stop()
saveLoggerData()
```

4. Add 'fig4.py' the line which defines EntityStub for 'Process://R_toy3' as 'R_toy3_Stub'.

```
loadModel( 'Drosophila.eml' )
Pn_logger = createLoggerStub( 'Variable://Pn:Value' )
Pn_logger.create()

R_toy10_Stub = createEntityStub( 'Process://R_toy10' )
R_toy10_Stub.setProperty( 'vd', vd_value )

R_toy3_Stub = createEntityStub( 'Process://R_toy3' )

run(72)
stop()
saveLoggerData()
```

5. Add a line to set parameter 'Ks' of 'R_toy3_Stub' to '0.78', as mentioned in Goldbeter1995, Figure4 legend.

```
loadModel( 'Drosophila.eml' )
Pn_logger = createLoggerStub( 'Variable://Pn:Value' )
Pn_logger.create()

R_toy10_Stub = createEntityStub( 'Process://R_toy10' )
R_toy10_Stub.setProperty( 'vd', vd_value )

R_toy3_Stub = createEntityStub( 'Process://R_toy3' )
R_toy3_Stub.setProperty( 'Ks', 0.78 )

run(72)
stop()
saveLoggerData()
```

6. Modify the script to run simulation longer, 480seconds(Note that actually the unit of time in the reference is hour.).

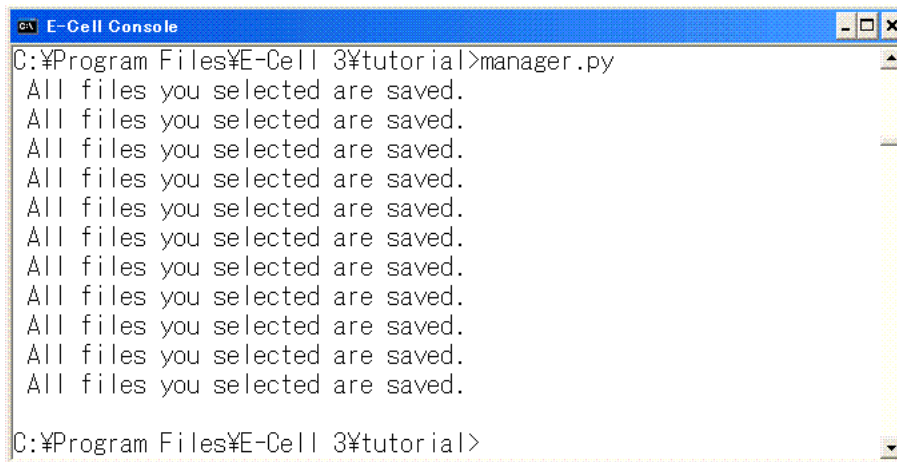
```
loadModel( 'Drosophila.eml' )
Pn_logger = createLoggerStub( 'Variable://Pn:Value' )
Pn_logger.create()

R_toy10_Stub = createEntityStub( 'Process://R_toy10' )
R_toy10_Stub.setProperty( 'vd', vd_value )

R_toy3_Stub = createEntityStub( 'Process://R_toy3' )
R_toy3_Stub.setProperty( 'Ks', 0.78 )

run(480)
stop()
saveLoggerData()
```

7. On E-CELL console, type:



```
C:\Program Files\E-Cell 3\tutorial>manager.py
All files you selected are saved.
All files you selected are saved.
All files you selected are saved.
All files you selected are saved.
All files you selected are saved.
All files you selected are saved.
All files you selected are saved.
All files you selected are saved.
All files you selected are saved.
All files you selected are saved.
All files you selected are saved.
All files you selected are saved.
C:\Program Files\E-Cell 3\tutorial>
```

This will invoke simulation sessions changing value of v_d from 0.5 to 2.8.

'manager.py' is a simple script that calls ecell3-session giving different parameters.

manager.py

```
import os
x = 0.5
while x <= 2.8 :
    print "vd = " + str( x )
    os.system("ecell3-session -Dvd_value=%f fig4.py" % x)
    os.system("mv Data/Variable_Pn_Value.ecd Data/fig4_%.2f.ecd" % x)
    x += 0.05
```

Note
You can do the same experiment in more sophisticated way with ecell-session-manager which manages multiple ecell-sessions. It is available in the linux version. Please check E-CELL tutorial book Chapter 5 or user's manual Chapter 5 for detail.

8. Make sure that you have ecd files in the 'Data' directory.

```

C:\Program Files\E-Cell 3\tutorial>ls Data
Variable_S_Value.ecd  fig4_1.05.ecd  fig4_1.70.ecd  fig4_2.35.ecd
Variable_S_Value.xls  fig4_1.10.ecd  fig4_1.75.ecd  fig4_2.40.ecd
fig4_0.50.ecd         fig4_1.15.ecd  fig4_1.80.ecd  fig4_2.45.ecd
fig4_0.55.ecd         fig4_1.20.ecd  fig4_1.85.ecd  fig4_2.50.ecd
fig4_0.60.ecd         fig4_1.25.ecd  fig4_1.90.ecd  fig4_2.55.ecd
fig4_0.65.ecd         fig4_1.30.ecd  fig4_1.95.ecd  fig4_2.60.ecd
fig4_0.70.ecd         fig4_1.35.ecd  fig4_2.00.ecd  fig4_2.65.ecd
fig4_0.75.ecd         fig4_1.40.ecd  fig4_2.05.ecd  fig4_2.70.ecd
fig4_0.80.ecd         fig4_1.45.ecd  fig4_2.10.ecd  fig4_2.75.ecd
fig4_0.85.ecd         fig4_1.50.ecd  fig4_2.15.ecd  fig4_2.80.ecd
fig4_0.90.ecd         fig4_1.55.ecd  fig4_2.20.ecd
fig4_0.95.ecd         fig4_1.60.ecd  fig4_2.25.ecd
fig4_1.00.ecd         fig4_1.65.ecd  fig4_2.30.ecd
  
```

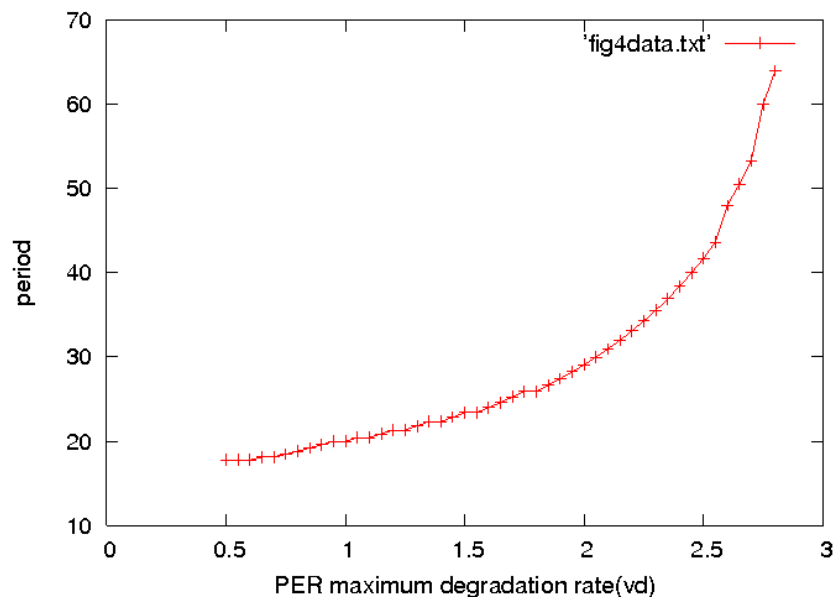
9. Calculate period of simulated oscillations for all simulation result.

```

C:\Program Files\E-Cell 3\tutorial>for %i in (Data\fig4_*.ecd); do countPeriods.py %i >> fig4data.txt
  
```

'countPeriods.py' is a roughly made script that count oscillation and print approximate period for each simulated time-course. It will produce lines like '1.00 [tab] 20.0' for the given .ecd file that means when simulated the model with v_d value == 1.00, the oscillation period was approx. 20.0.

10. Plot 'fig4data.txt'.

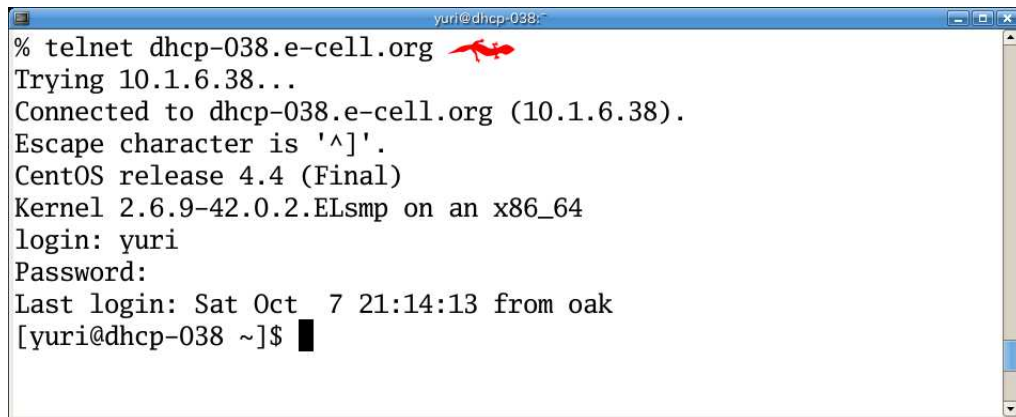


10 Hands-on 4: Using E-CELL analysis toolkit

In section 9, we used E-CELL session scripts written on your own. On the other hand, E-CELL have some ready-made analysis scripts. In this section, you will see the simple example to make a bifurcation diagram with a ready-made script.

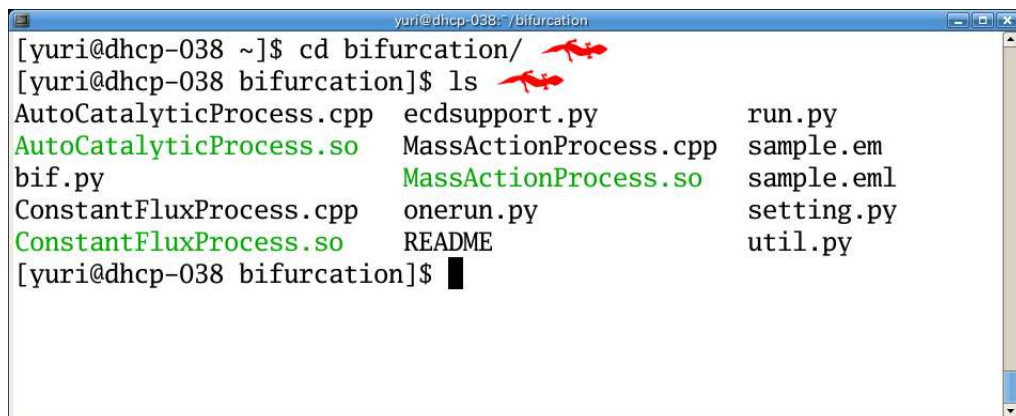
Since it uses ecell-session-manager which doesn't work on Windows version, we need to log in to linux server and do the sample analysis.

1. On E-CELL console, type: (server name, user name and password will be provided on site.)



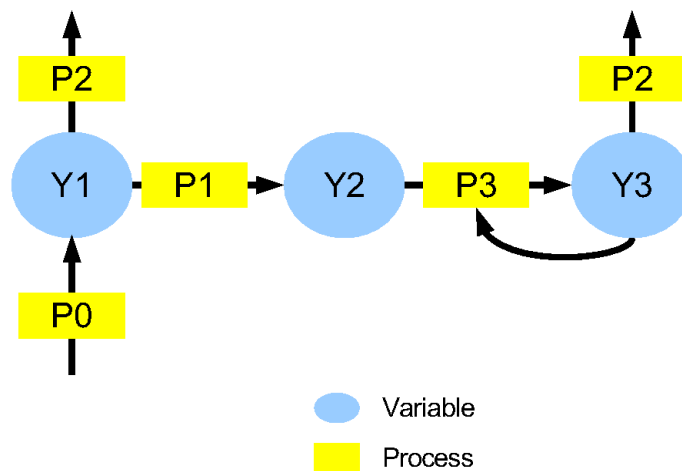
```
yuri@dhcp-038:~  
% telnet dhcp-038.e-cell.org  
Trying 10.1.6.38...  
Connected to dhcp-038.e-cell.org (10.1.6.38).  
Escape character is '^]'.  
CentOS release 4.4 (Final)  
Kernel 2.6.9-42.0.2.ELsmp on an x86_64  
login: yuri  
Password:  
Last login: Sat Oct 7 21:14:13 from oak  
[yuri@dhcp-038 ~]$
```

2. Change directory and see list of files provided. Type:



```
yuri@dhcp-038:~/bifurcation  
[yuri@dhcp-038 ~]$ cd bifurcation/  
[yuri@dhcp-038 bifurcation]$ ls  
AutoCatalyticProcess.cpp  ecdsupport.py          run.py  
AutoCatalyticProcess.so  MassActionProcess.cpp  sample.em  
bif.py                   MassActionProcess.so  sample.eml  
ConstantFluxProcess.cpp  onerun.py             setting.py  
ConstantFluxProcess.so  README                util.py  
[yuri@dhcp-038 bifurcation]$
```

You can assign which model to use and what kind of parameters of the model to change in settings.py. In this case, sample.eml is assigned as a model and parameter k of 'P2' Process is changed from 0.1 to 3.0.



3. Run simulations.

```

yuri@dhcp-038: ~/bifurcation
[yuri@dhcp-038 bifurcation]$ ecell3-session-manager run.py
Bifurcation analysis is started

setting : Data directory has been made
setting : set the SessionManager environment as Local

cpu(1) finished(0) error(0) runnning(1)
(0/29)   cpu(1) finished(1) error(0) runnning(1)
(1/29) .  cpu(1) finished(2) error(0) runnning(1)
(2/29) .. cpu(1) finished(3) error(0) runnning(1)
(3/29) ... cpu(1) finished(4) error(0) runnning(1)
(4/29) .... cpu(1) finished(5) error(0) runnning(1)
(5/29) .....cpu(1) finished(6) error(0) runnning(1)
(6/29)      cpu(1) finished(7) error(0) runnning(1)
(7/29) .    cpu(1) finished(8) error(0) runnning(1)
(8/29) ..   cpu(1) finished(9) error(0) runnning(1)
  
```

4. Make bifurcation diagram.

```

yuri@dhcp-038: ~/bifurcation
[yuri@dhcp-038 bifurcation]$ ecell3-python bif.py
[yuri@dhcp-038 bifurcation]$
  
```

5. Make sure that the bifurcation diagram data are successfully saved in the Data directory.

```

yuri@dhcp-038: ~/bifurcation
[yuri@dhcp-038 bifurcation]$ ls Data/*diagram.dat
Data/Variable__Y1_Value_diagram.dat
Data/Variable__Y2_Value_diagram.dat
Data/Variable__Y3_Value_diagram.dat
[yuri@dhcp-038 bifurcation]$
  
```

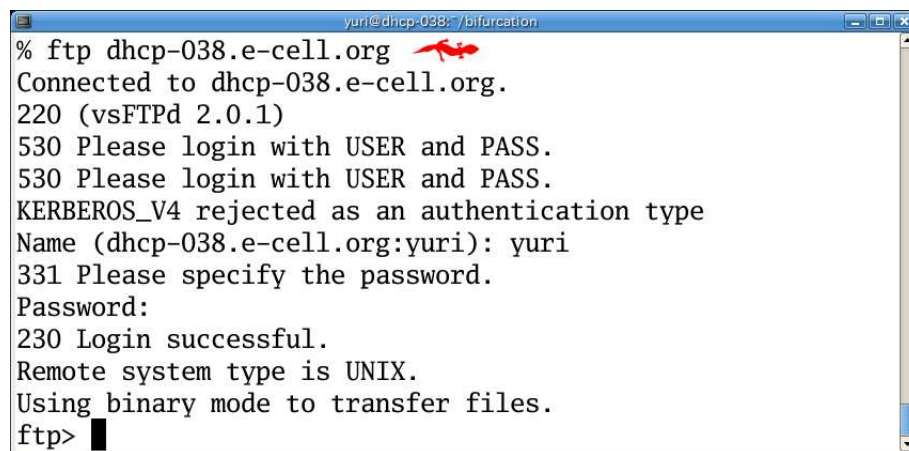
6. Logout from the server.



```
yuri@dhcp-038: /bifurcation
[yuri@dhcp-038 bifurcation]$ exit
```

7. Get the data of bifurcation diagram.

- Connect the server via ftp (server name will be given on site).



```
yuri@dhcp-038: /bifurcation
% ftp dhcp-038.e-cell.org
Connected to dhcp-038.e-cell.org.
220 (vsFTPd 2.0.1)
530 Please login with USER and PASS.
530 Please login with USER and PASS.
KERBEROS_V4 rejected as an authentication type
Name (dhcp-038.e-cell.org:yuri): yuri
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

- Change directory.



```
yuri@dhcp-038: /bifurcation
ftp> cd bifurcation/Data
250 Directory successfully changed.
ftp>
```

- Get diagram files.

```

yuri@dhcp-038: ~/bifurcation
ftp> mget *diagram.dat
mget Variable___Y1_Value_diagram.dat? y
227 Entering Passive Mode (10,1,6,38,93,195)
150 Opening BINARY mode data connection for Variable___Y1_Value_diagram.
dat (18458 bytes).
226 File send OK.
18458 bytes received in 0.0019 seconds (9.6e+03 Kbytes/s)
mget Variable___Y2_Value_diagram.dat? y
227 Entering Passive Mode (10,1,6,38,253,231)
150 Opening BINARY mode data connection for Variable___Y2_Value_diagram.
dat (21098 bytes).
226 File send OK.
21098 bytes received in 0.0019 seconds (1.1e+04 Kbytes/s)
mget Variable___Y3_Value_diagram.dat? y
227 Entering Passive Mode (10,1,6,38,58,239)
150 Opening BINARY mode data connection for Variable___Y3_Value_diagram.
dat (23056 bytes).
226 File send OK.
23056 bytes received in 0.002 seconds (1.1e+04 Kbytes/s)
ftp> █

```

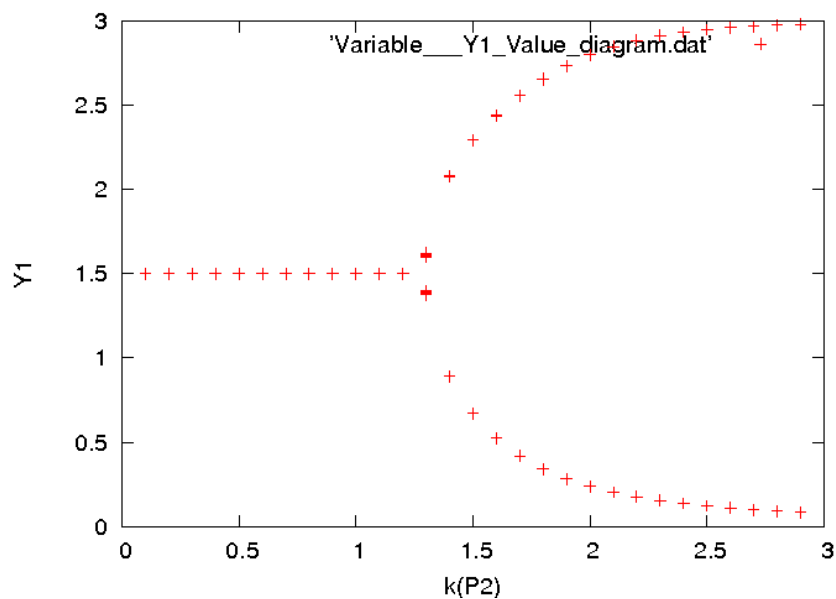
- Close the connection.

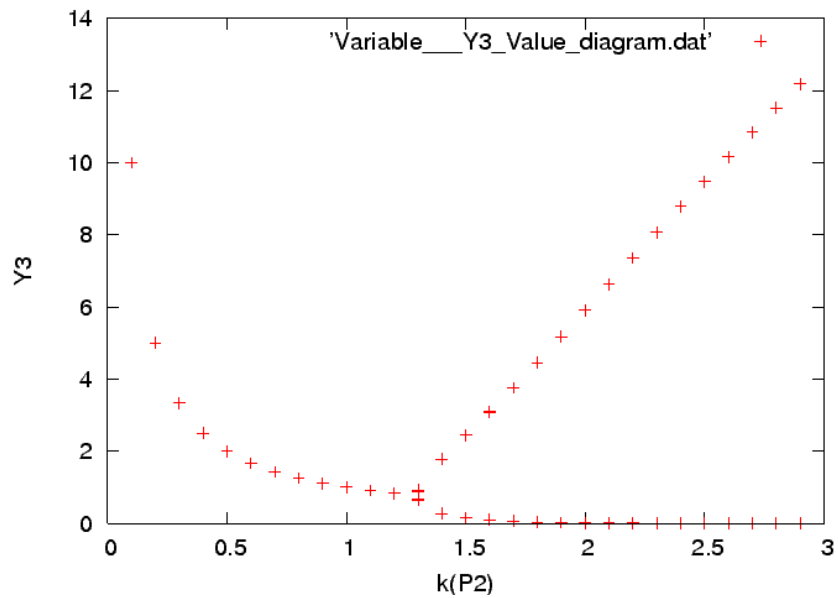
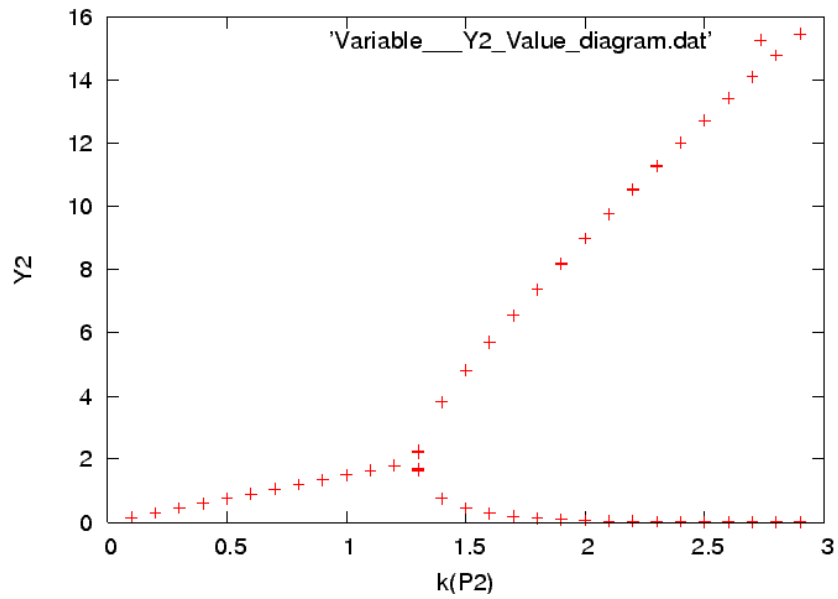
```

yuri@dhcp-038: ~/bifurcation
ftp> bye
221 Goodbye.
% █

```

8. Graph using obtained data files('Variable___Y1_Value_diagram.dat', 'Variable___Y2_Value_diagram.dat', 'Variable___Y3_Value_diagram.dat').





Note

E-CELL analysis toolkit only have MCA module and bifurcation module thus far. Though, E-CELL 3 is implemented to facilitate user-defined analysis scripts as we did in section 8, 9.

E-CELL is an open-source project. Anybody is welcomed to contribute the development. One of the many ways to contribute is to send us your models (EML or SBML) and analysis scripts. Please contact to E-CELL project's webadmin (see <http://www.e-cell.org>) about the detail. Enjoy!

This handout was originally written by Katsuyuki Yugi,
 edited for ICSB2006 tutorial by Yuri Matsuzaki.