

定量生物に効く数値計算

2012年1月7日

慶應義塾大学 舟橋 啓



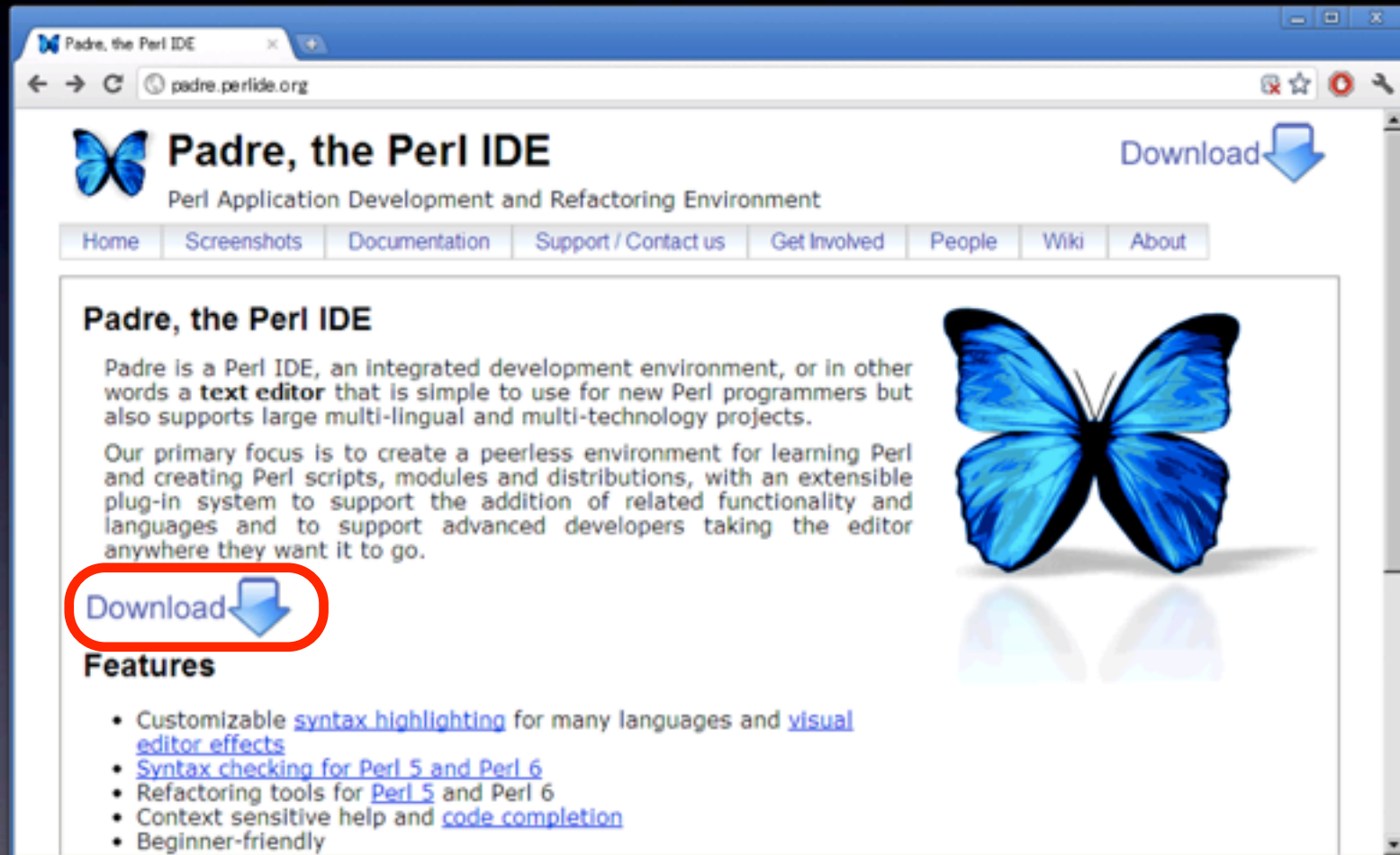
Keio University

1858

CALAMVS
GLADIO
FORTIOR

Windowsの人

<http://padre.perlide.org>



Padre, the Perl IDE
Perl Application Development and Refactoring Environment

Download

Home Screenshots Documentation Support / Contact us Get Involved People Wiki About

Padre, the Perl IDE

Padre is a Perl IDE, an integrated development environment, or in other words a **text editor** that is simple to use for new Perl programmers but also supports large multi-lingual and multi-technology projects.

Our primary focus is to create a peerless environment for learning Perl and creating Perl scripts, modules and distributions, with an extensible plug-in system to support the addition of related functionality and languages and to support advanced developers taking the editor anywhere they want it to go.

Download

Features

- Customizable [syntax highlighting](#) for many languages and [visual editor effects](#)
- [Syntax checking for Perl 5 and Perl 6](#)
- Refactoring tools for [Perl 5](#) and Perl 6
- Context sensitive help and [code completion](#)
- Beginner-friendly


- Beginner-friendly
- Context sensitive help and [code completion](#)
- Refactoring tools for [Perl 5](#) and Perl 6
- [Syntax checking for Perl 5 and Perl 6](#)

Padreをダウンロード

<http://padre.perlide.org>

code.google.com/p/padre-perl-ide/downloads/detail?name=padre-on-strawberry-5.12.3.0-v5.exe

My favorites | Sign in

 padre-perl-ide
Perl IDE written in Perl

Project Home Downloads

Search Current downloads for Search


Download: Padre on Strawberry 5.12.3.0 (v5)
5 people starred this download

Uploaded by: szab_@email.com

Released:
Uploaded: Jun 20, 2011
Downloads: 16156
Featured
Type-Installer
OpSys-Windows

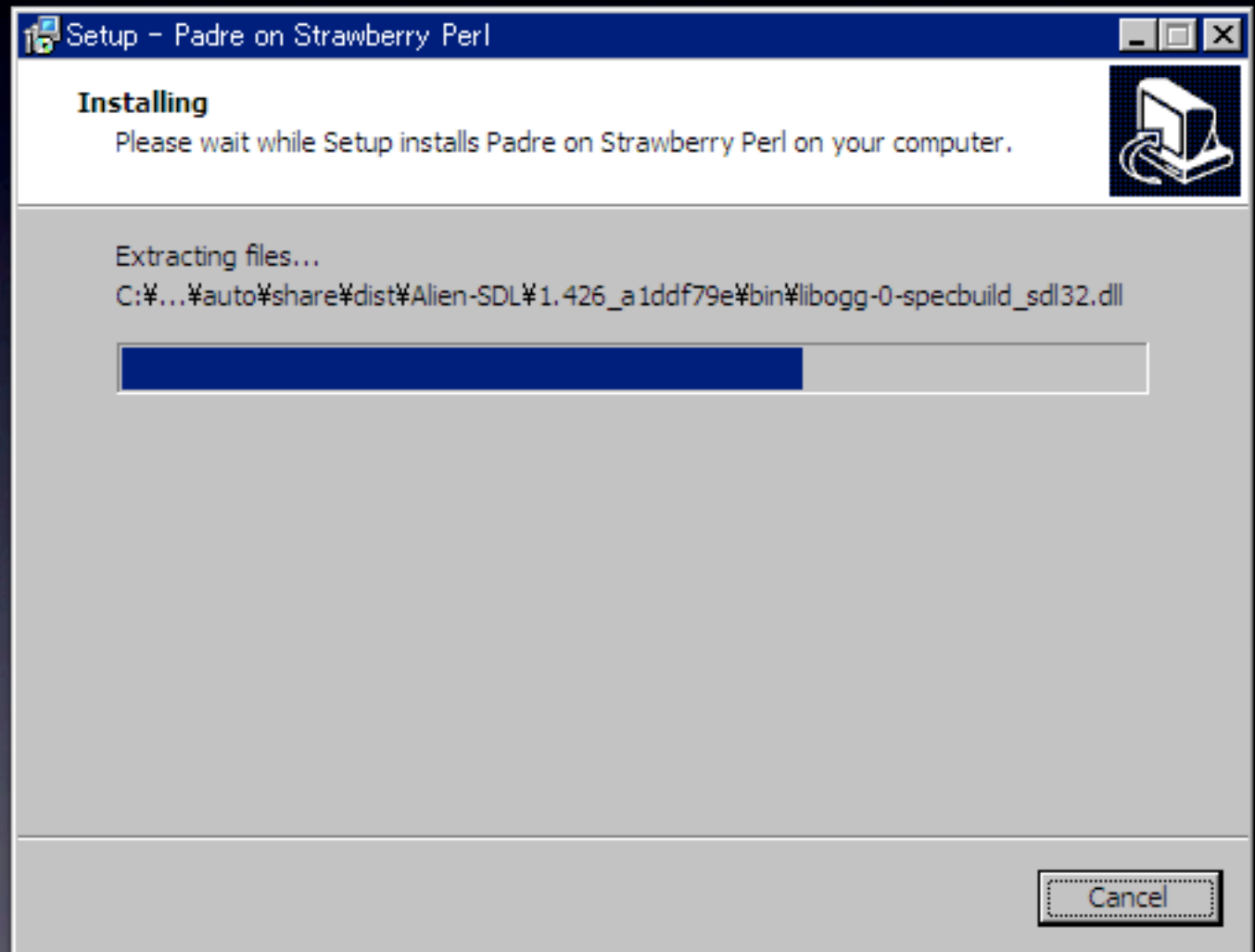
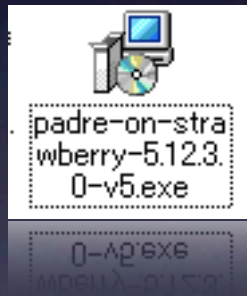
File: [padre-on-strawberry-5.12.3.0-v5.exe](#) 71.9 MB

Description:
SHA1 Checksum: f81c15a56a3efc253458b1fc39beef9c6d939274 [What's this?](#)



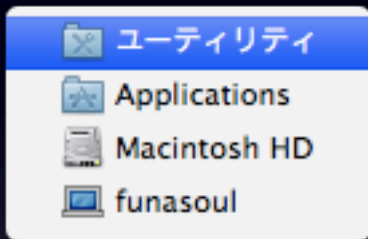
Padreをinstall

<http://padre.perlide.org>

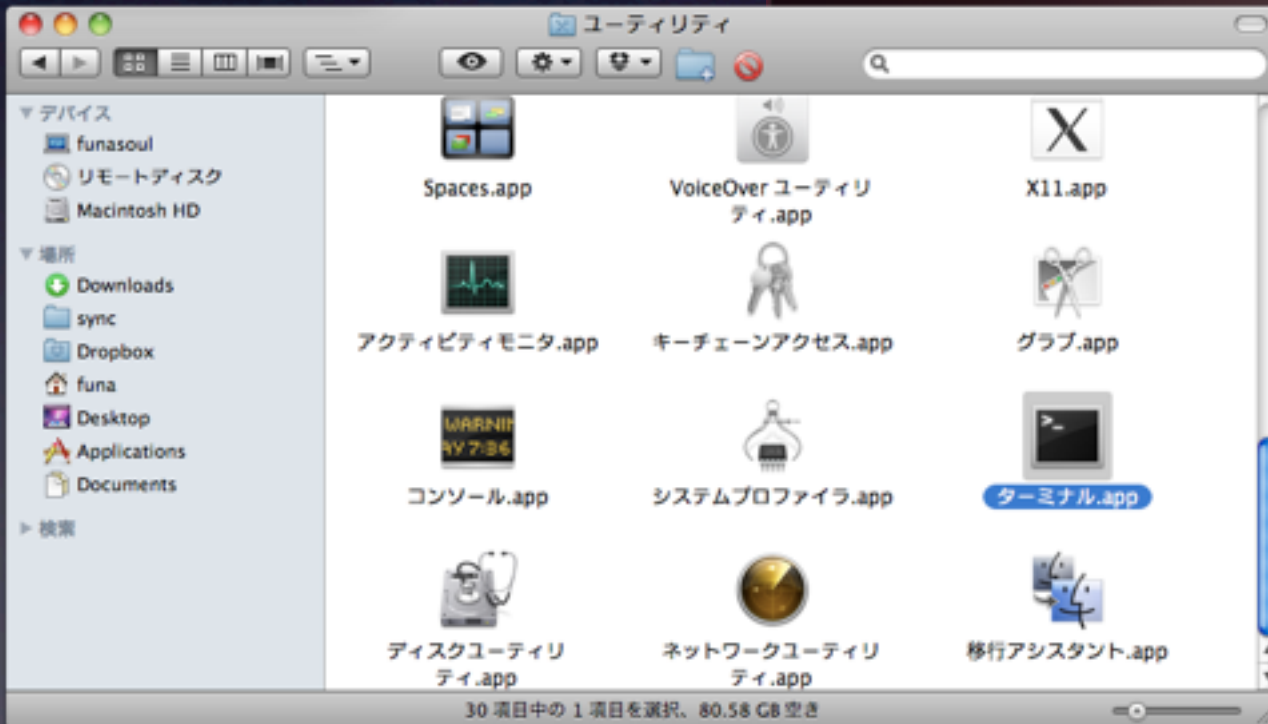


Mac, Linuxの人

Terminalを起動

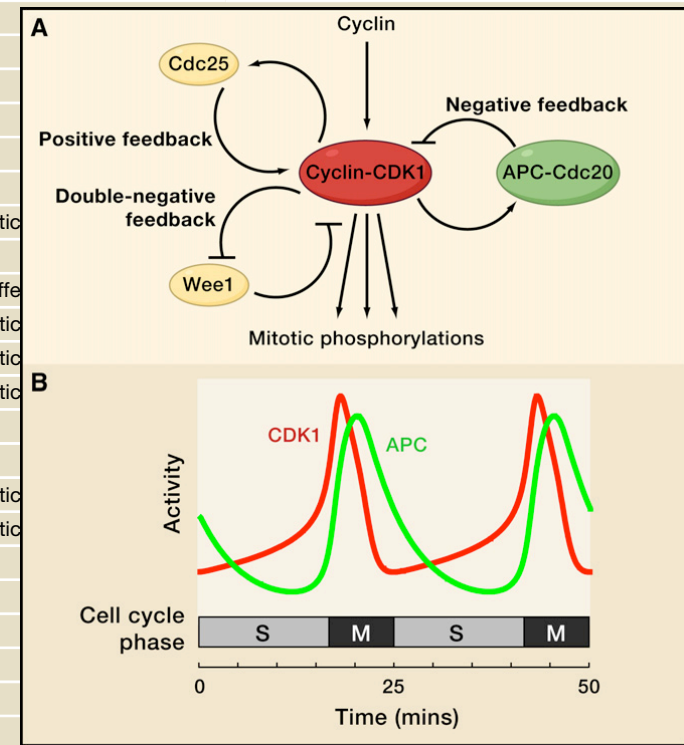


```
funa@ubuntu: ~  
File Edit View Search Terminal Help  
ubuntu% ls  
CellDesigner4.2/          Documents/          runCellDesigner4.2@  
CellDesigner-4.2-linux-installer.bin*  Downloads/        Templates/  
CellDesigner42linux.tar.gz  examples.desktop  Uninstall_CellDesigner4.2@  
CellDesigner4.2-tar/      Music/             Videos/  
CellDesignerSim/         Pictures/          workspace/  
Desktop/                 Public/  
ubuntu% |
```



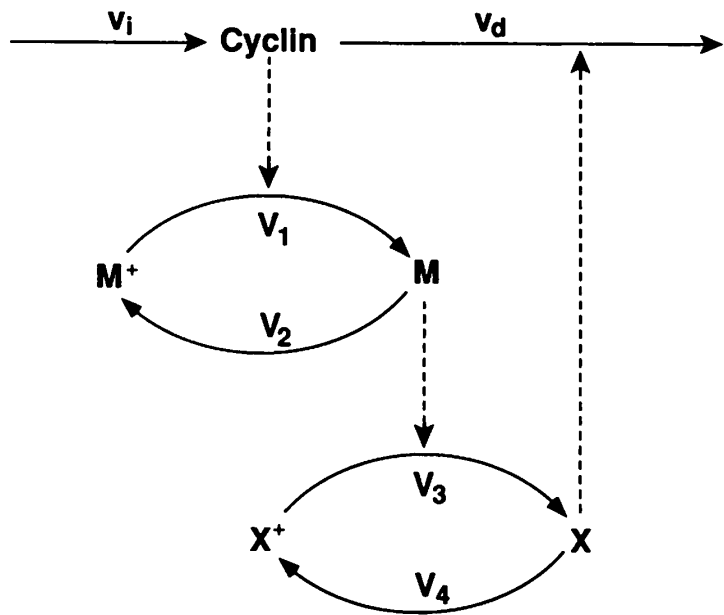
数理モデルを扱った論文

1970	No specific organism	ODE	(Sel'kov, 1970)
1974	No specific organism	ODE	(Gilbert, 1974)
1975	<i>Physarum polycephalum</i>	ODE	(Kauffman and Wille, 1975)
1975	<i>Physarum polycephalum</i>	ODE	(Tyson and Kauffman, 1975)
1991	<i>Xenopus laevis</i> embryos	ODE	(Goldbeter, 1991)
1991	<i>Xenopus</i> embryos	ODE	(Norel and Agur, 1991)
1991	<i>Xenopus</i> embryos, somatic cells	ODE	(Tyson, 1991)
1992	<i>Xenopus</i> embryos	ODE	(Obeyesekere et al., 1992)
1993	<i>Xenopus</i> embryos	ODE	(Novak and Tyson, 1993a)
1993	<i>Xenopus</i> embryos	ODE	(Novak and Tyson, 1993b)
1994	<i>Xenopus</i> embryos	ODE, delay differential equations	(Busenberg and Tang, 1994)
1996	<i>Xenopus</i> embryos	ODE	(Goldbeter and Guilmot, 1996)
1997	<i>S. pombe</i>	ODE	(Novak and Tyson, 1997)
1998	<i>S. pombe</i>	ODE	(Novak et al., 1998)
1998	<i>Xenopus</i> embryos	ODE	
1999	Mammalian somatic cells	ODE	
2003	<i>Xenopus</i> embryos	ODE	
2003	<i>S. cerevisiae</i>	ODE	
2004	<i>S. cerevisiae</i>	ODE	
2004	<i>S. pombe</i>	Boolean	
2004	<i>S. pombe</i>	Stochastic	
2005	<i>Xenopus</i> embryos	ODE	
2006	Mammalian somatic cells	Delay diffe	
2006	<i>S. cerevisiae</i>	Stochastic	
2007	<i>S. cerevisiae</i>	Stochastic	
2007	<i>S. cerevisiae</i>	Stochastic	
2007	<i>S. cerevisiae</i>	Hybrid	
2008	<i>Xenopus</i> embryos	ODE	
2008	<i>S. cerevisiae</i>	Stochastic	
2008	<i>S. cerevisiae</i>	Stochastic	
2008	<i>S. pombe</i>	Boolean	
2008	Mammalian somatic cells	ODE	
2009	Mammalian somatic cells	ODE	
2010	<i>S. cerevisiae</i>	ODE	
2010	<i>S. cerevisiae</i> , <i>S. pombe</i>	Boolean	
2010	<i>S. pombe</i>	ODE	



Cell, 2011, 144(6): 874-85

数理モデルを扱った論文



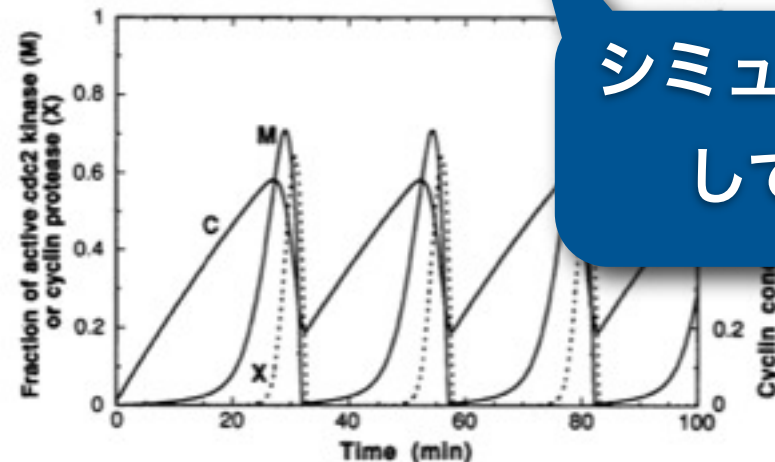
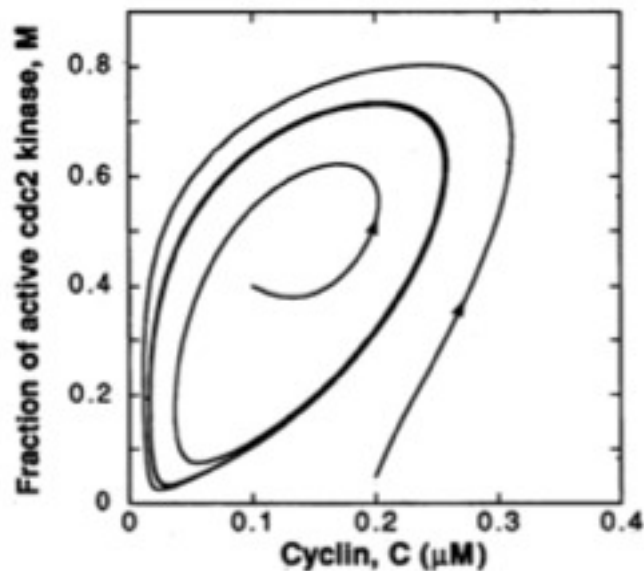
$$\frac{dC}{dt} = v_i - v_d X \frac{C}{K_d + C} - k_d C$$

$$\frac{dM}{dt} = V_1 \frac{(1-M)}{K_1 + (1-M)} - V_2 \frac{M}{K_2 + M}$$

$$\frac{dX}{dt} = V_3 \frac{(1-X)}{K_3 + (1-X)} - V_4 \frac{X}{K_4 + X} \quad [1]$$

with

$$V_1 = \frac{C}{K_c + C} V_{M1}, \quad V_3 = M V_{M3}. \quad [2]$$

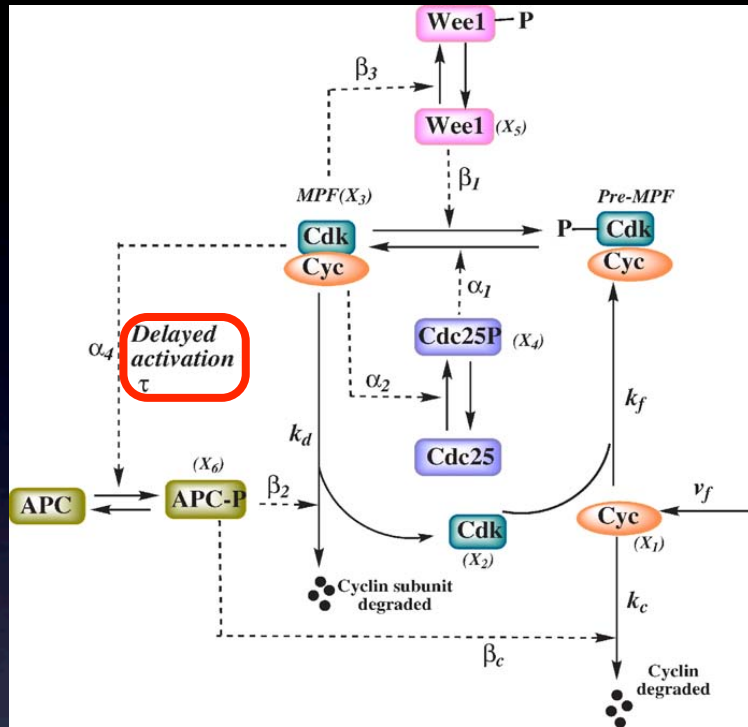


シミュレーション
してみたい!

PNAS, 1991, 88(20): 9107-9111.

数理モデルを扱った論文

シミュレーション
してみたい!



$$\frac{dX_1}{dt} = v_f - k_f X_1 X_2 - (k_c + \beta_c X_6) X_1$$

$$\frac{dX_2}{dt} = k_d [1 + \beta_2 X_6] X_3 - k_f X_1 X_2$$

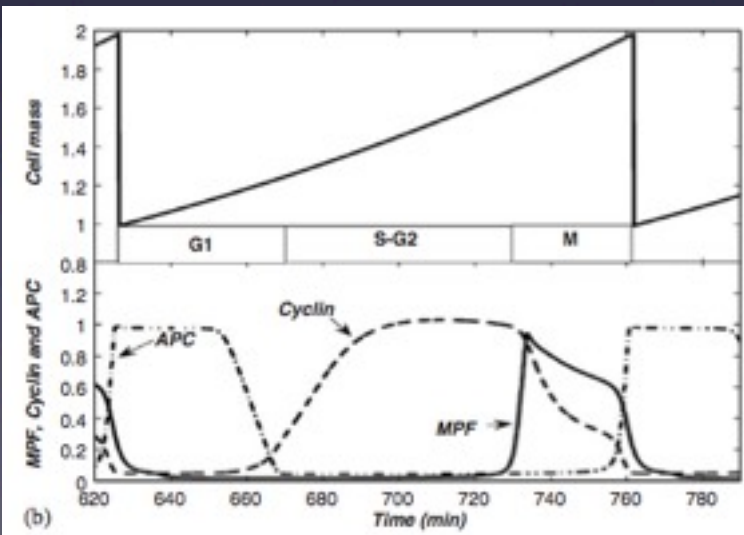
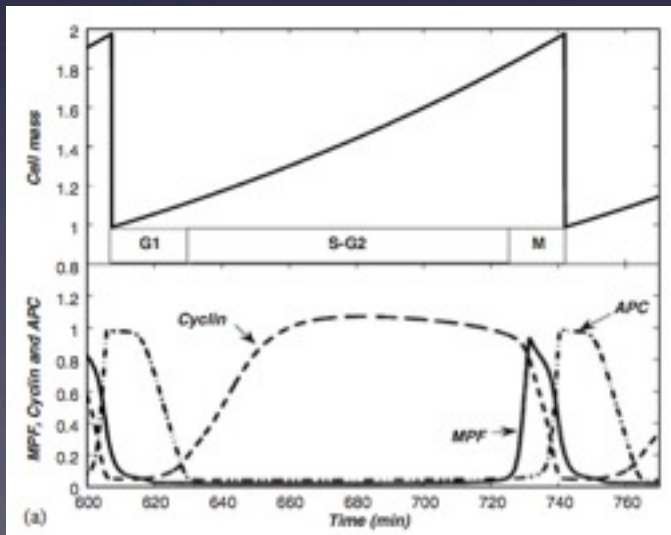
$$\frac{dX_3}{dt} = v_{M1} \left[1 + \alpha_1 \frac{X_4}{K_a + X_4} \right] \frac{(C - X_2 - X_3)}{J_1 + (C - X_2 - X_3)} - v_{M1} \frac{X_3}{K_a + X_3} \frac{1}{J_1 + X_3}$$

$$\frac{dX_4}{dt} = v_{M2} \left[1 + \alpha_2 \frac{m X_3}{K_a + m X_3} \right] \frac{(1 - X_4)}{J_2 + (1 - X_4)} - v_{M2} \frac{X_4}{J_2 + X_4}$$

$$\frac{dX_5}{dt} = v_{M3} \frac{(1 - X_5)}{J_3 + (1 - X_5)} - v_{M3}' \left[1 + \beta_3 \frac{m X_3}{K_a + m X_3} \right] \frac{X_5}{J_3 + X_5}$$

$$\frac{dX_6}{dt} = v_{M4} \left[1 + \alpha_4 \frac{m X_3(t - \tau)}{K_a + m X_3(t - \tau)} \right] \frac{(1 - X_6)}{J_4 + (1 - X_6)} - v_{M4}' \frac{X_6}{J_4 + X_6}$$

$$\frac{dm}{dt} = \mu' m (1 - m/a)$$



J Theor Biol, 2006, 241(3): 617-27

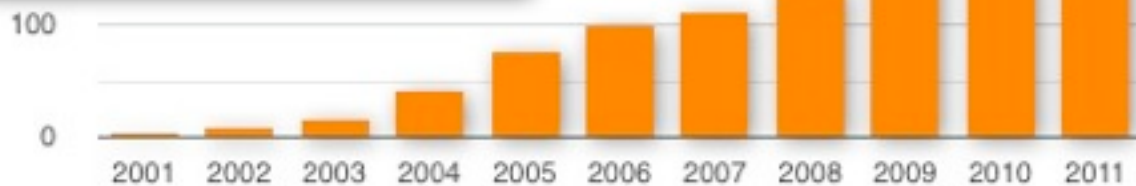
シミュレータの現状

	Recent contact	Capabilities					Frameworks						API	Dep.	Platforms				SBML		Availabil.						
		Creation	Simulation	Analysis	Database	Utility	ODE	DAE	PDE	Stochastic	Events	Logical			Other	Linux	Mac OS X	Windows	Web Browser	Import	Export	Open source	Academic use	Commercial use			
Cain	•	•	•	•	•	•			•				C++		•	•	•		•	•	•	•	•	•	•	•	•
CARMEN	•	•	•	•								•	Perl, CellDesigner		•	•	•		•	•	•	•	•	•	•	•	•
Cell Illustrator		•	•	•				•					Java		•	•	•		•	•	•	•	•	•	•	•	•
CellDesigner	•	•	•	•	•	•		•					Java		•	•	•		•	•	•	•	•	•	•	•	•
Cellerator	•	•	•	•				•					Mathematica		•	•	•		•	•	•	•	•	•	•	•	•
CellMC		•	•	•				•							•	•	•		•	•	•	•	•	•	•	•	•
CellML2SBML															•	•	•		•	•	•	•	•	•	•	•	•
CellNetAnalyzer	•	•	•	•	•	•		•			•	•	MATLAB	MATLAB	•	•	•		•	•	•	•	•	•	•	•	•
Cellware		•	•	•				•							•	•	•		•	•	•	•	•	•	•	•	•
CLEML															•	•	•		•	•	•	•	•	•	•	•	•
CL-SBML	•											•	Common Lisp		•	•	•		•	•	•	•	•	•	•	•	•
COBRA	•		•												•	•	•		•	•	•	•	•	•	•	•	•
CompuCell3D	•		•									•	Python, C++		•	•	•		•	•	•	•	•	•	•	•	•
ConsensusPathDB																		•	•	•	•	•	•	•	•	•	•
COPASI	•	•	•	•	•	•		•		•	•		C++, Perl, R, Java, Octave, SBW, Python, R, REST,		•	•	•		•	•	•	•	•	•	•	•	•

This matrix was generated on 2012-01-03 (total number of tools: 231)

どれを選べばいい?

<http://sbml.org/>



今日の目標

- 数理モデルの構築
- シミュレータの実装
- シミュレーション

シミュレーション
してみたい!

どれを選ばいい?

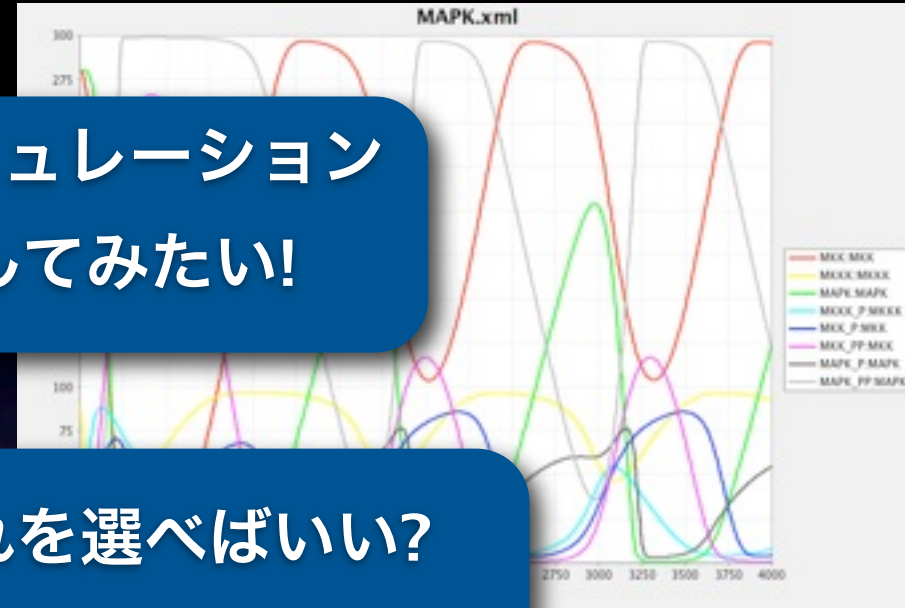


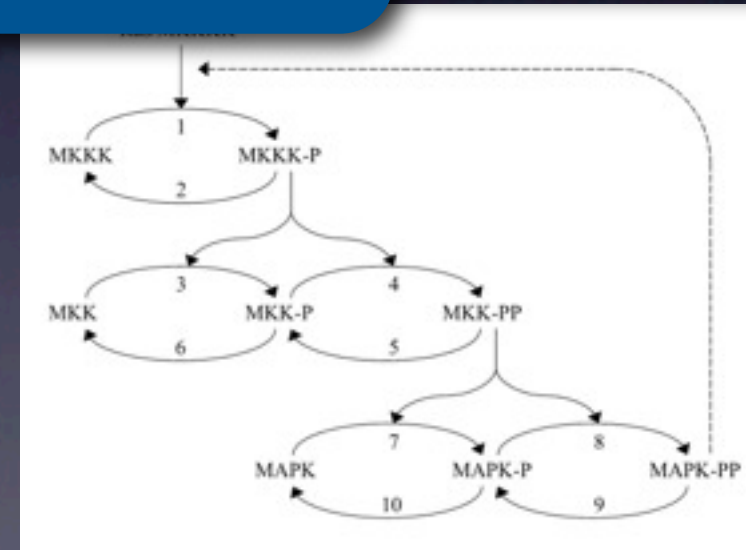
Table 1. Kinetic equations comprising the computation

Reaction number	Rate equation
1	$V_1 \cdot [\text{MKKK}] / (1 + ([\text{MAPK-PP}] / K_7)^n) \cdot (K_1 + [\text{MKKK}])$
2	$V_2 \cdot [\text{MKKK-P}] / (K_2 + [\text{MKKK-P}])$
3	$k_3 \cdot [\text{MKKK-P}] \cdot [\text{MKK}] / (K_3 + [\text{MKK}])$
4	$k_4 \cdot [\text{MKKK-P}] \cdot [\text{MKK-P}] / (K_4 + [\text{MKK-P}])$
5	$V_5 \cdot [\text{MKK-PP}] / (K_5 + [\text{MKK-PP}])$
6	$V_6 \cdot [\text{MKK-P}] / (K_6 + [\text{MKK-P}])$
7	$k_7 \cdot [\text{MKK-PP}] \cdot [\text{MAPK}] / (K_7 + [\text{MAPK}])$
8	$k_8 \cdot [\text{MKK-PP}] \cdot [\text{MAPK-P}] / (K_8 + [\text{MAPK-P}])$
9	$V_9 \cdot [\text{MAPK-PP}] / (K_9 + [\text{MAPK-PP}])$
10	$V_{10} \cdot [\text{MAPK-P}] / (K_{10} + [\text{MAPK-P}])$

Total concentrations: $[\text{MKKK}]_{\text{total}} = 100$; $[\text{MKK}]_{\text{total}} = 300$; $[\text{MAPK}]_{\text{total}} = 300$

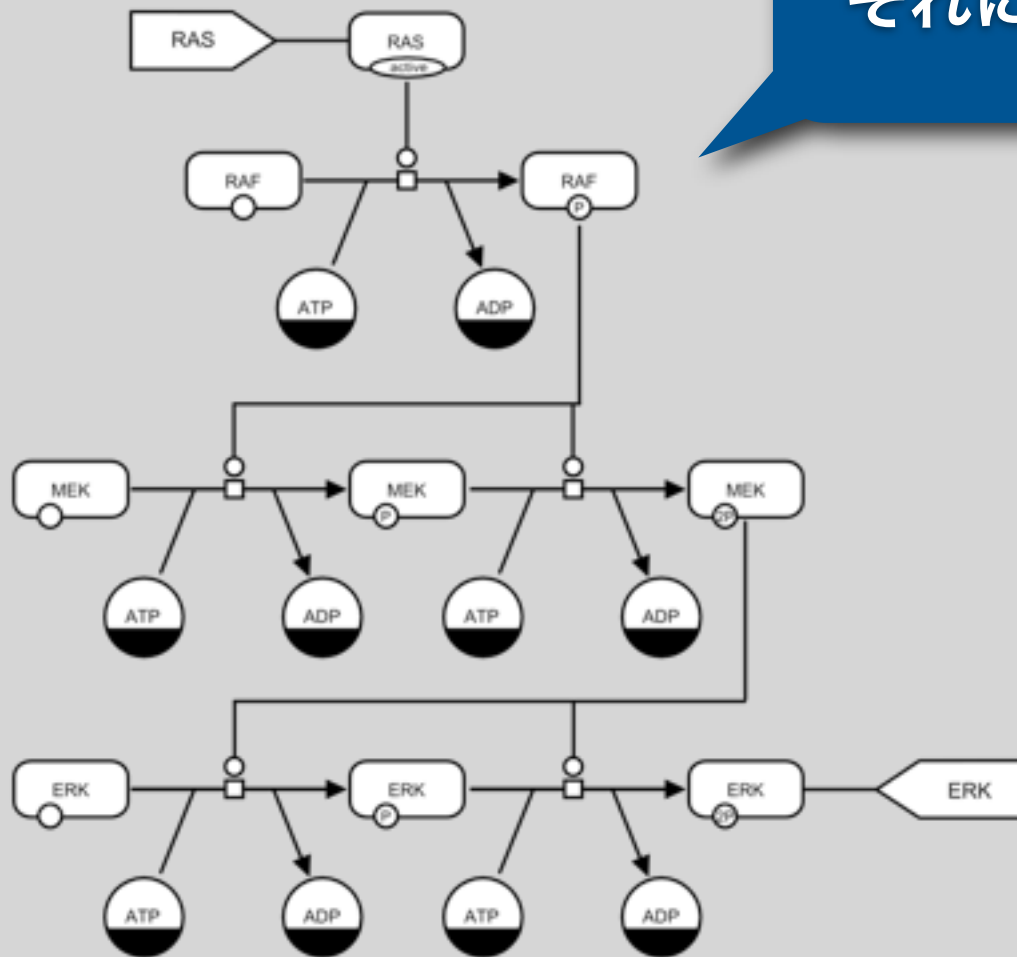
$$[\text{MKK}]_{\text{total}} = [\text{MKK}] + [\text{MKK-P}] + [\text{MKK-PP}]$$

$$[\text{MAPK}]_{\text{total}} = [\text{MAPK}] + [\text{MAPK-P}] + [\text{MAPK-PP}]$$



モデル

対象としている現象
それに関わる分子・生化学反応

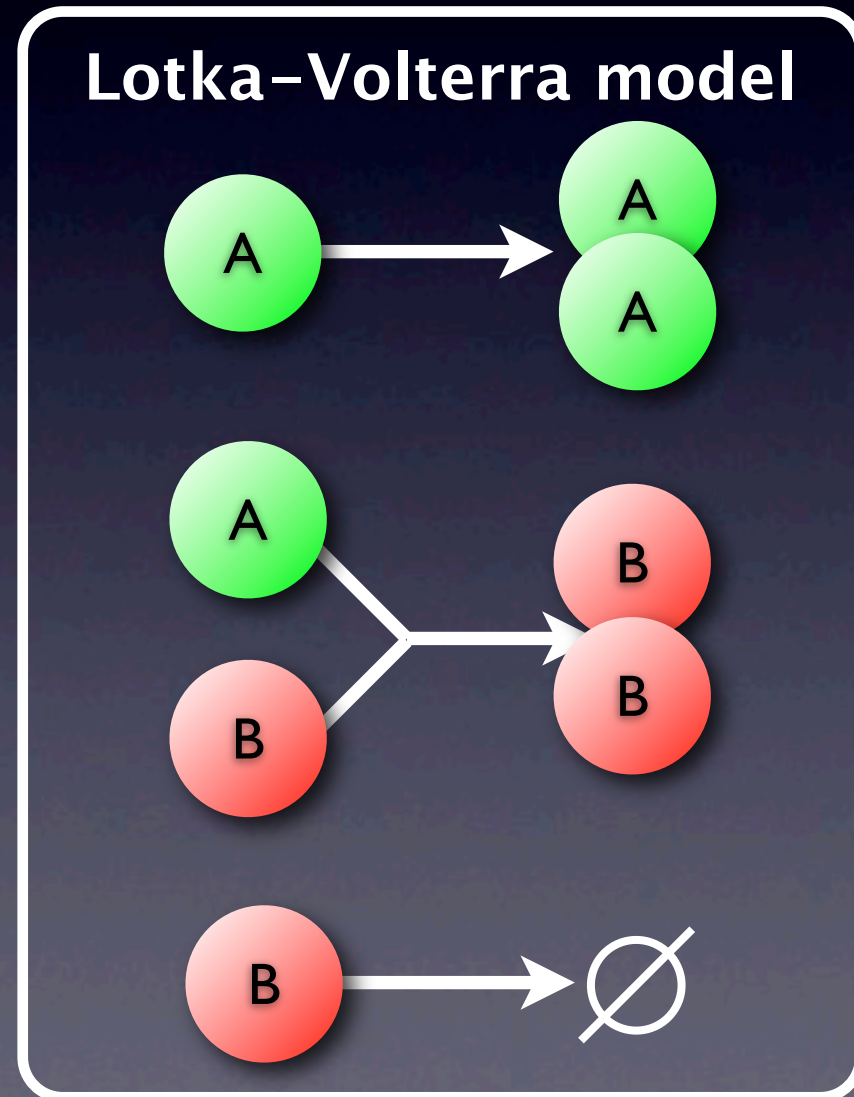
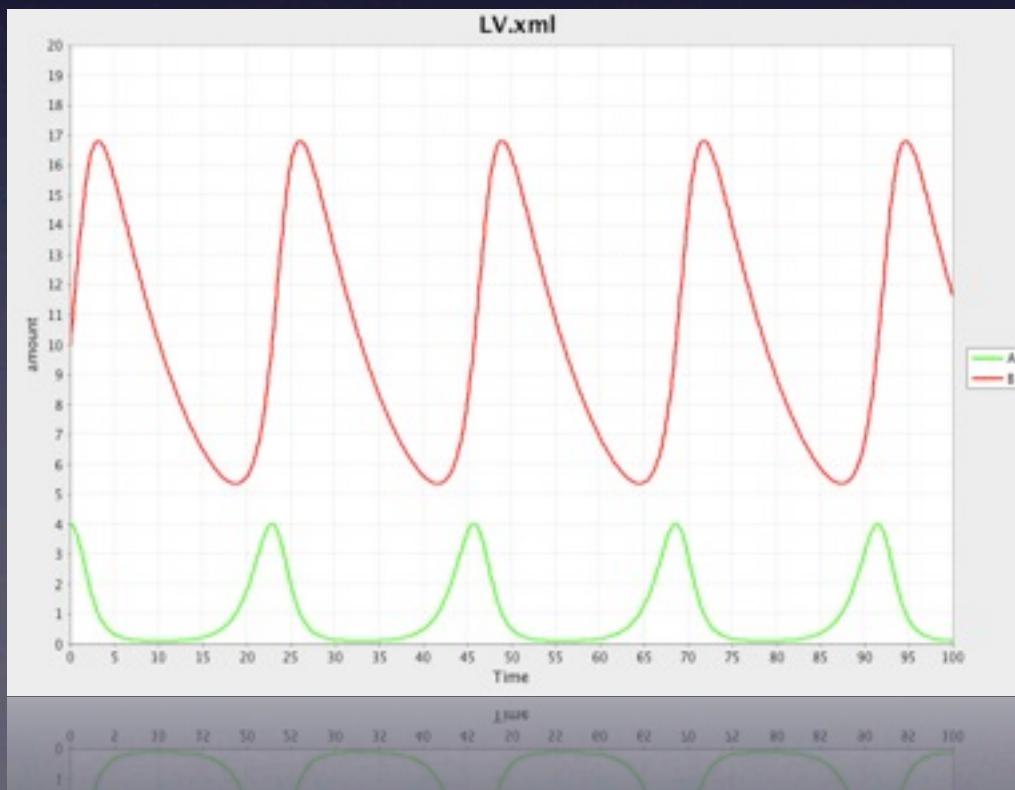


$$\frac{dx}{dt} = kx$$



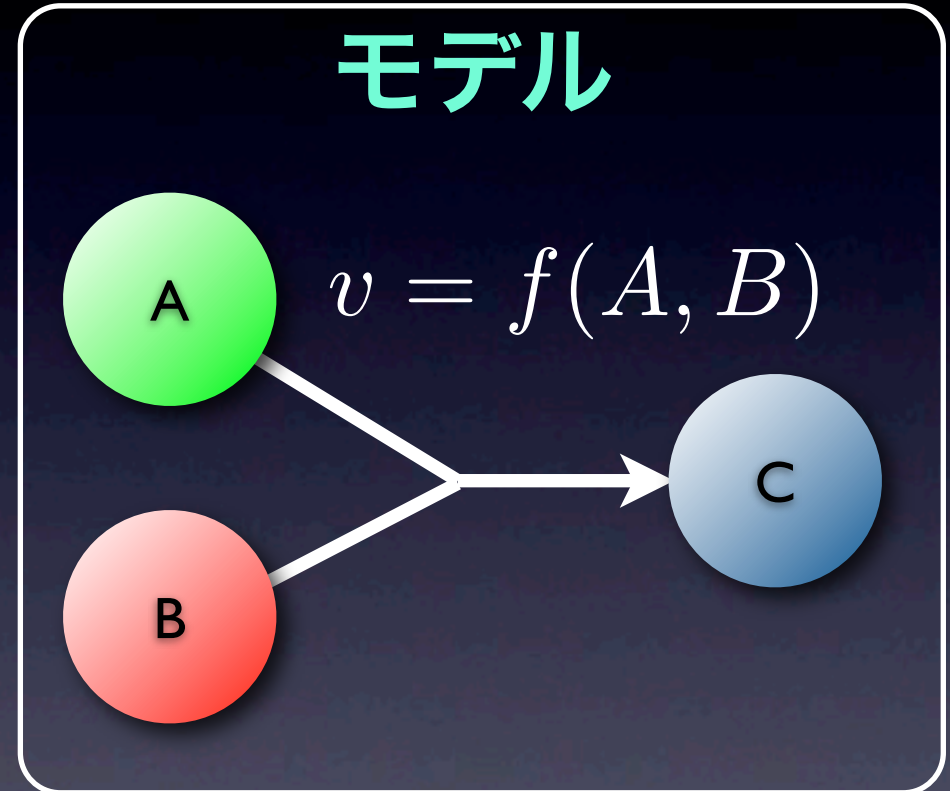
モデル

- シミュレーションしたい対象(現象)
- 抽象化
- 未来の予測



モデル

- 反応方程式
 - 分子濃度・数の増減
- 分子濃度(初期値)

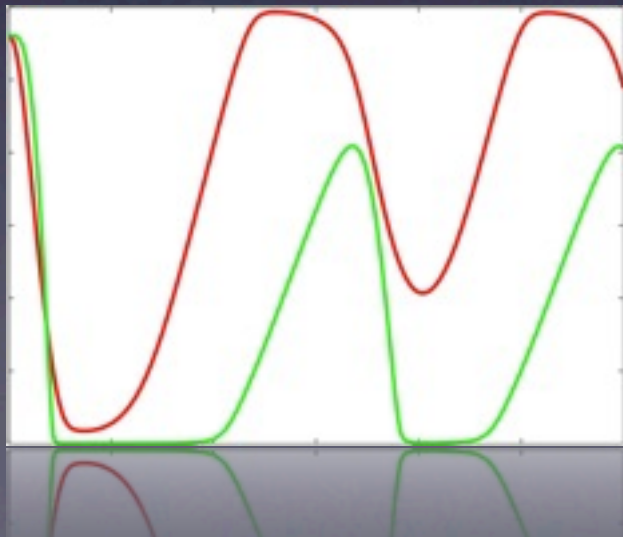


モデル内のすべての分子濃度・反応方程式が
定義されていればシミュレーション可能

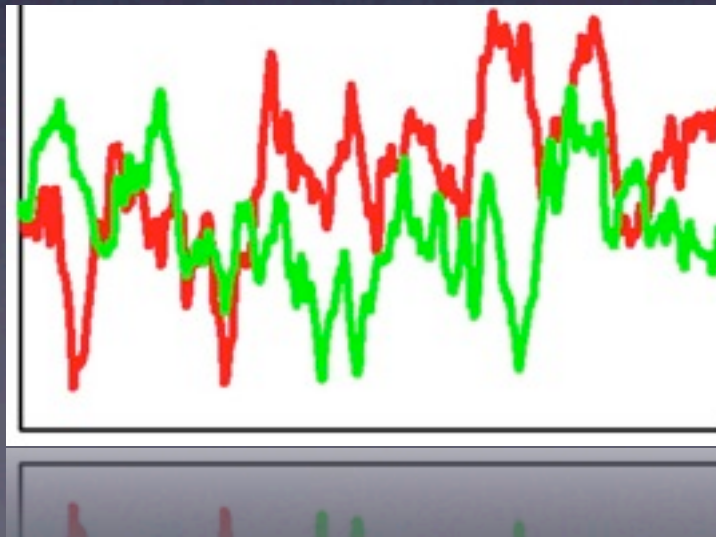
シミュレーション

- 分子濃度を記述した常微分方程式 (ODE)
- 分子数の確率的な変化を記述した確率モデル (SSA)
- 分子濃度の空間的分布を記述した偏微分方程式 (PDE)

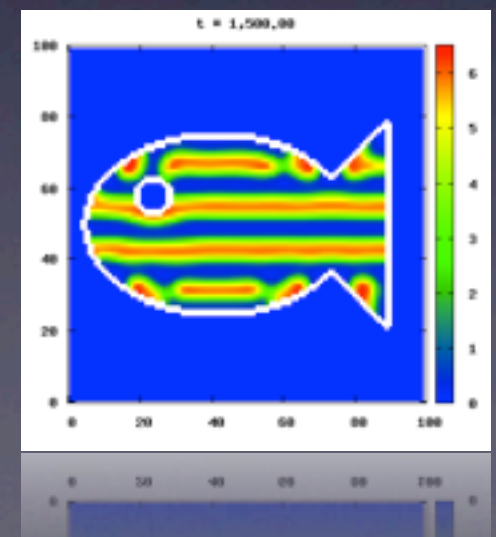
ODE



SSA

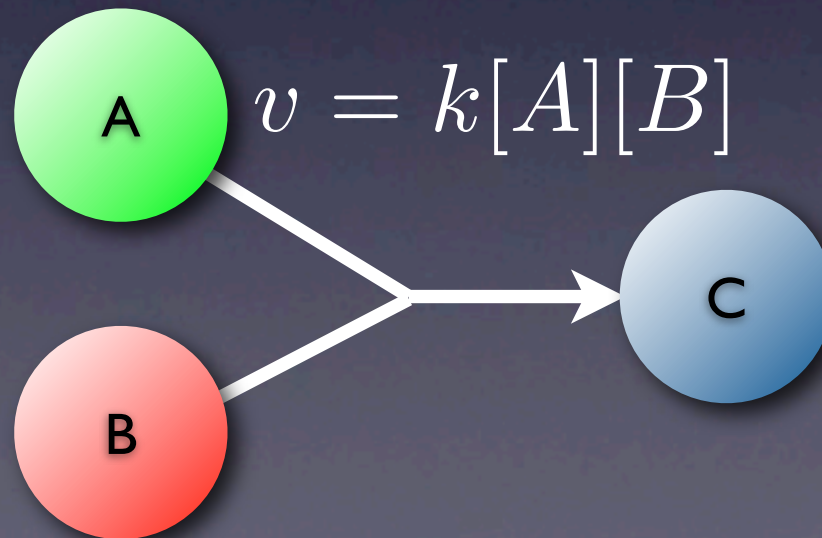


PDE

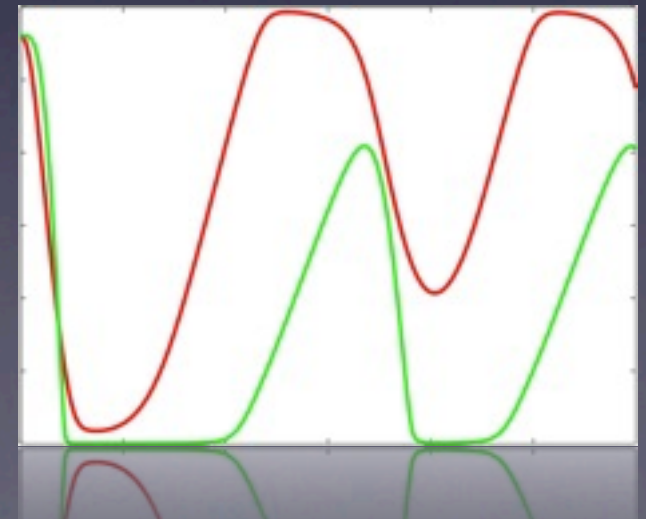


常微分方程式(ODE)

- 分子濃度の変化をモデル化
- 各反応による分子濃度の変化を記述
- 連続的な値を取り扱う
- 決定論的
- 数値積分



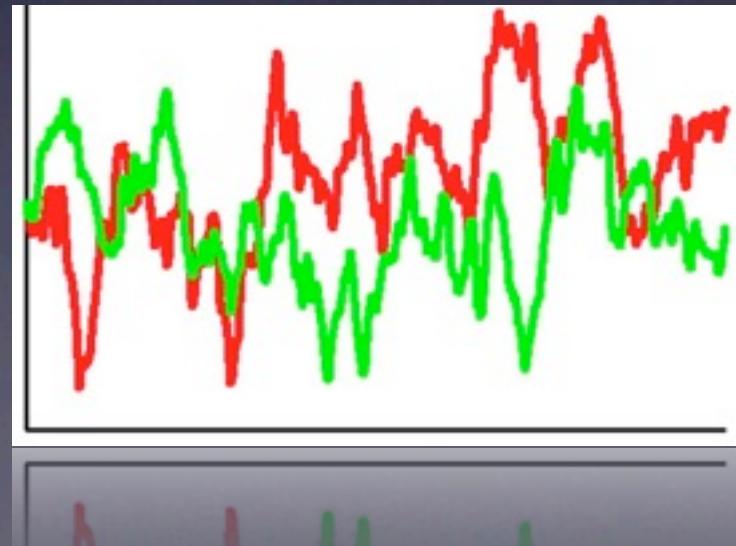
ODE



確率モデル(SSA)

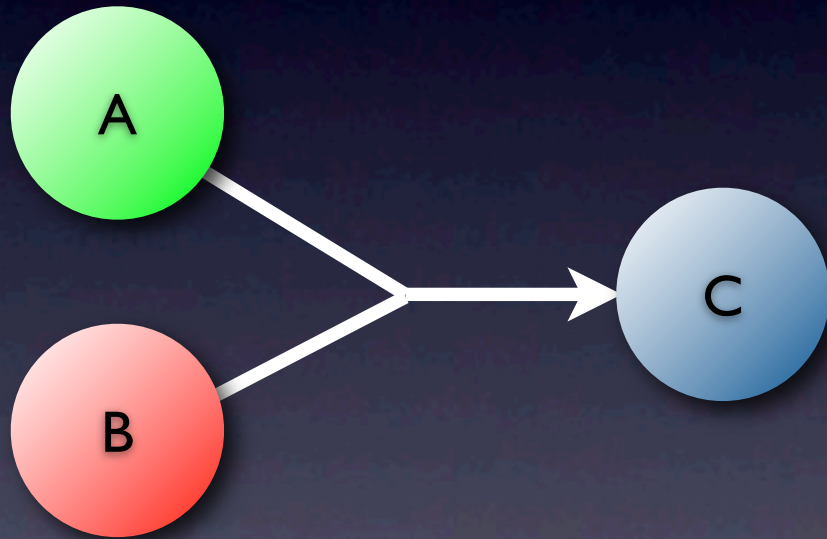
- 分子数の変化をモデル化
- 各反応による分子数の変化を記述
- 連続的な値ではなく、離散値を取り扱う
- 非決定論的

SSA

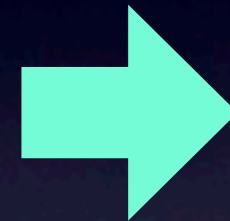


確率モデル(SSA)

- 分子数の変化をモデル化

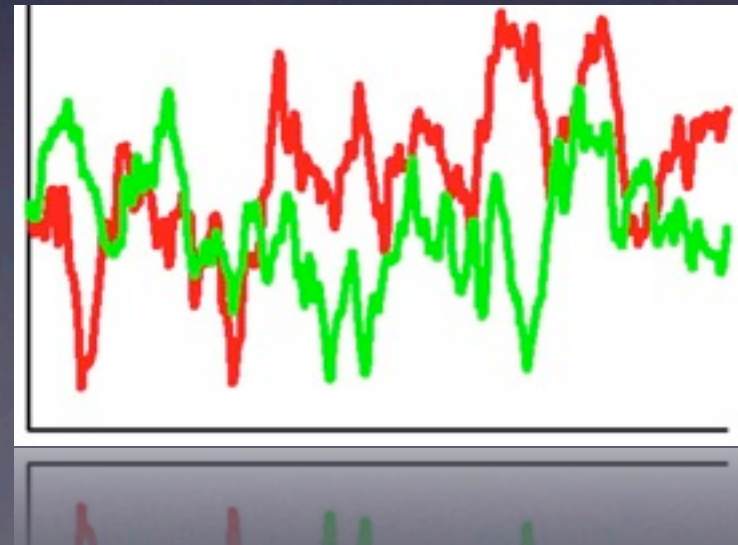


A: 300
B: 200
C: 0



A: 298
B: 198
C: 2

SSA



偏微分方程式(PDE)

分子濃度: 高

分子濃度: 低

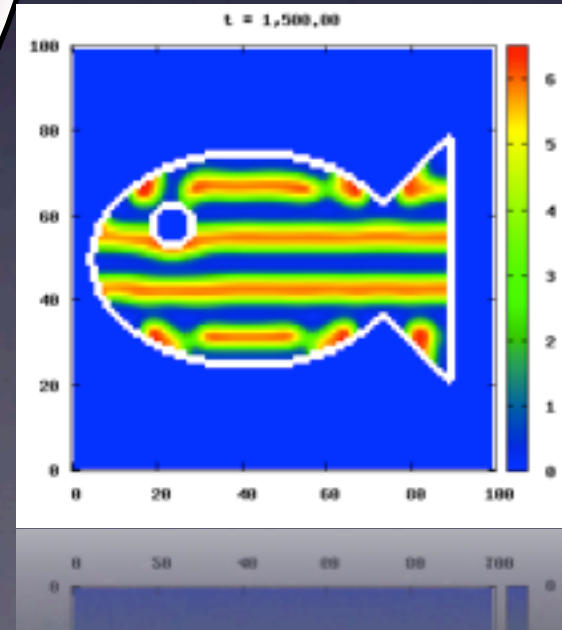
分子の拡散

分子の移流

境界での
膜輸送

核

細胞質



ODE? SSA? PDE?

Applications and trends in systems biology in biochemistry

Katrin Hübner, Sven Sahle and Ursula Kummer

Department of Modeling of Biological Processes, COS Heidelberg/BioQuant, University of Heidelberg, Germany

Keywords

metabolism; modeling; quantitative experiments; signaling; simulation; systems biology

Correspondence

U. Kummer, Department of Modeling of Biological Processes, COS Heidelberg/BioQuant, University of Heidelberg, Im Neuenheimer Feld 267, 69120 Heidelberg, Germany
Fax: +49 6221 5451483
E-mail: ursula.kummer@bioquant.uni-heidelberg.de

(Received 10 January 2011; revised 31 May 2011; accepted 15 June 2011)

doi:10.1111/j.1742-4658.2011.08217.x

Introduction

One of the fastest growing fields in the life sciences is systems biology. PubMed lists more than 3000 articles which, in one way or the other, use this term in their title or abstract during the last decade (precisely, the last 11 years, including the year 2000) compared to a mere three articles in the preceding century. Obviously, this is partially a result of the fact that the term 'systems biology' had not been used during that time. However, as we will see in the present review, also with respect to research that would now be called systems biology, there is clearly significantly less to report before the year 2000. Interestingly, looking closely at the more than 3000 articles using the term 'systems biology', it becomes apparent that approximately half of them describe methodological work either on the computational or the experimental side, and more than one-third are classified as reviews. However, only a

Systems biology has received an ever increasing interest during the last decade. A large amount of third-party funding is spent on this topic, which involves quantitative experimentation integrated with computational modeling. Industrial companies are also starting to use this approach more and more often, especially in pharmaceutical research and biotechnology. This leads to the question of whether such interest is wisely invested and whether there are success stories to be told for basic science and/or technology/biomedicine. In this review, we focus on the application of systems biology approaches that have been employed to shed light on both biochemical functions and previously unknown mechanisms. We point out which computational and experimental methods are employed most frequently and which trends in systems biology research can be observed. Finally, we discuss some problems that we have encountered in publications in the field.

handful of the latter represent reviews that actually review a set of articles. Most of the articles classified as reviews could rather be classified as news and views. Another large portion of articles uses the term 'systems biology' in a different sense than we would understand it (e.g. stating that they are investigating a biological system and it is therefore systems biology). This latter point necessitates the definition of the term 'systems biology' as we (the authors) understand it, as outlined below.

Systems biology combines quantitative experimental data from complex molecular networks (e.g. biochemistry, cell biology in the living cell) with computational modeling. Here, computational modeling does not refer to statistical models or models of data mining but rather to a mathematical or 'virtual' representation of the living system of interest in the computer, where

Abbreviations

FBA, flux balance analysis; ODE, ordinary differential equation; PDE, partial differential equation.

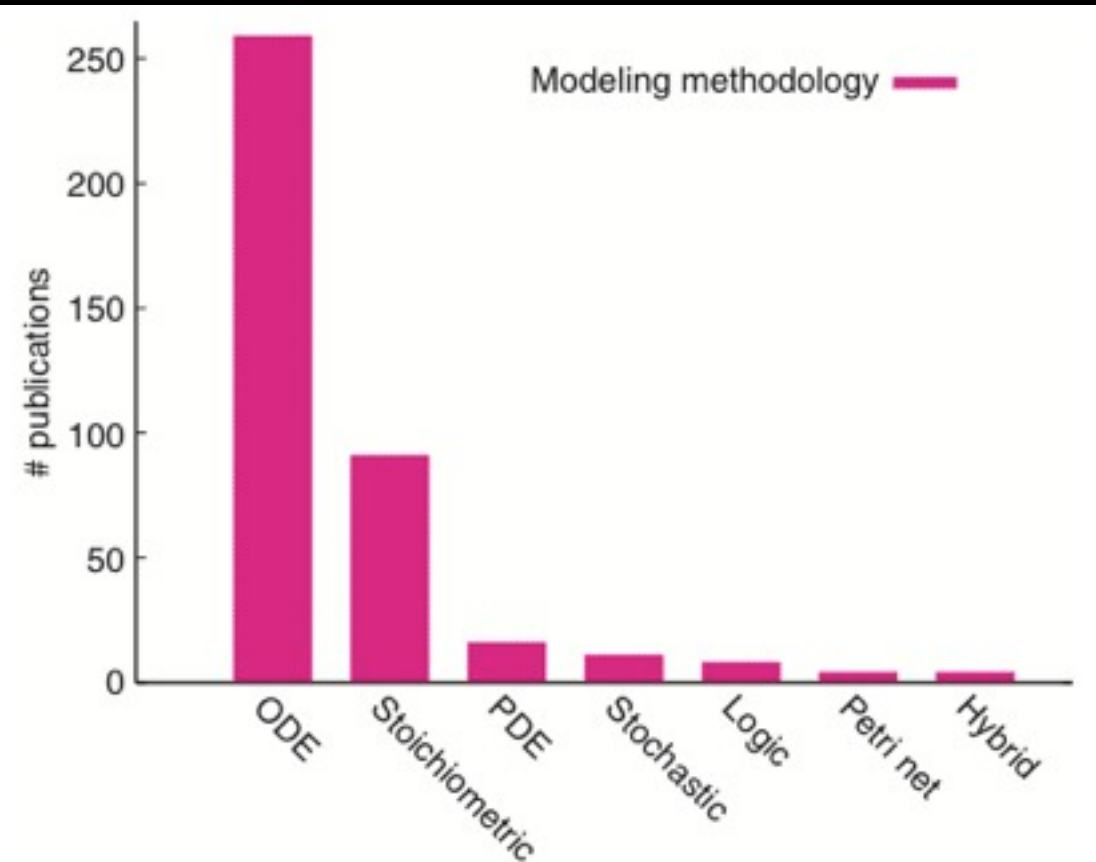
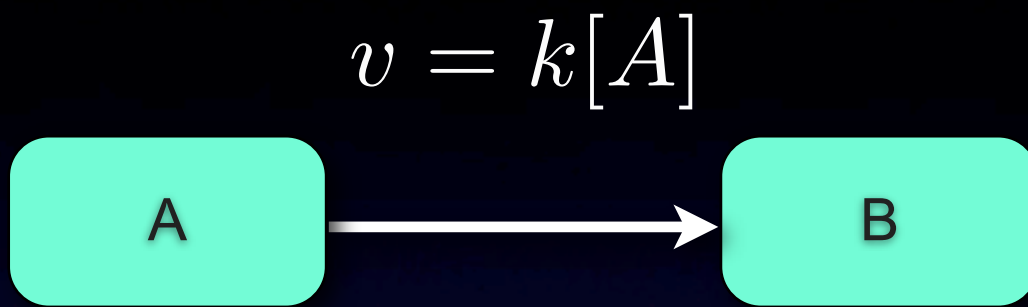
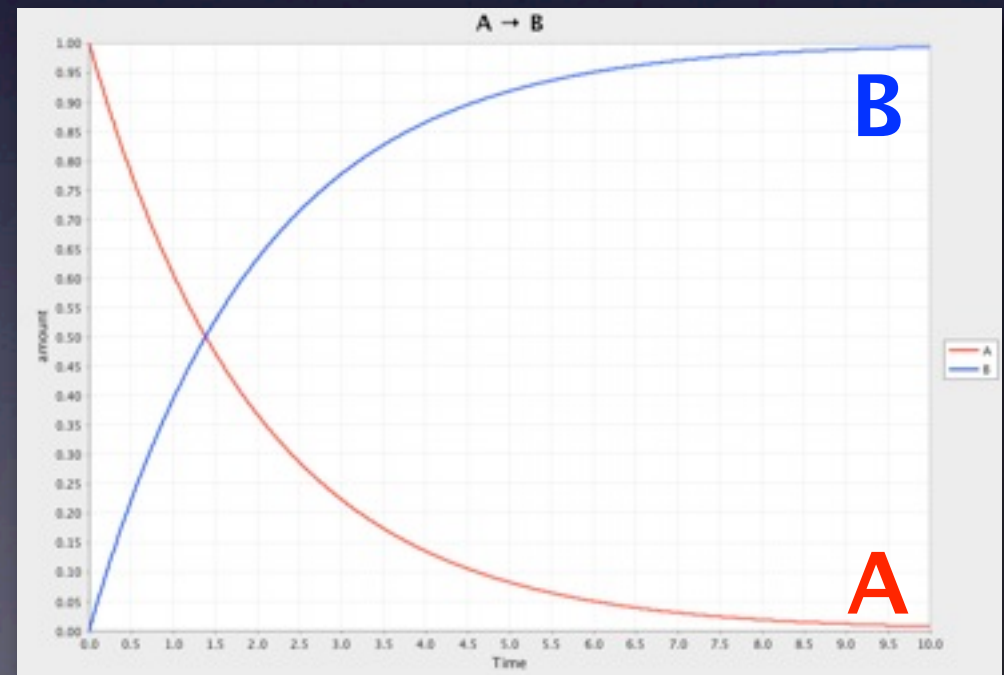
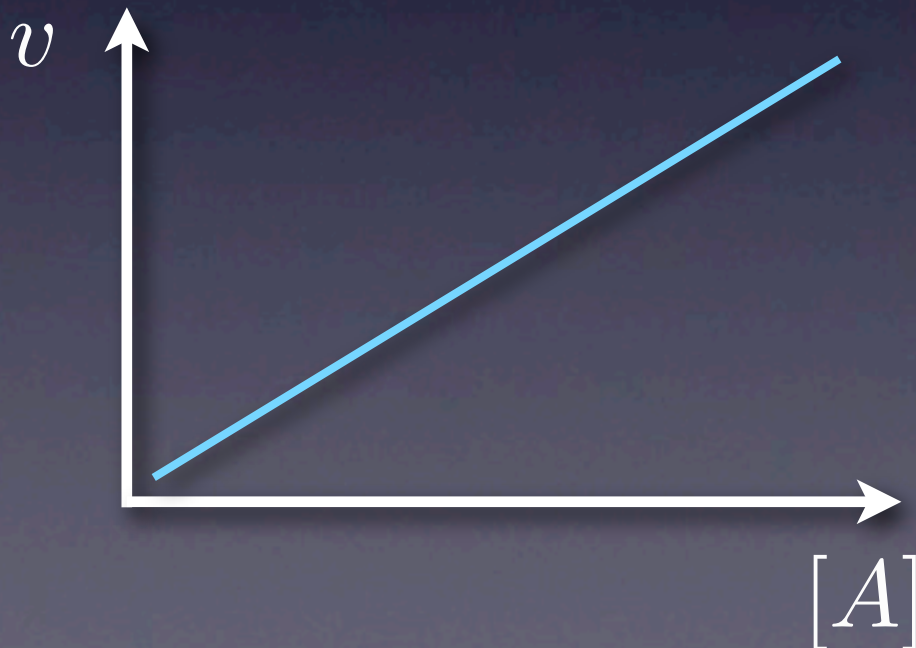


Fig. 7. Number of publications describing systems biology applied to biochemistry in the years 2000–2010 using a specific computational modeling approach.

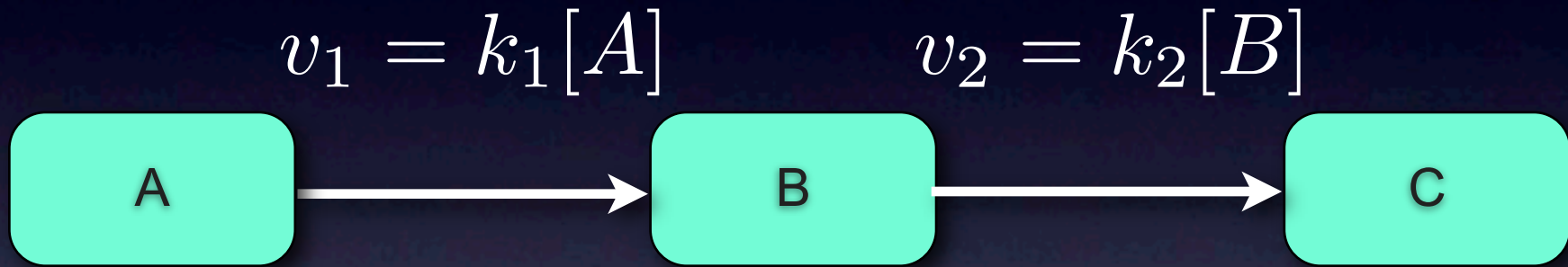
ODEで書かれた反応方程式



$$\frac{d[A]}{dt} = -k[A]$$
$$\frac{d[B]}{dt} = k[A]$$

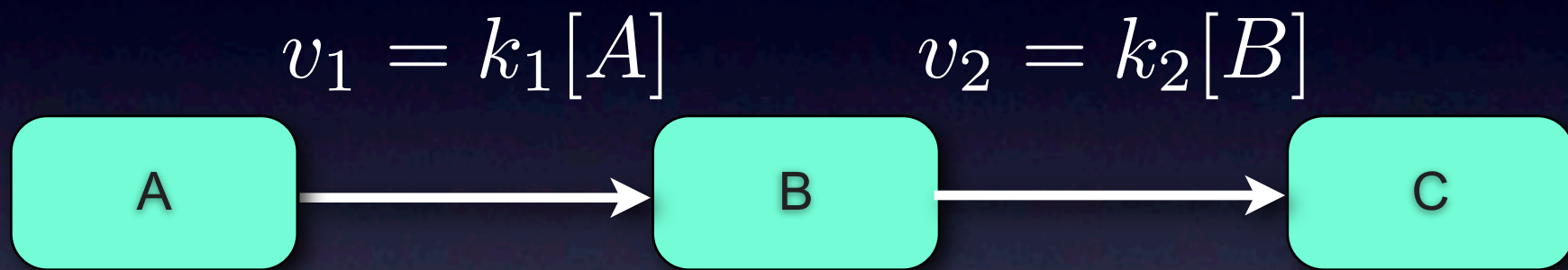


Mass-action



$$\frac{d[B]}{dt} = k_1[A] - k_2[B]$$

大事なこと



$$\frac{d[B]}{dt} = k_1[A] - k_2[B]$$

● 和、差で反応毎に分解できる

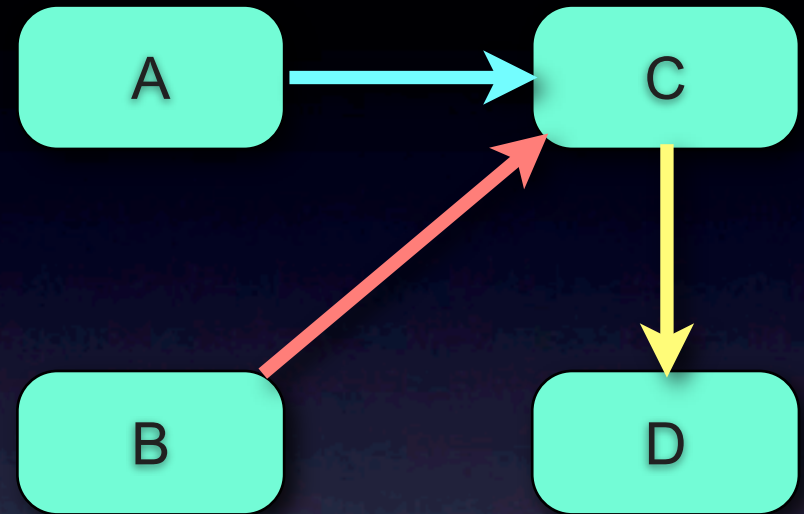
ODE \leftrightarrow ネットワーク

$$\frac{dA}{dt} = -k_1 A$$

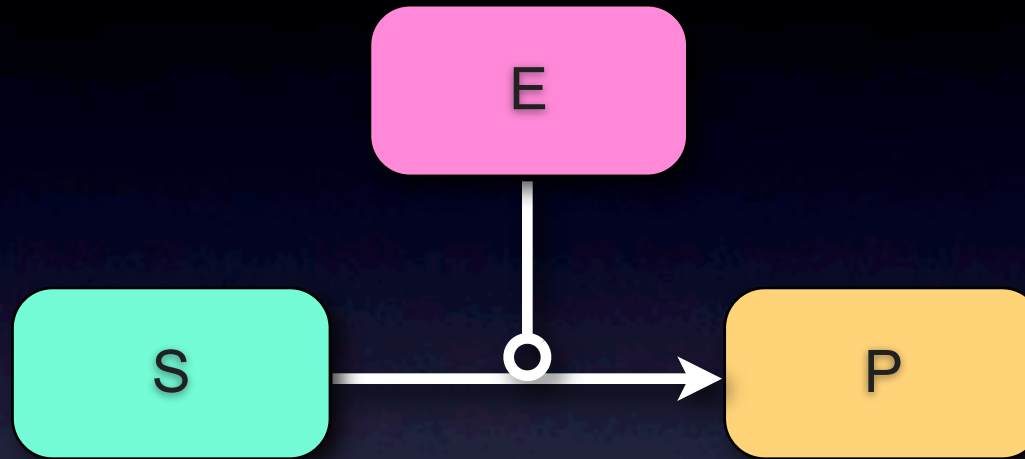
$$\frac{dB}{dt} = -k_2 B$$

$$\frac{dC}{dt} = k_1 A + k_2 B - k_3 C$$

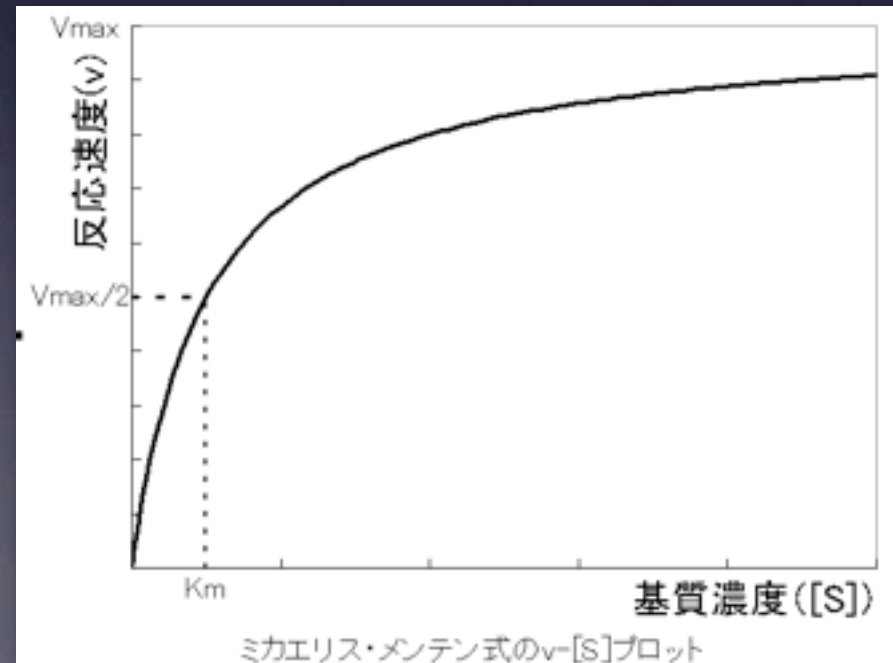
$$\frac{dD}{dt} = k_3 C$$



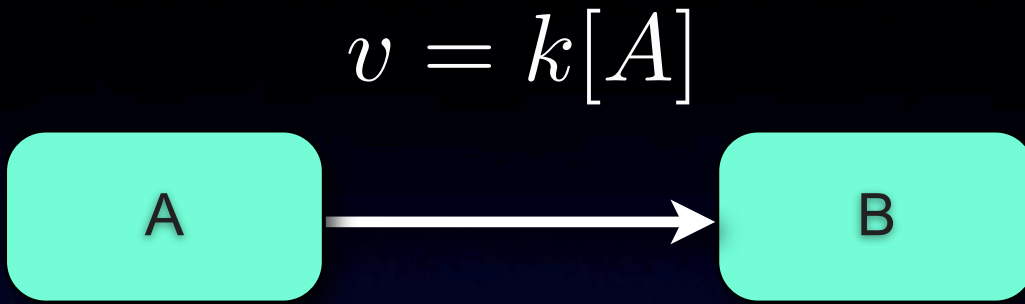
Michaelis-Menten



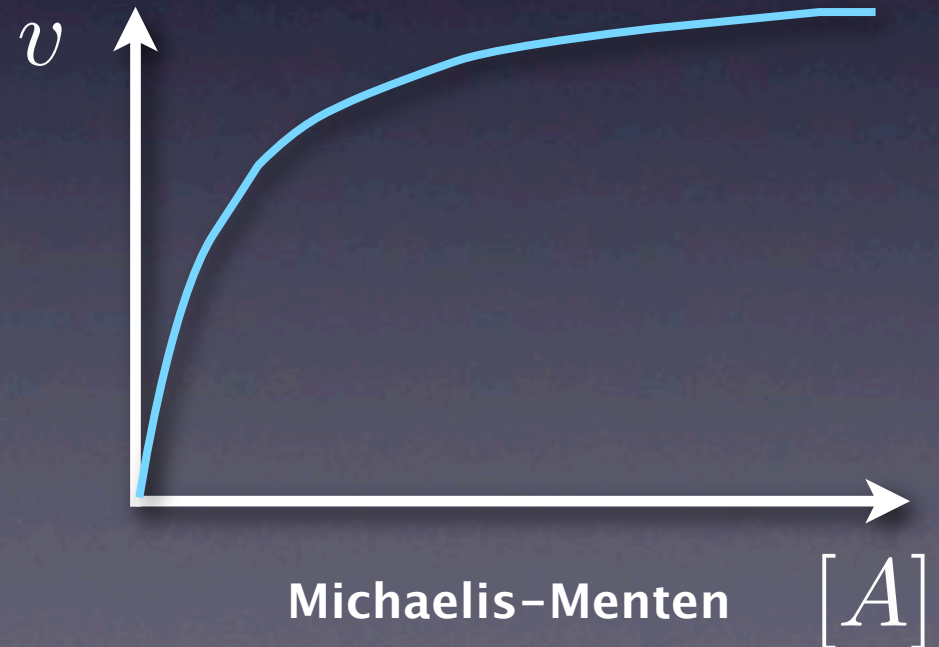
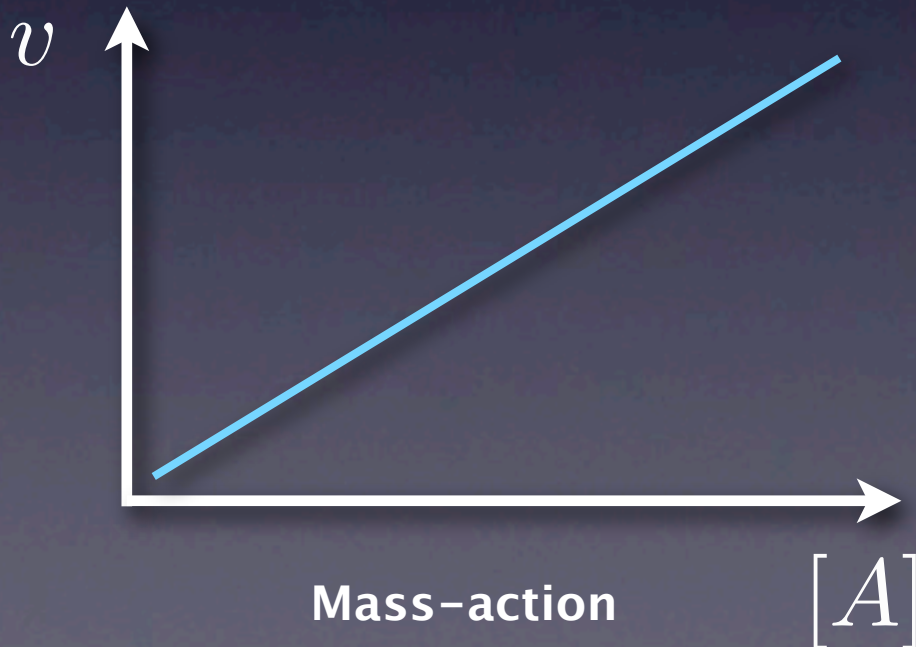
$$\frac{d[P]}{dt} = \frac{V_{max} [S]}{K_m + [S]}$$



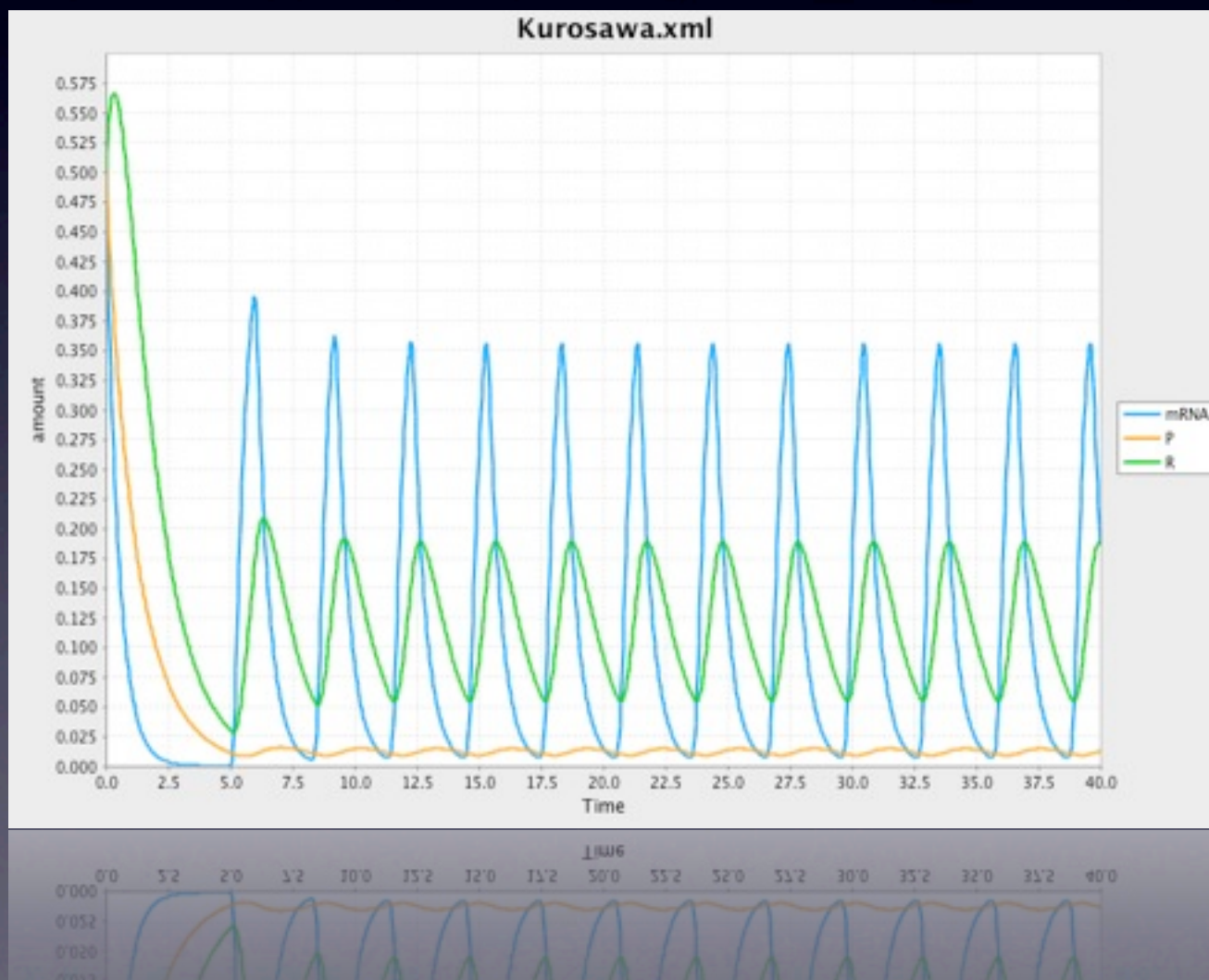
Mass-action



$$\frac{d[A]}{dt} = -k[A]$$
$$\frac{d[B]}{dt} = k[A]$$

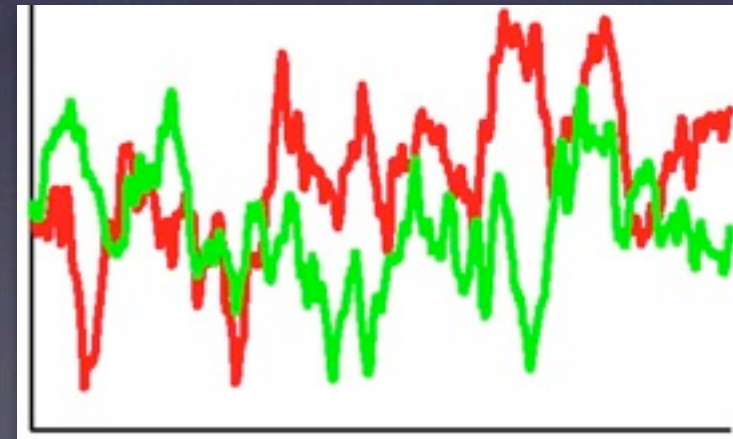
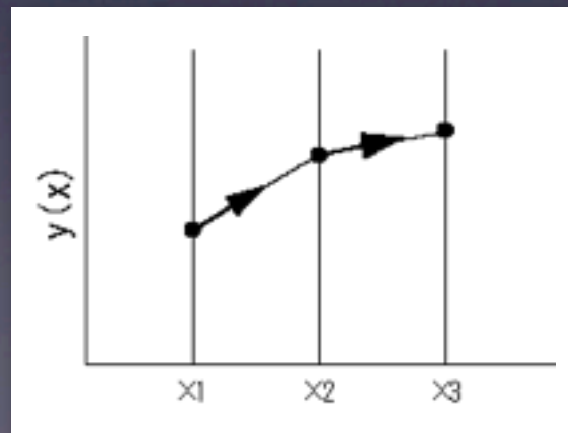


ODEシミュレータの実装



Why Simulate a model?

- 実験で(検証する前に/検証できない事項を)予測したい
- 未来 (ある時間: t) の状態を予測



未来の予測

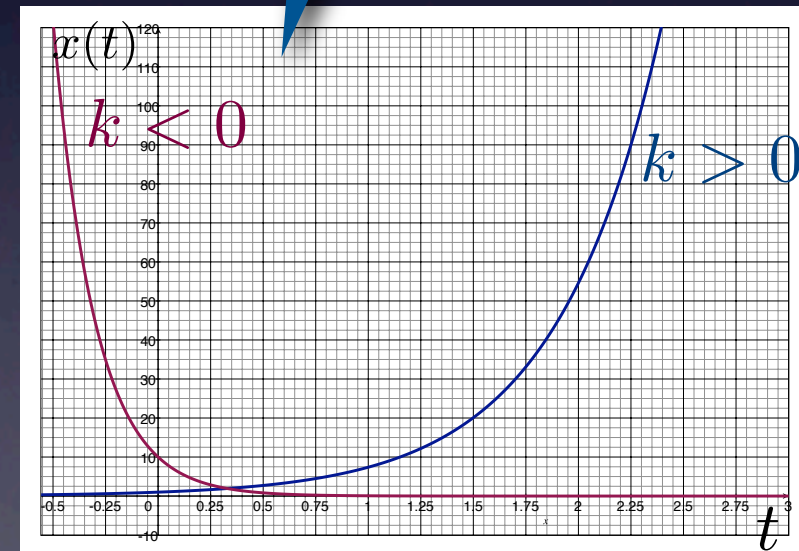
- 未来 (ある時間: t) の状態を予測



解析的に未来を予測



$$\frac{dx}{dt} = kx$$



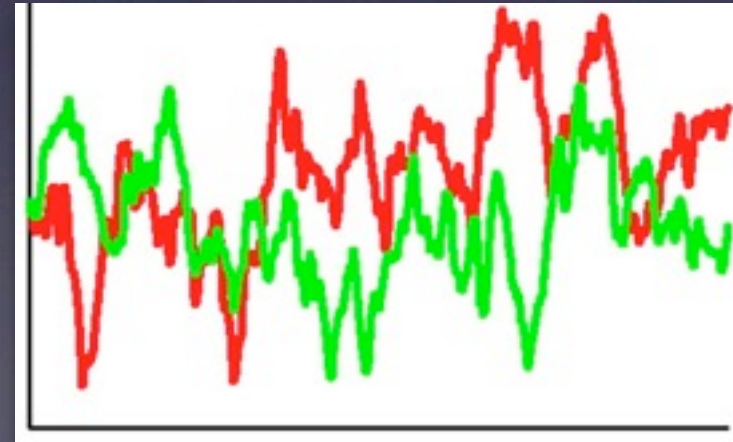
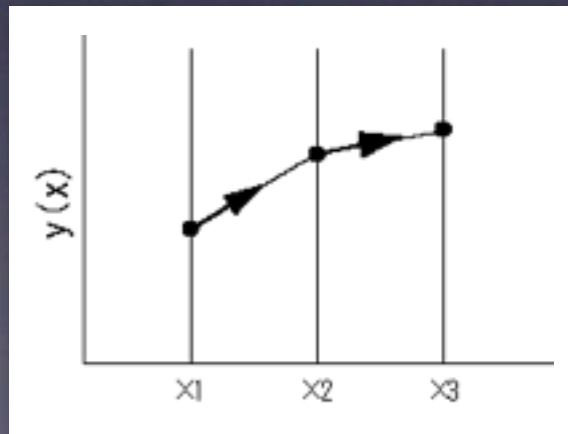
$$x = C_1 e^{kt}$$

数値計算が必要な場合

- 解析的に解けない時
- x_{t+1} を x_t から求めるしかない時



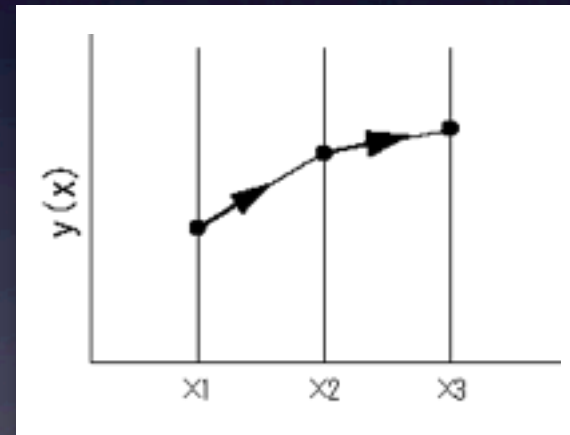
- 数値計算
- 確率モデル



数値計算が必要な場合

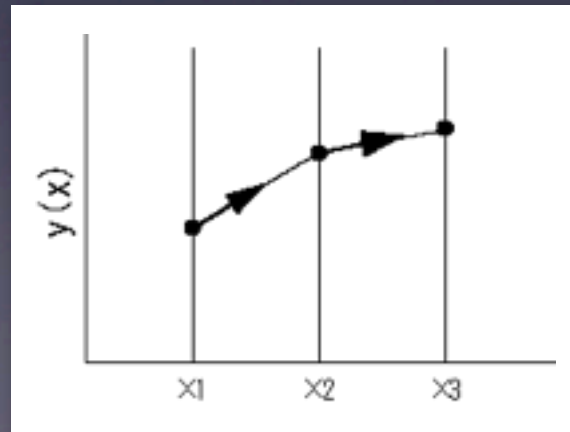
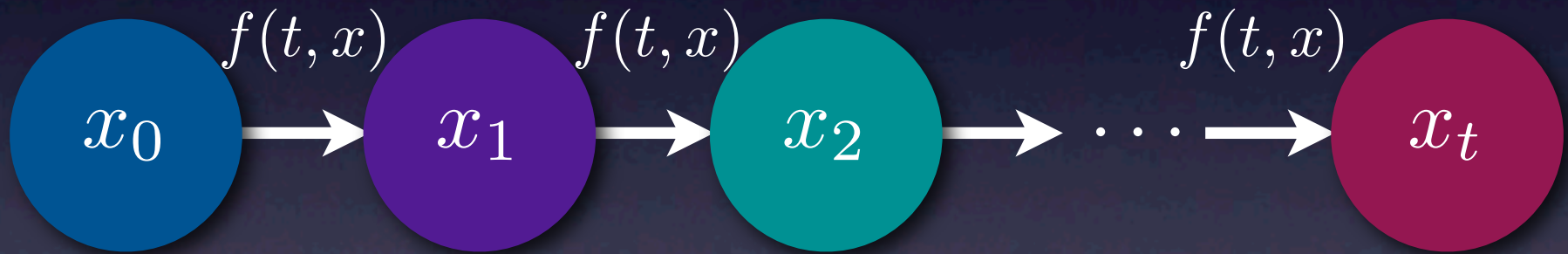
- 多くの場合、連立微分方程式は解析的に解けない
→ 積分できない
- 数値積分が必要

ODEシミュレータは
数値積分を行なっている



ODEの数値積分

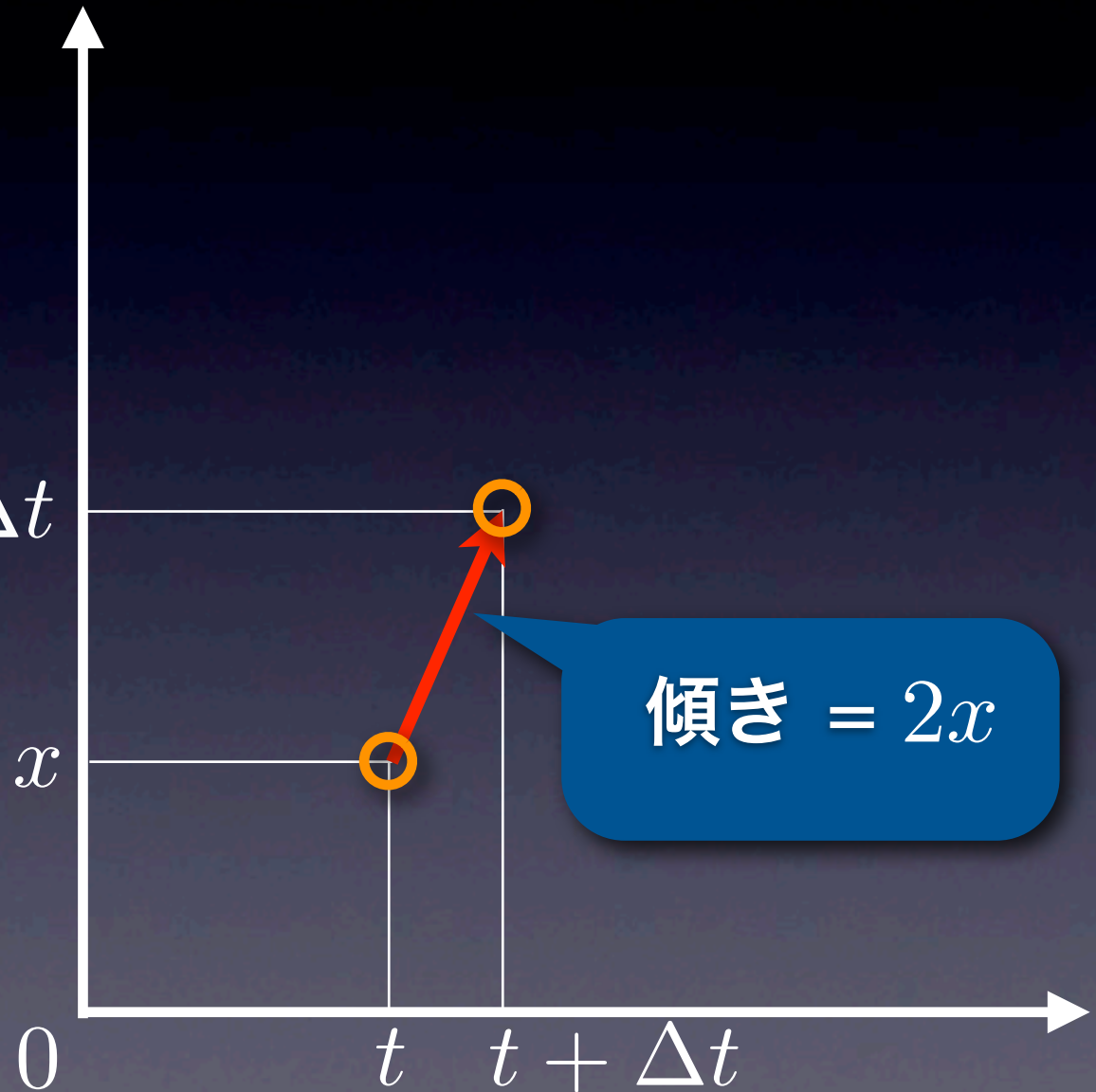
$$\frac{dx}{dt} = f(t, x)$$



ODE Simulatorの作り方

$$\frac{dx}{dt} = 2x$$

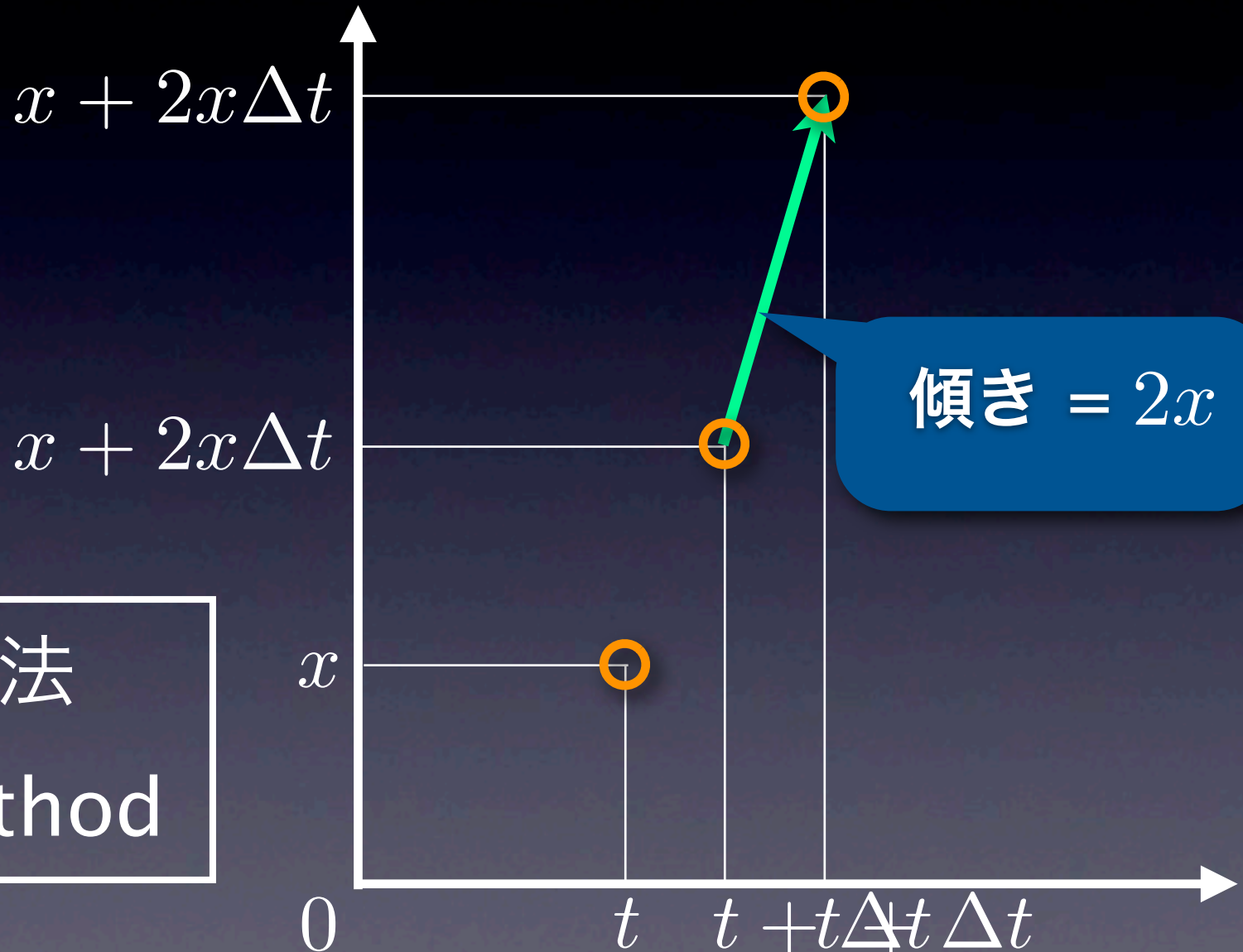
$$x + 2x\Delta t$$



ODE Simulatorの作り方

$$\frac{dx}{dt} = 2x$$

オイラー法
Euler's Method

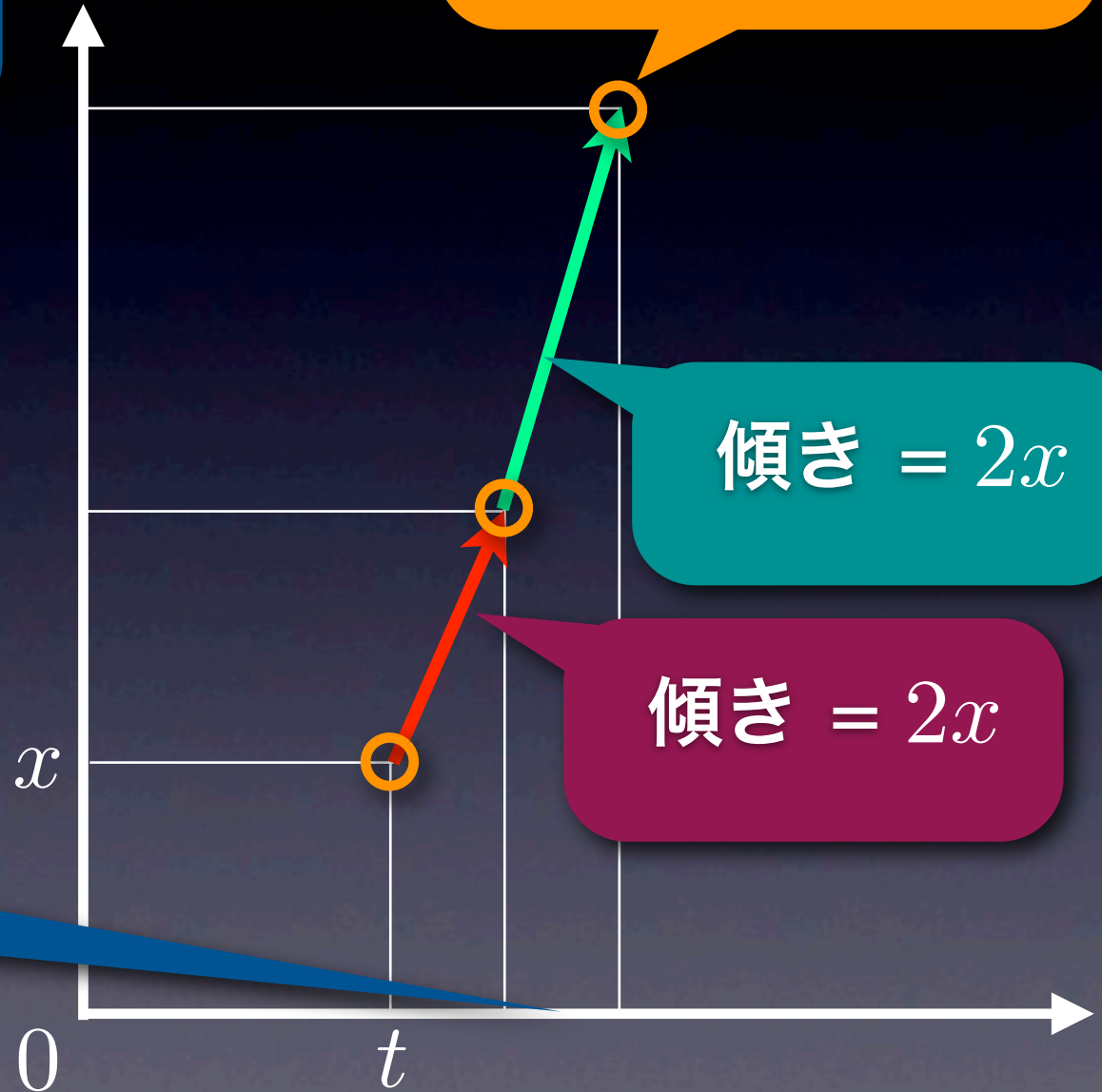


ポイント

x は dx に応じて変化

$$\frac{dx}{dt} = 2x$$

t は Δt で増加



ODE Simulator

- $\frac{dx}{dt} = 2x$ を解くシミュレータ $t = 0, x = 1.0$

```
#!/usr/bin/perl
```

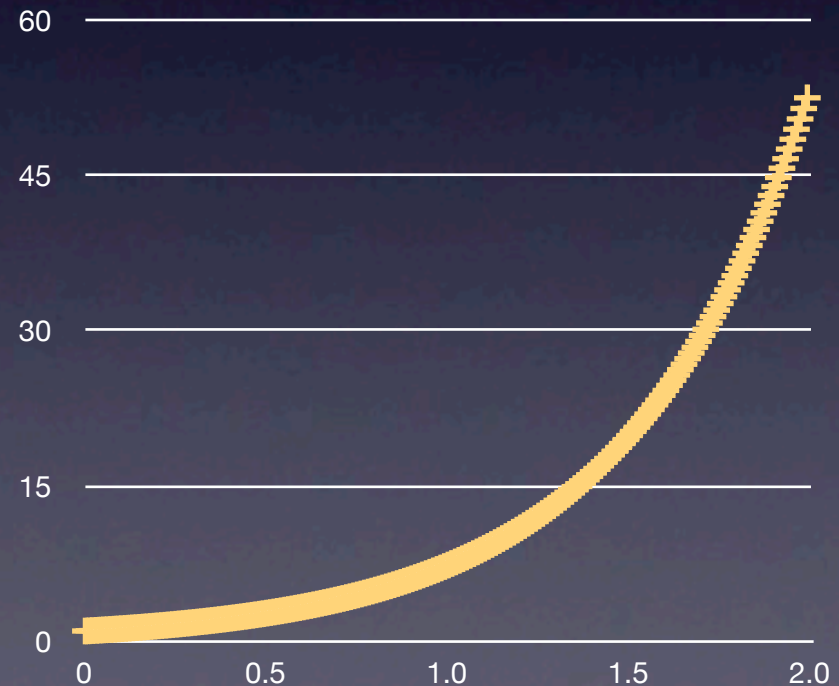
```
$dt = 0.01;
```

```
$t = 0.0;
```

```
$x = 1.0;
```

```
for ($i = 0; $i <= 200; $i++) {  
    print "$t,$x\n";  
    $dx = 2 * $x * $dt;  
    $x = $x + $dx;  
    $t = $t + $dt;  
}
```

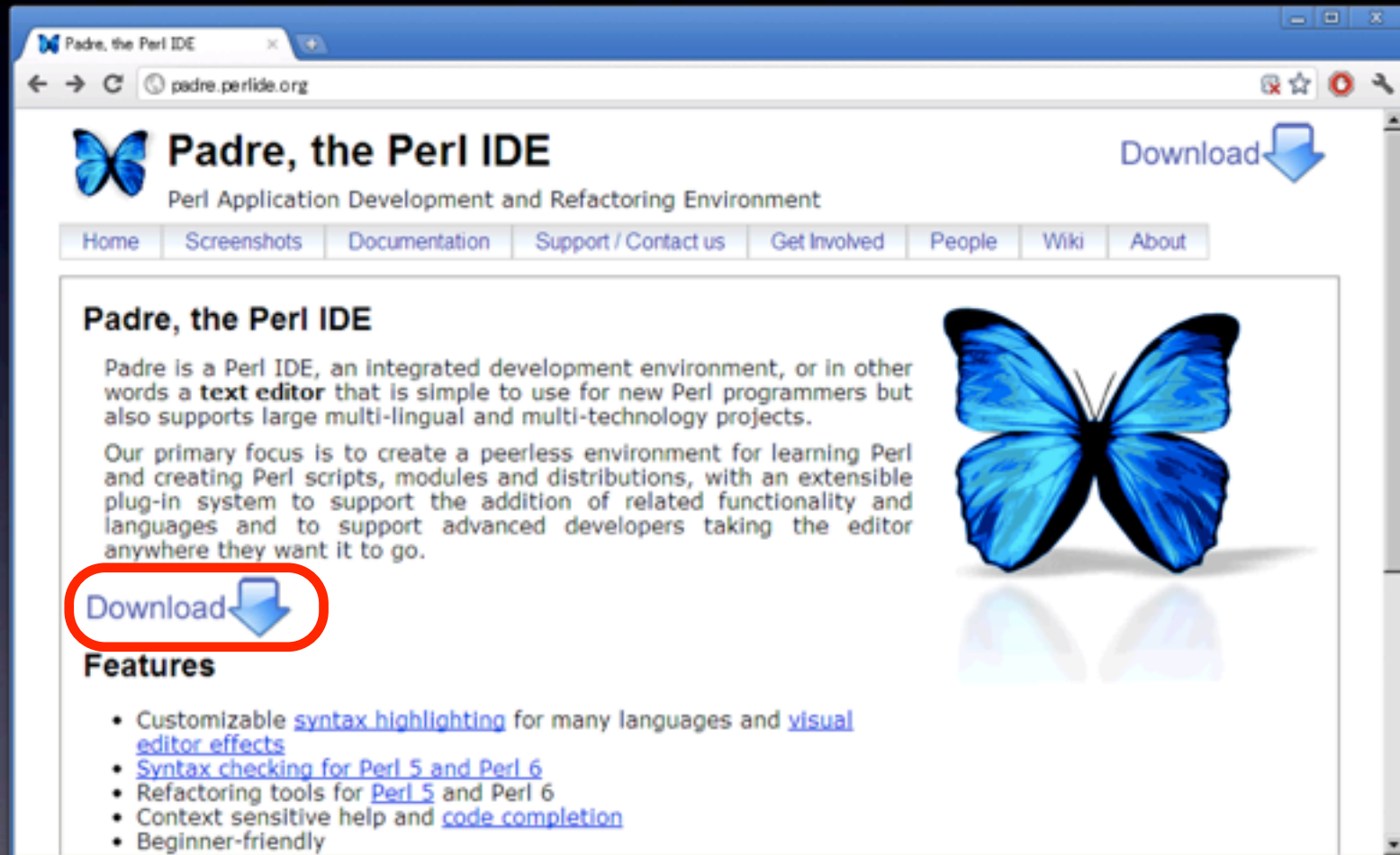
シミュレーション結果



PerlでODEシミュレータ

Windowsの人

<http://padre.perlide.org>

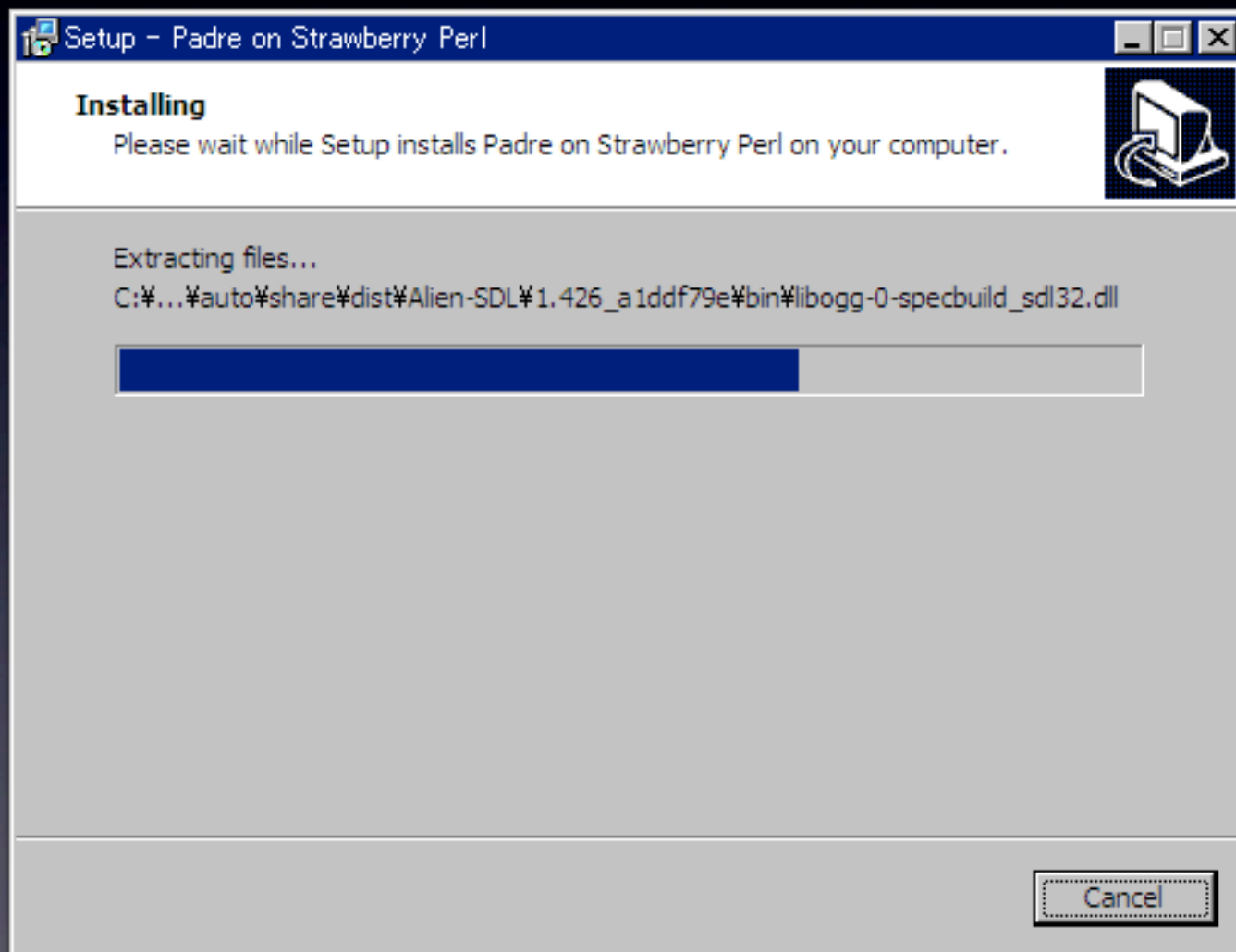


The screenshot shows a web browser window with the address bar displaying "padre.perlide.org". The page title is "Padre, the Perl IDE" and the subtitle is "Perl Application Development and Refactoring Environment". A navigation menu includes links for Home, Screenshots, Documentation, Support / Contact us, Get Involved, People, Wiki, and About. A large blue butterfly logo is featured on the right side of the page. The main content area contains a "Download" button with a downward arrow icon, which is circled in red. Below this, the "Features" section lists several capabilities:

- Customizable [syntax highlighting](#) for many languages and [visual editor effects](#)
- [Syntax checking for Perl 5 and Perl 6](#)
- Refactoring tools for [Perl 5](#) and Perl 6
- Context sensitive help and [code completion](#)
- Beginner-friendly

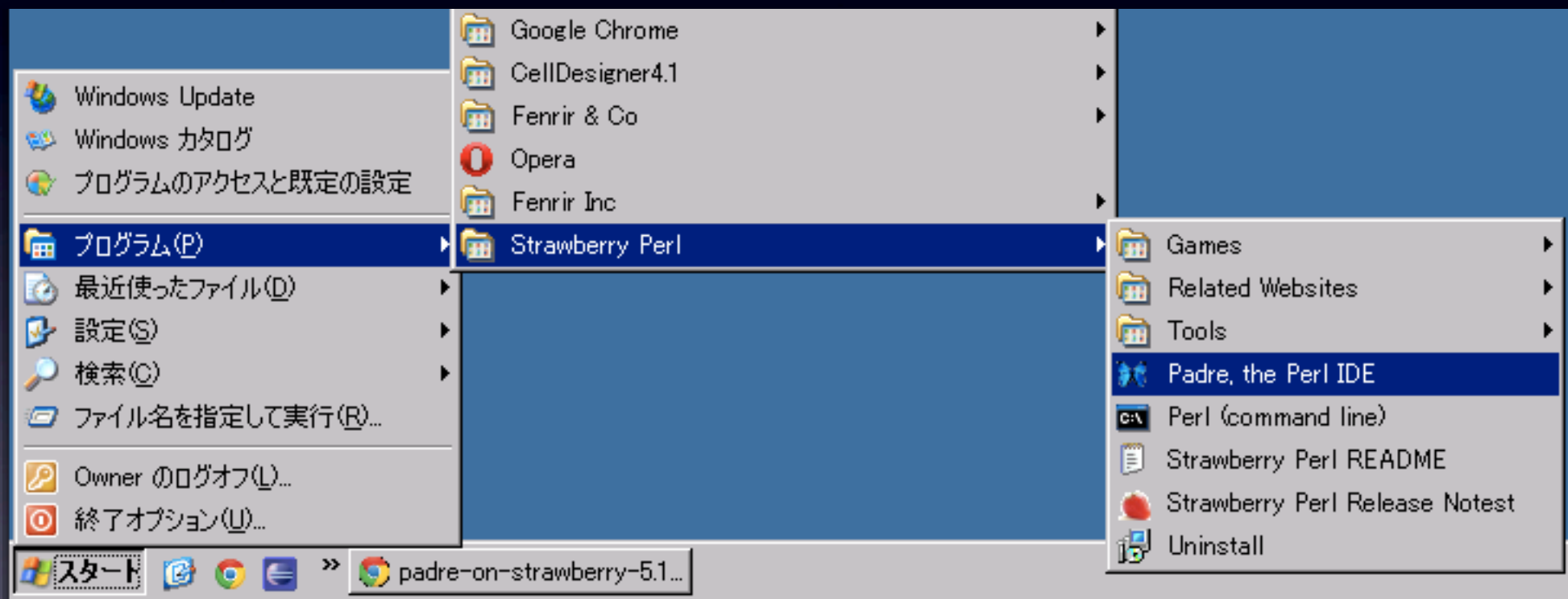
Padreをinstall

<http://padre.perlide.org>

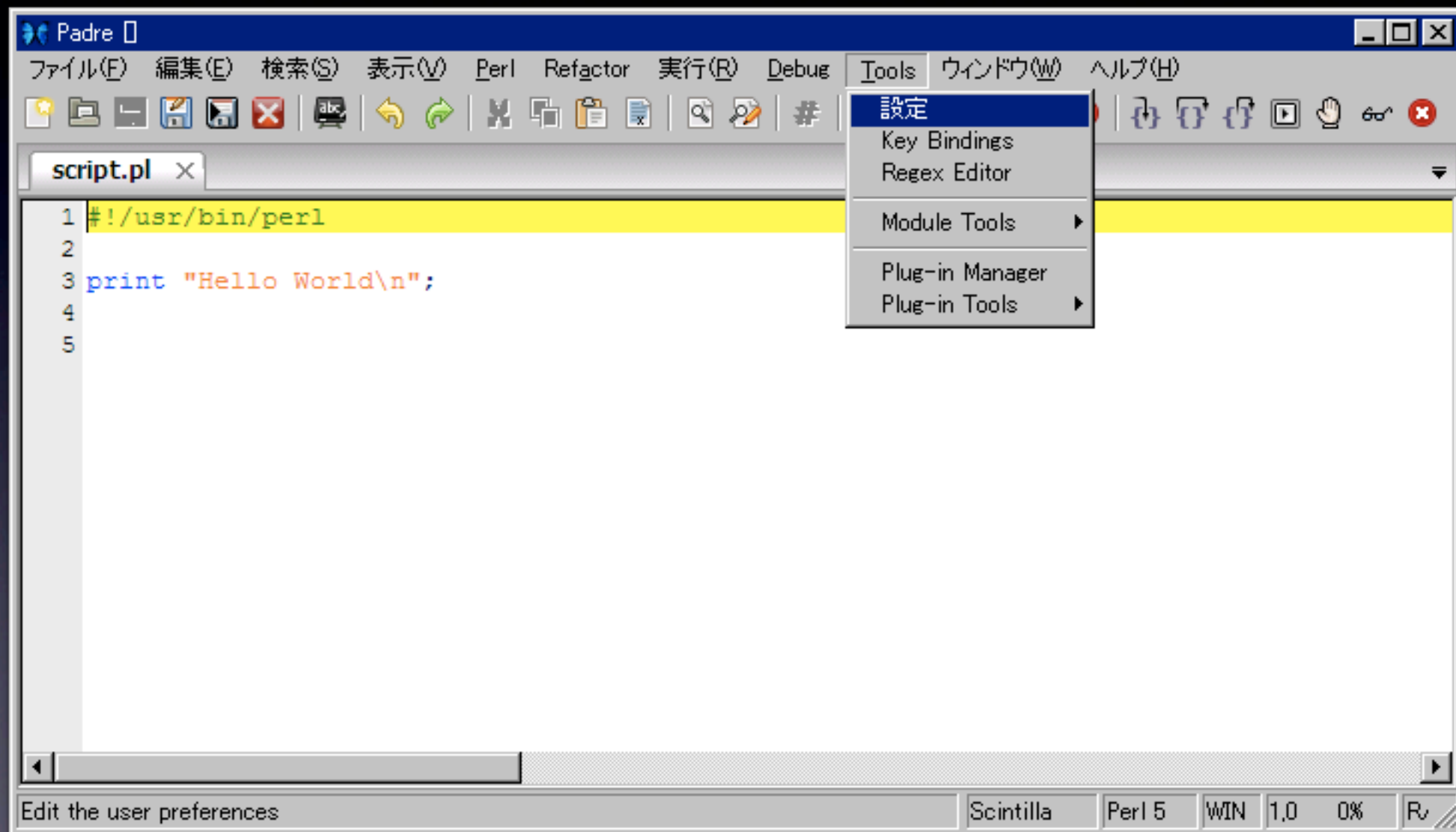


Padreを実行

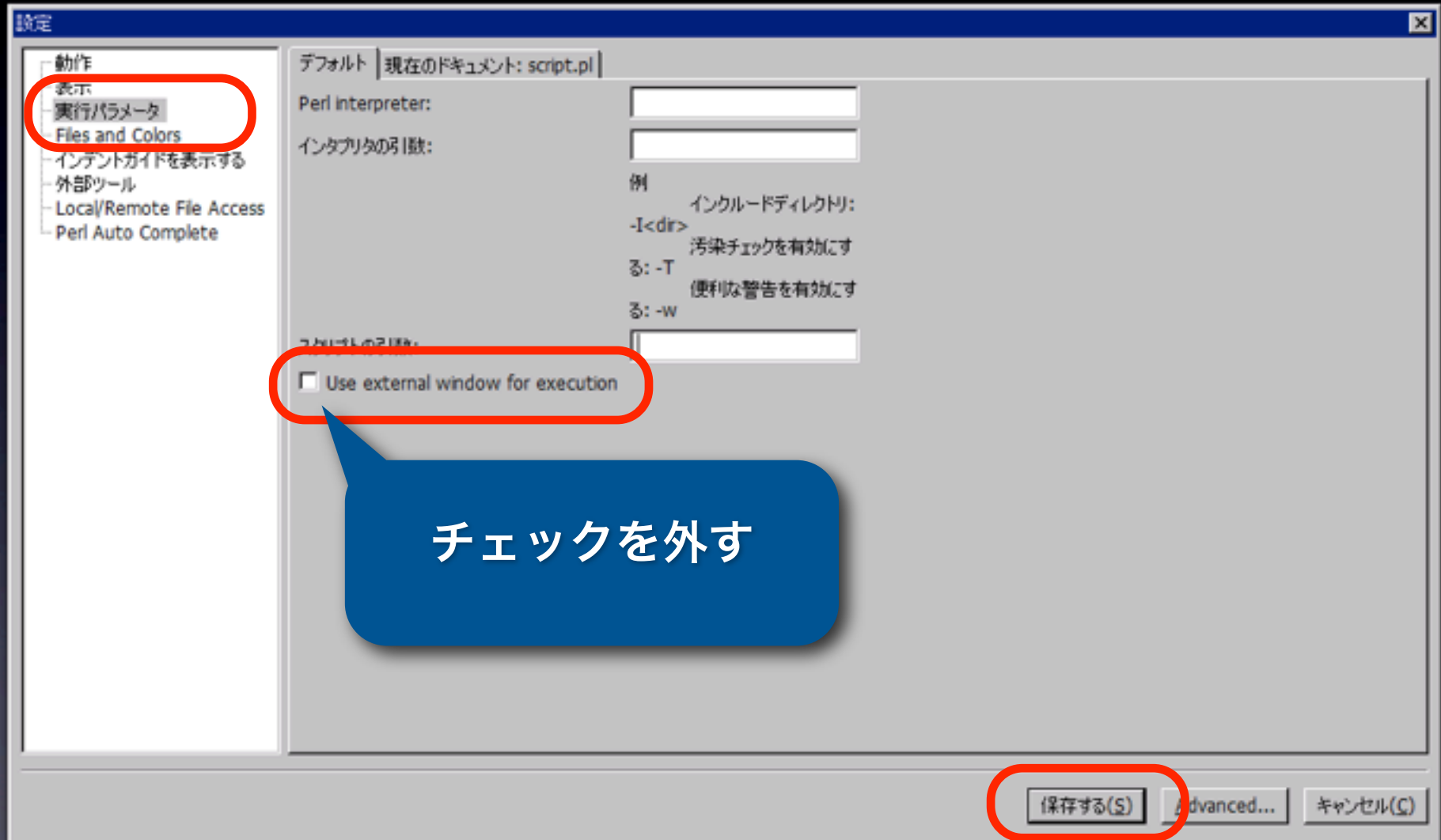
<http://padre.perlide.org>



まずは設定



まずは設定

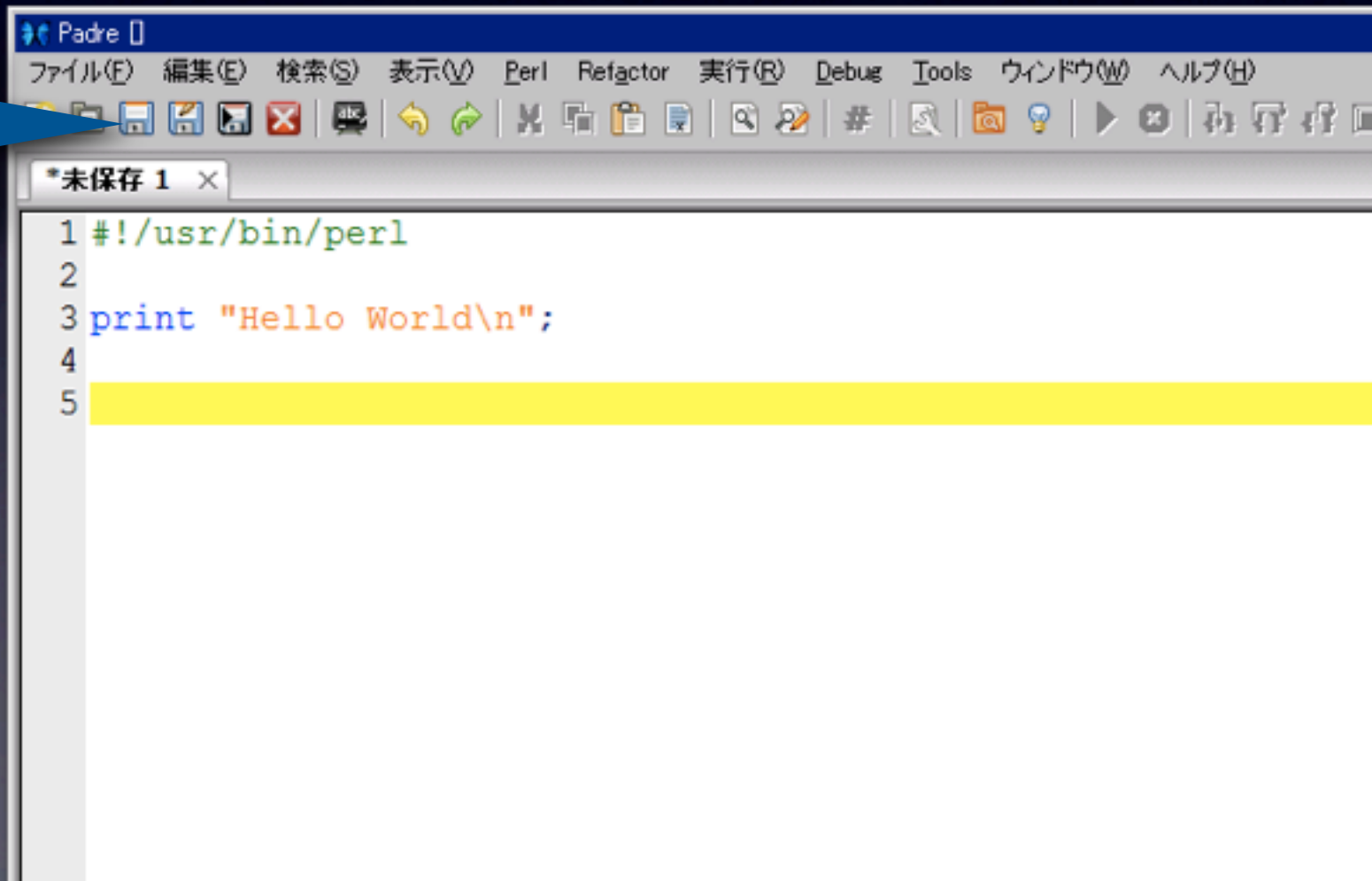


最初のプログラム

```
#!/usr/bin/perl  
print "Hello World\n";
```

¥

保存
(script.pl)

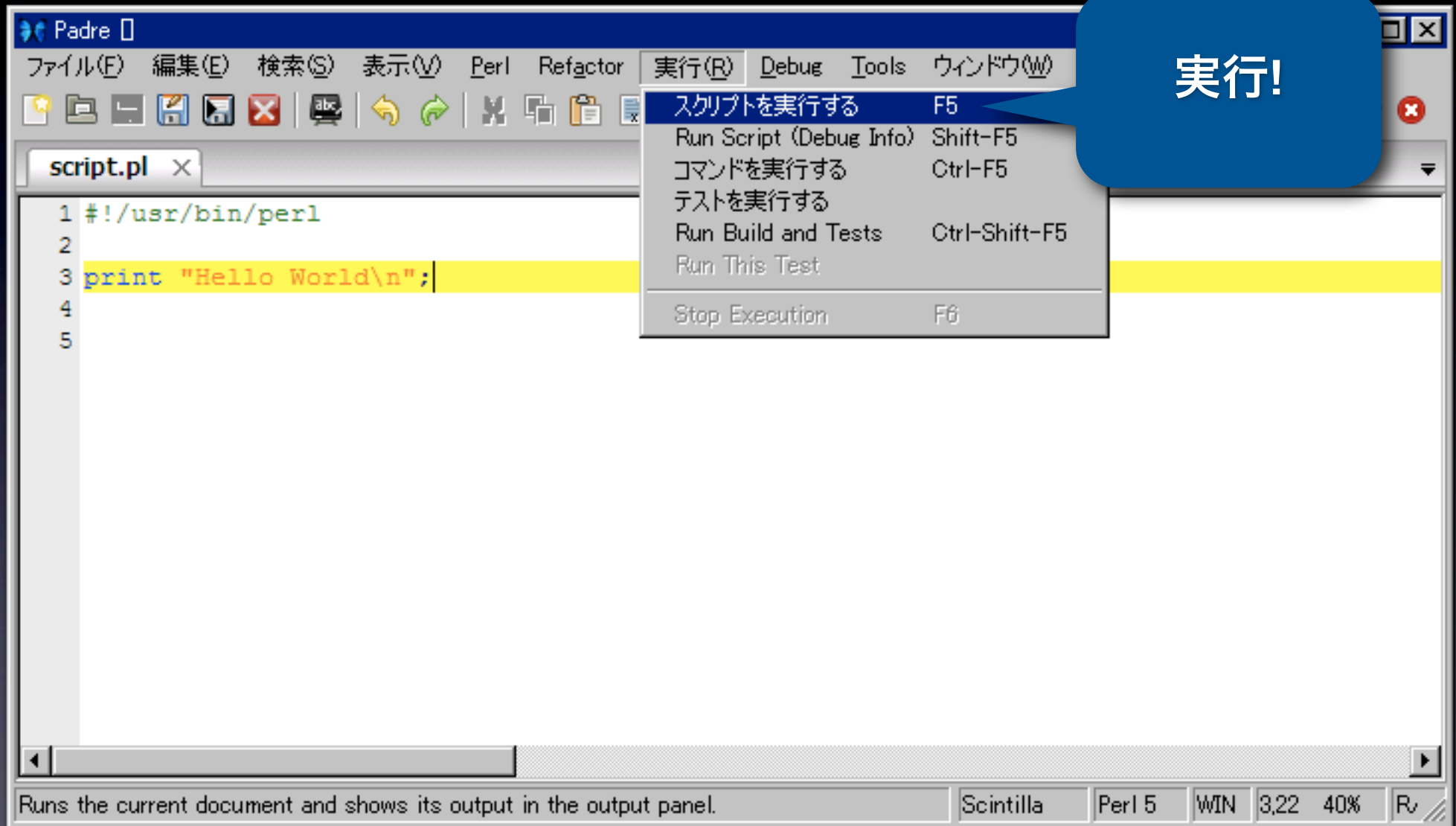


The screenshot shows the Padre Perl IDE interface. The menu bar includes options like 'ファイル(F)', '編集(E)', '検索(S)', '表示(V)', 'Perl', 'Refactor', '実行(R)', 'Debug', 'Tools', 'ウインドウ(W)', and 'ヘルプ(H)'. The toolbar contains various icons for file operations and execution. The main editor window, titled '*未保存 1 x', displays the Perl code from the previous block. The code is:

```
1 #!/usr/bin/perl  
2  
3 print "Hello World\n";  
4  
5
```

 A yellow highlight is visible under the line number 5.

```
#!/usr/bin/perl  
print "Hello World\n";
```



```
#!/usr/bin/perl  
print "Hello World\n";
```

MacでもLinuxでも動くように

print文は「表示」を行う

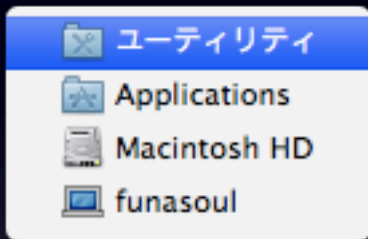
改行させるために \n か ¥n をつける

Hello World と表示される

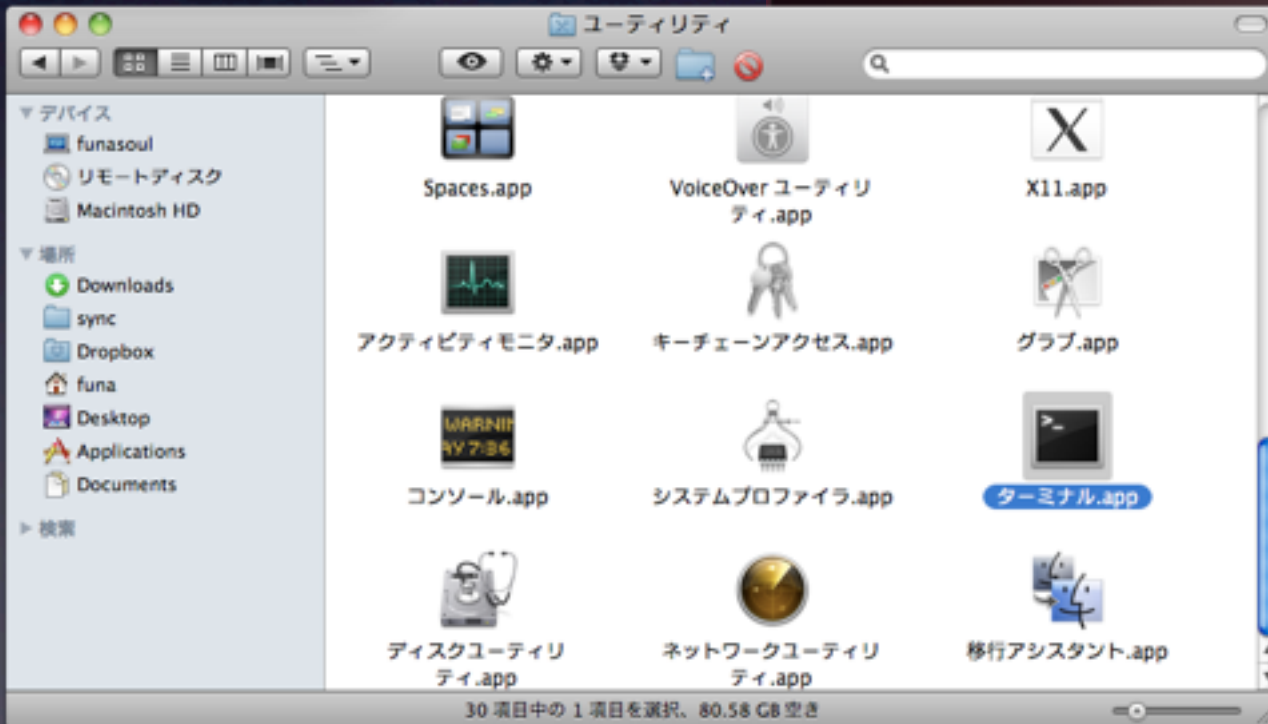


Mac, Linuxの人

Terminalを起動



```
funa@ubuntu: ~  
File Edit View Search Terminal Help  
ubuntu% ls  
CellDesigner4.2/           Documents/                runCellDesigner4.2@  
CellDesigner-4.2-linux-installer.bin*  Downloads/              Templates/  
CellDesigner42linux.tar.gz  examples.desktop       Uninstall_CellDesigner4.2@  
CellDesigner4.2-tar/       Music/                   Videos/  
CellDesignerSim/          Pictures/                workspace/  
Desktop/                  Public/  
ubuntu% |
```



Emacs, Xcode, nanoなどで作成

```
#!/usr/bin/perl  
print "Hello World\n";
```

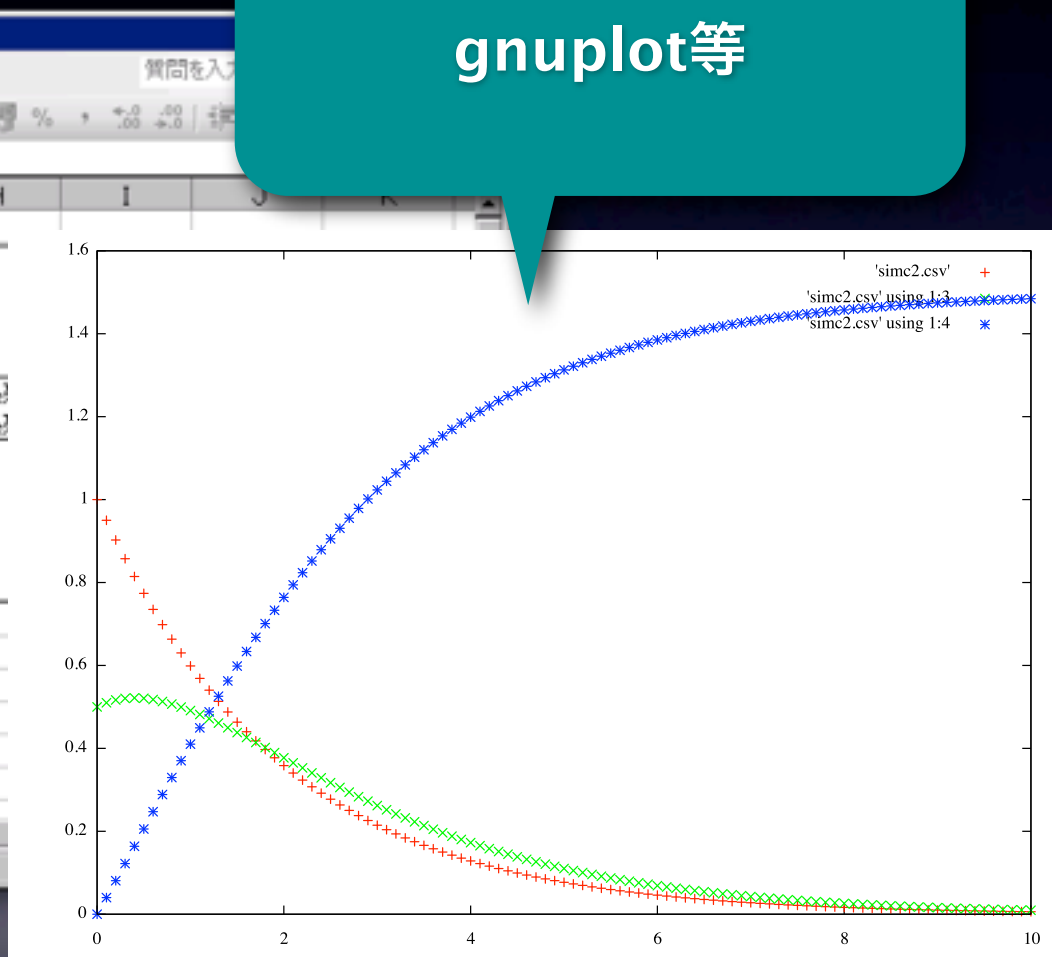
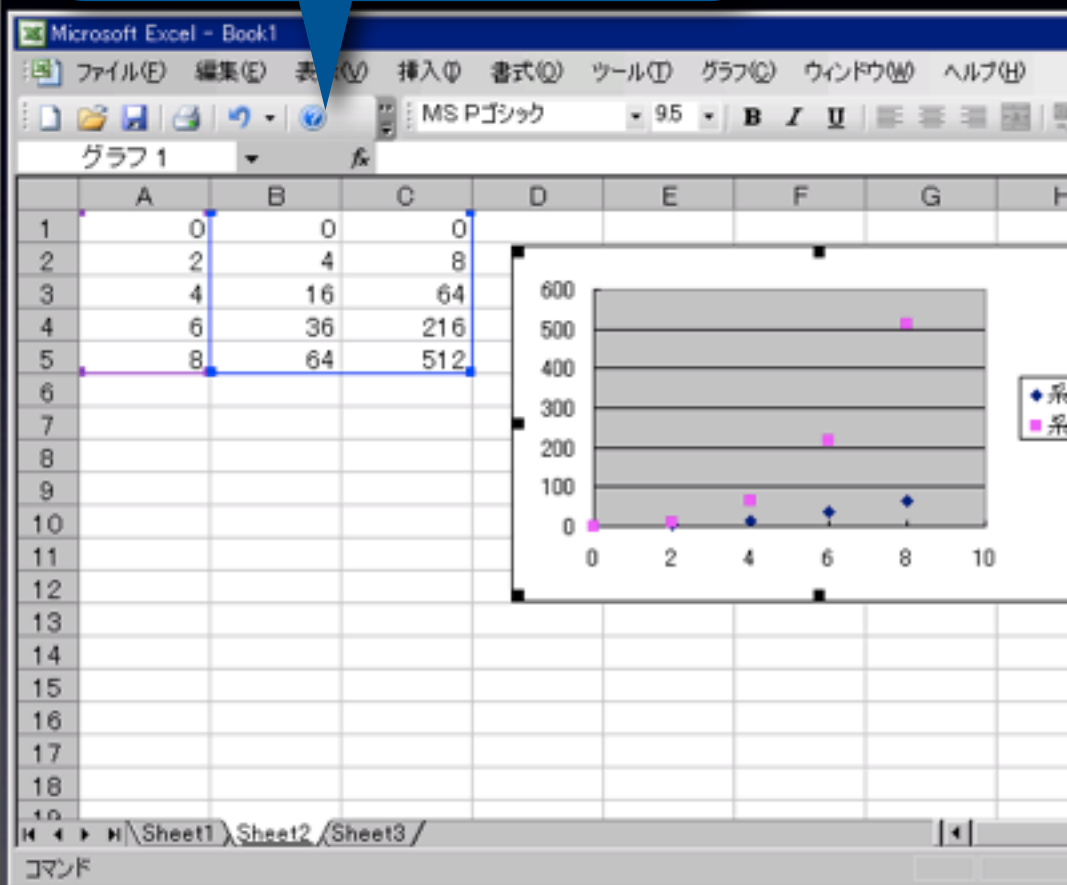
```
emacs script.pl  
(Perlのプログラムを書く)
```

```
perl script.pl  
Hello World
```

プロット

Excel等の表計算ソフト

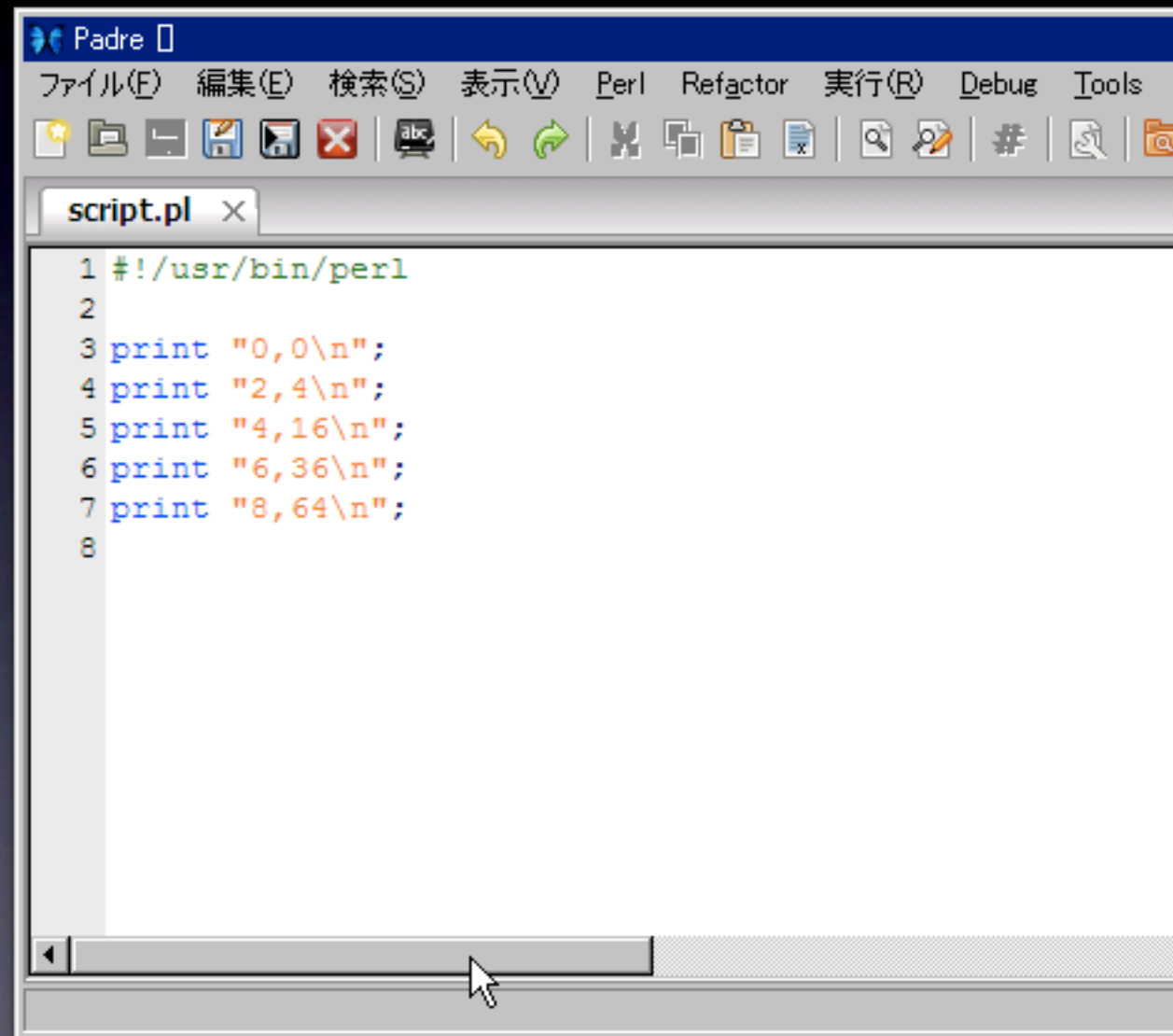
gnuplot等



プロット用データを用意

```
#!/usr/bin/perl

print "0,0\n";
print "2,4\n";
print "4,16\n";
print "6,36\n";
print "8,64\n";
```

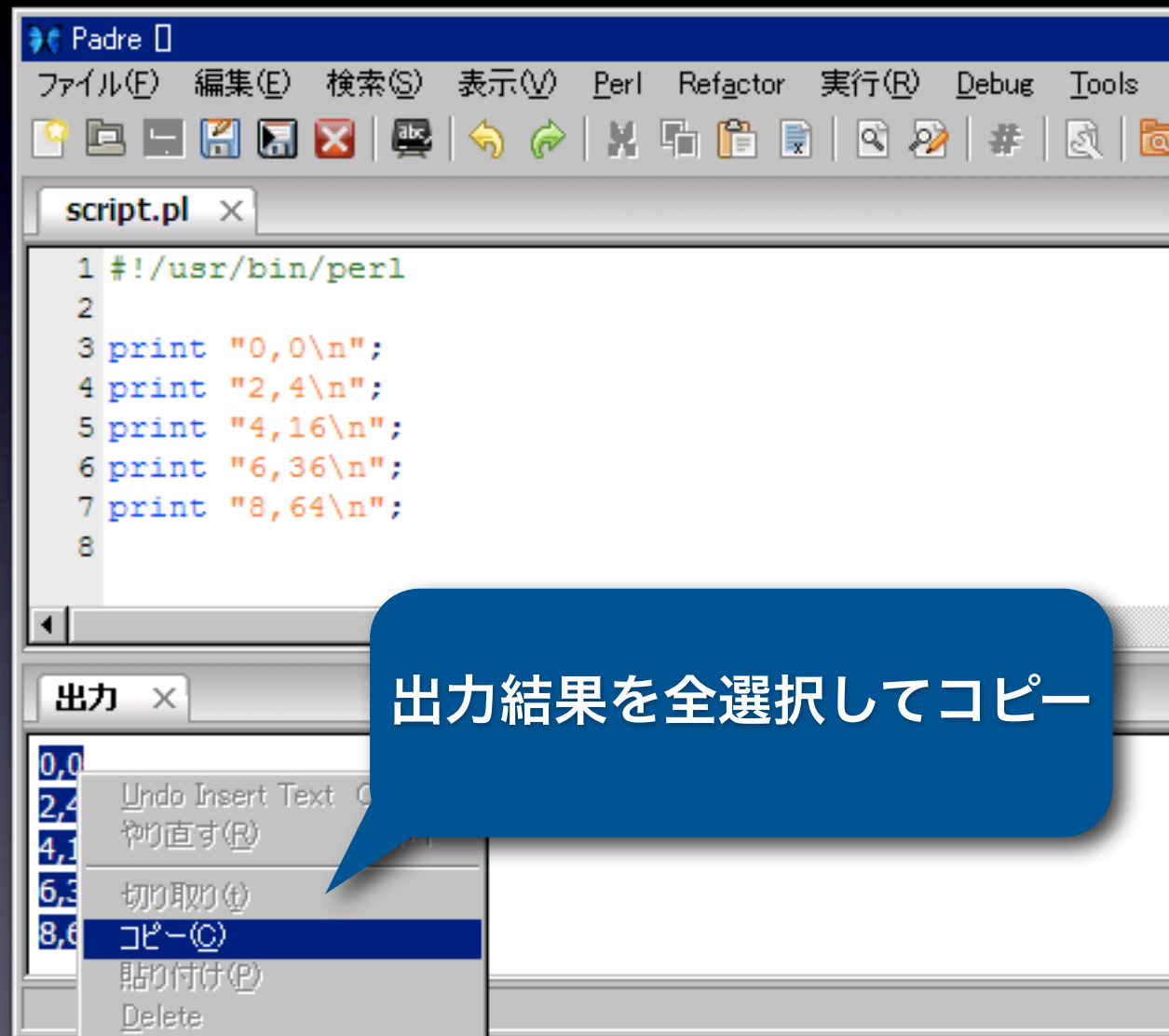


```
Padre
ファイル(F) 編集(E) 検索(S) 表示(V) Perl Refactor 実行(R) Debug Tools
script.pl x
1 #!/usr/bin/perl
2
3 print "0,0\n";
4 print "2,4\n";
5 print "4,16\n";
6 print "6,36\n";
7 print "8,64\n";
8
```

プロット用データを用意

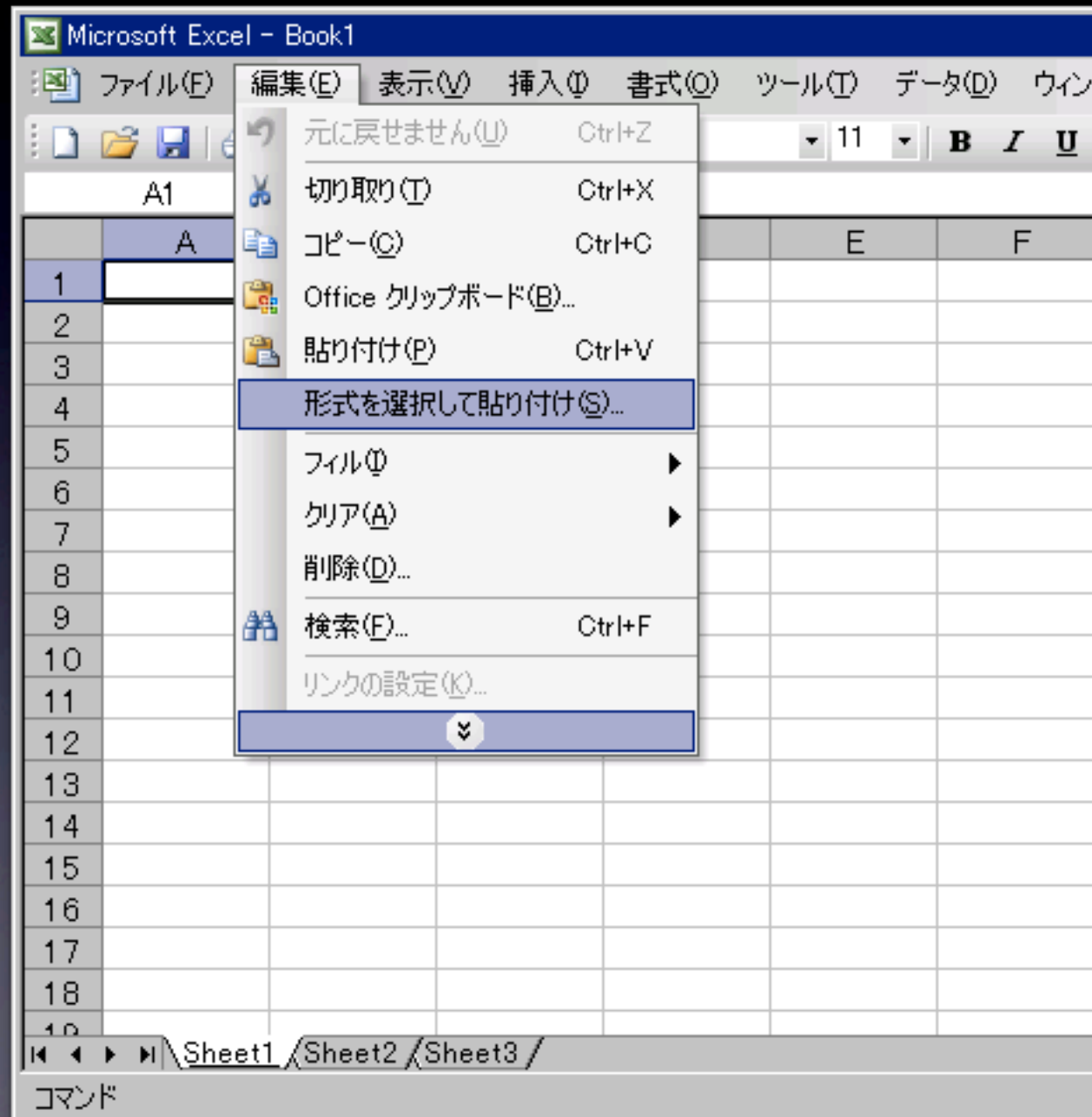
```
#!/usr/bin/perl

print "0,0\n";
print "2,4\n";
print "4,16\n";
print "6,36\n";
print "8,64\n";
```



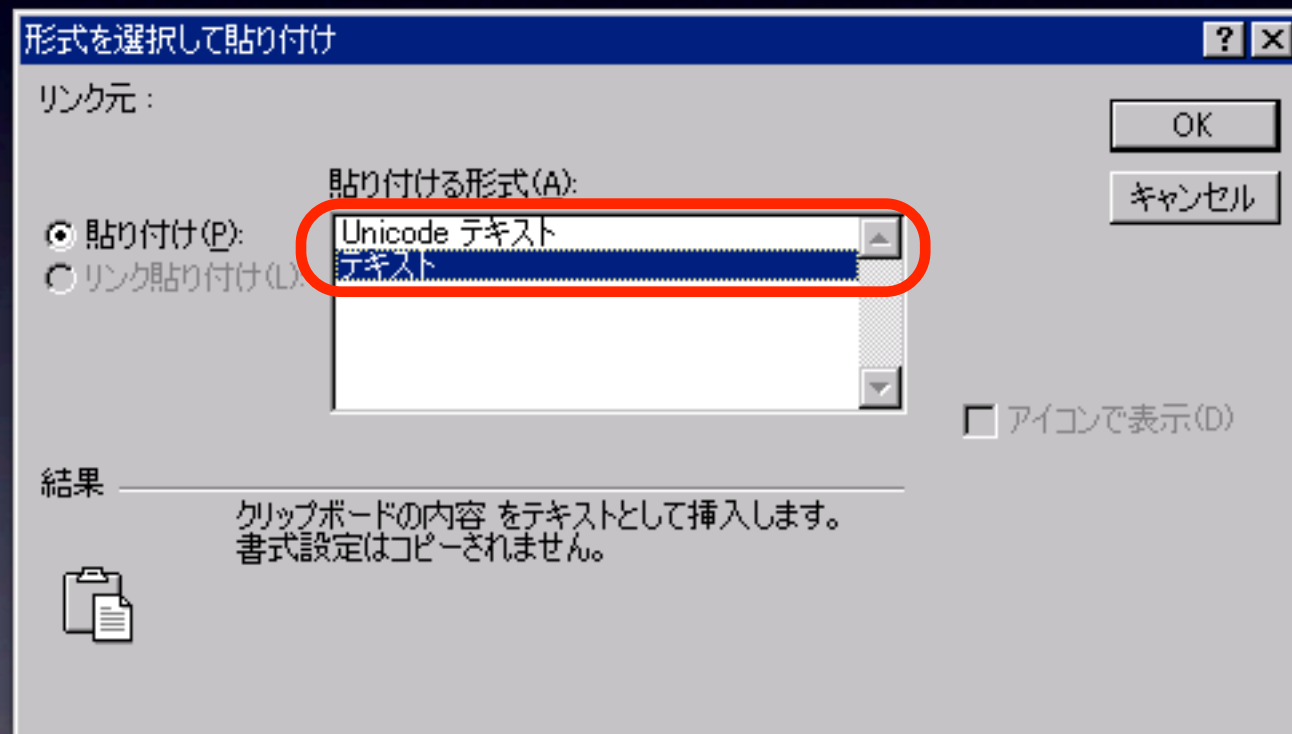
Excelに取り込む

```
#!/usr/bin/perl  
  
print "0,0\n";  
print "2,4\n";  
print "4,16\n";  
print "6,36\n";  
print "8,64\n";
```



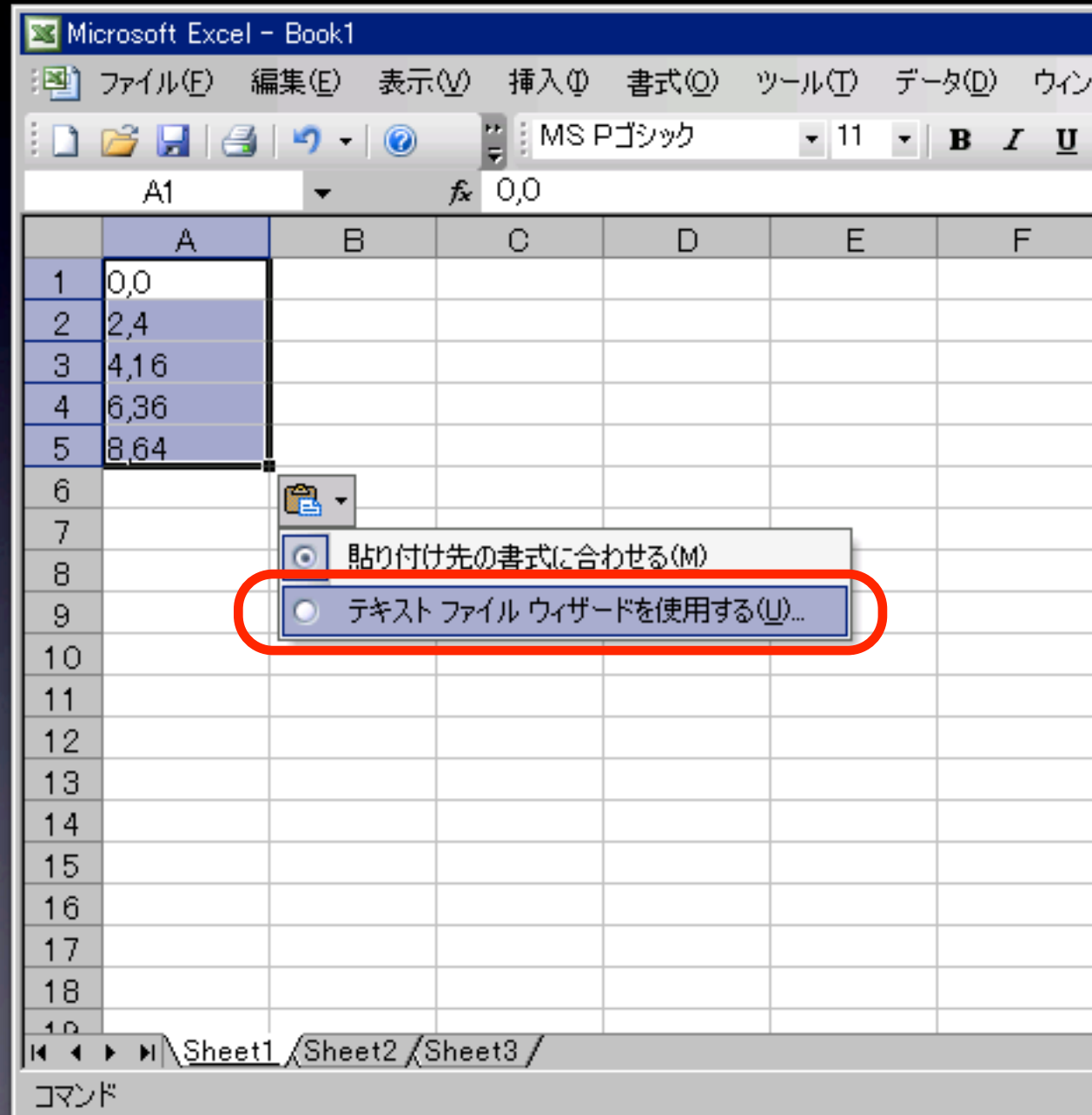
Excelに取り込む

```
#!/usr/bin/perl  
  
print "0,0\n";  
print "2,4\n";  
print "4,16\n";  
print "6,36\n";  
print "8,64\n";
```



Excelに取り込む

```
#!/usr/bin/perl  
  
print "0,0\n";  
print "2,4\n";  
print "4,16\n";  
print "6,36\n";  
print "8,64\n";
```



Excelに取り込む

```
#!/usr/bin/perl  
  
print "0,0\n";  
print "2,4\n";  
print "4,16\n";  
print "6,36\n";  
print "8,64\n";
```

テキストファイル ウィザード - 1 / 3

選択したデータは区切り文字で区切られています。
[次へ] をクリックするか、区切るデータの形式を指定してください。

元のデータの形式
データのファイル形式を選択してください:

- カンマやタブなどの区切り文字によってフィールドごとに区切られたデータ(D)
- スペースによって右または左に揃えられた固定長フィールドのデータ(W)

取り込み開始行(R): 元のファイル(O):

選択したデータのプレビュー:

1	0,0
2	2,4
3	4,16

キャンセル < 戻る(B) 次へ(N) > 完了(F)

Excelに取り込む

```
#!/usr/bin/perl  
  
print "0,0\n";  
print "2,4\n";  
print "4,16\n";  
print "6,36\n";  
print "8,64\n";
```

テキストファイル ウィザード - 2 / 3

フィールドの区切り文字を指定してください。[データのプレビュー] ボックスには区切り位置が表示されます。

区切り文字

タブ(T) セミコロン(M) カンマ(C) 文字列の引用符(Q): " ▾

スペース(S) その他(O):

連続した区切り文字は 1 文字として扱う(R)

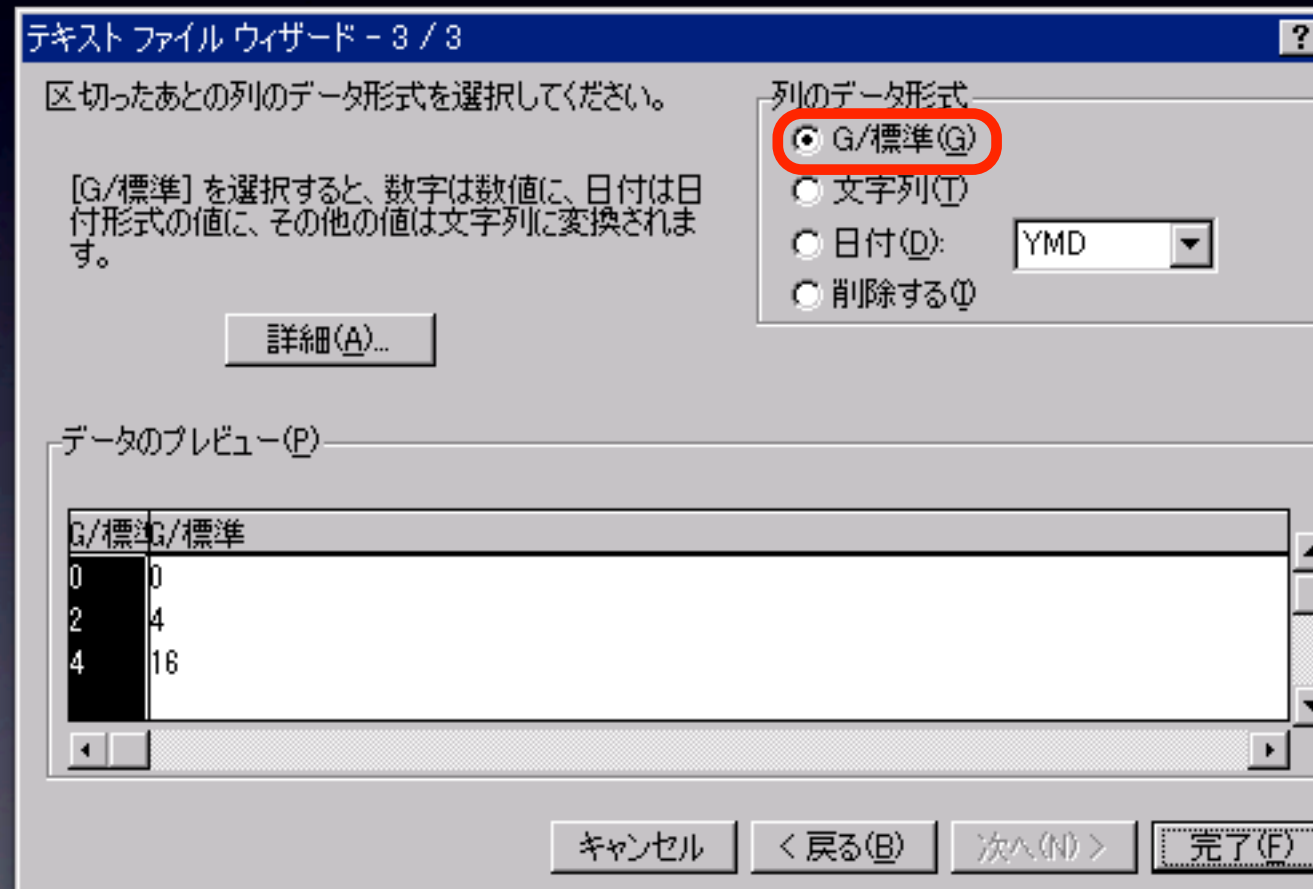
データのプレビュー(P)

0	0
2	4
4	16

キャンセル < 戻る(B) 次へ(N) > 完了(F)

Excelに取り込む

```
#!/usr/bin/perl  
  
print "0,0\n";  
print "2,4\n";  
print "4,16\n";  
print "6,36\n";  
print "8,64\n";
```



Excelに取り込む

```
#!/usr/bin/perl  
  
print "0,0\n";  
print "2,4\n";  
print "4,16\n";  
print "6,36\n";  
print "8,64\n";
```

カンマ区切り
(Comma Separated Values: CSV)

Microsoft Excel - Book1

ファイル(F) 編集(E) 表示(V) 挿入(I) 書式(O) ツール(T) データ(D) ウィンドウ(W) ヘルプ(H)

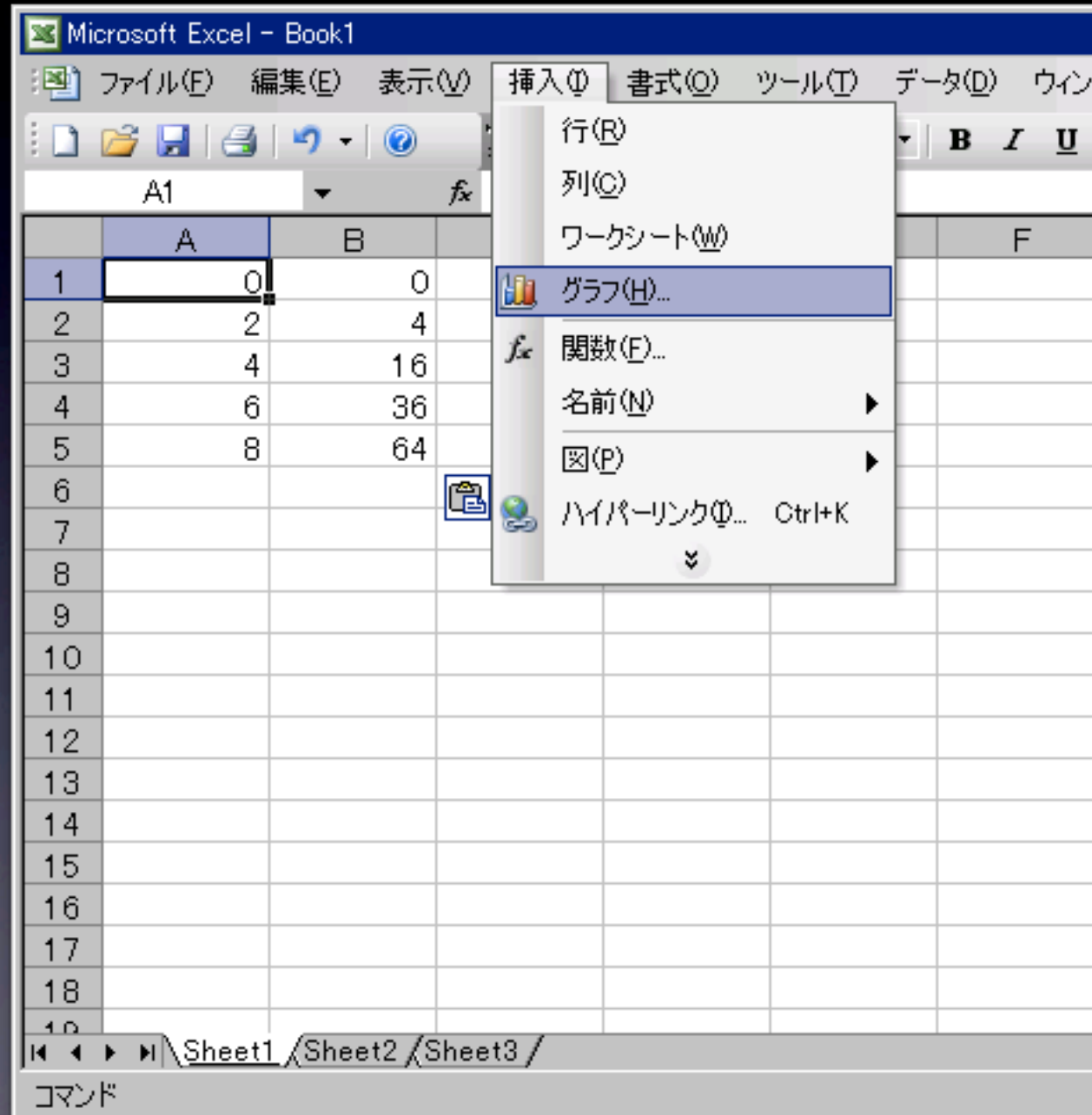
MS Pゴシック 11 B I U

	A	B	C	D	E	F
1	0	0	フォント			
2	2	4				
3	4	16				
4	6	36				
5	8	64				
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						

Sheet1 / Sheet2 / Sheet3 /

Excelでプロット

```
#!/usr/bin/perl  
  
print "0,0\n";  
print "2,4\n";  
print "4,16\n";  
print "6,36\n";  
print "8,64\n";
```



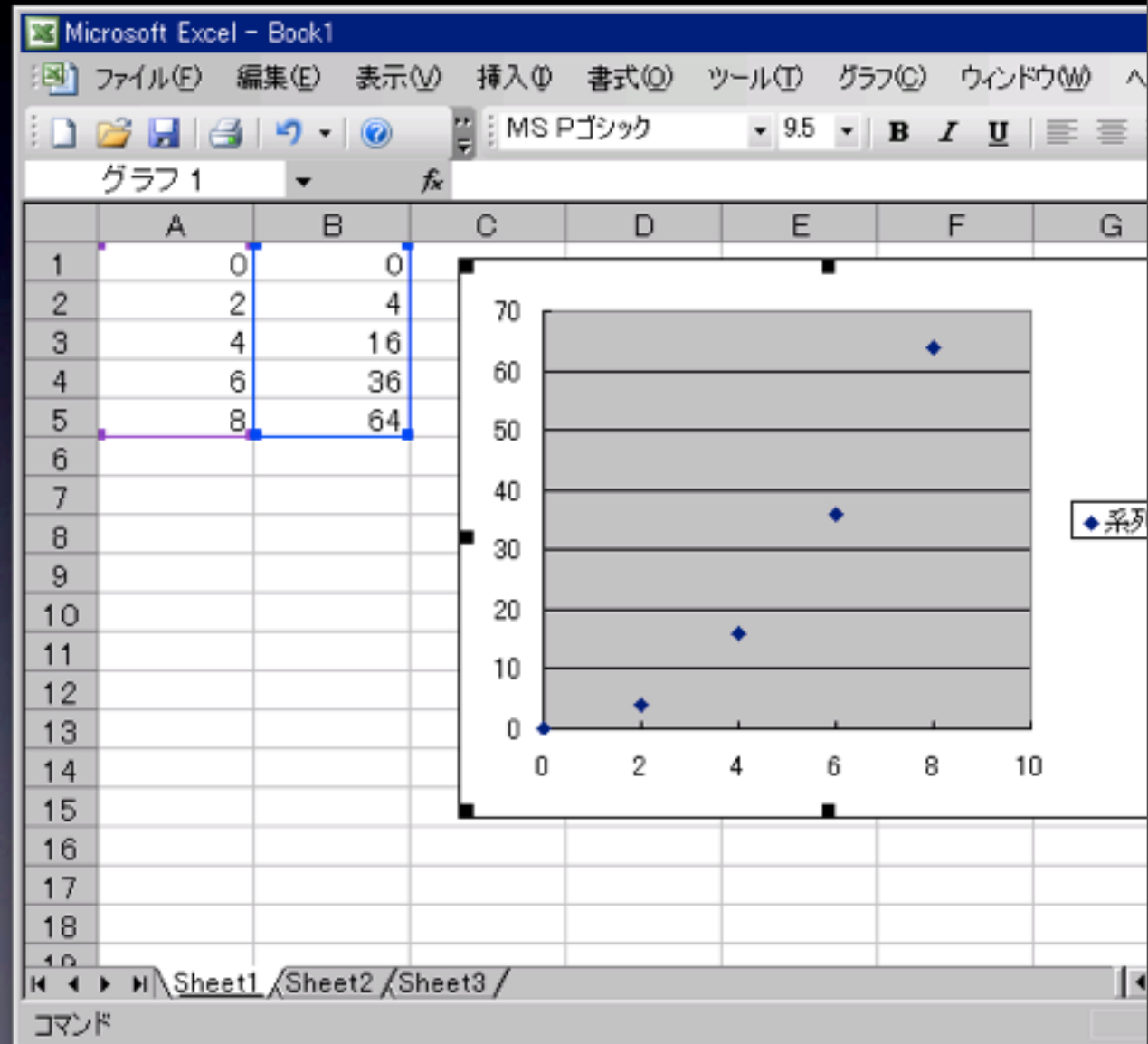
Excelでプロット

```
#!/usr/bin/perl  
  
print "0,0\n";  
print "2,4\n";  
print "4,16\n";  
print "6,36\n";  
print "8,64\n";
```



Excelでプロット

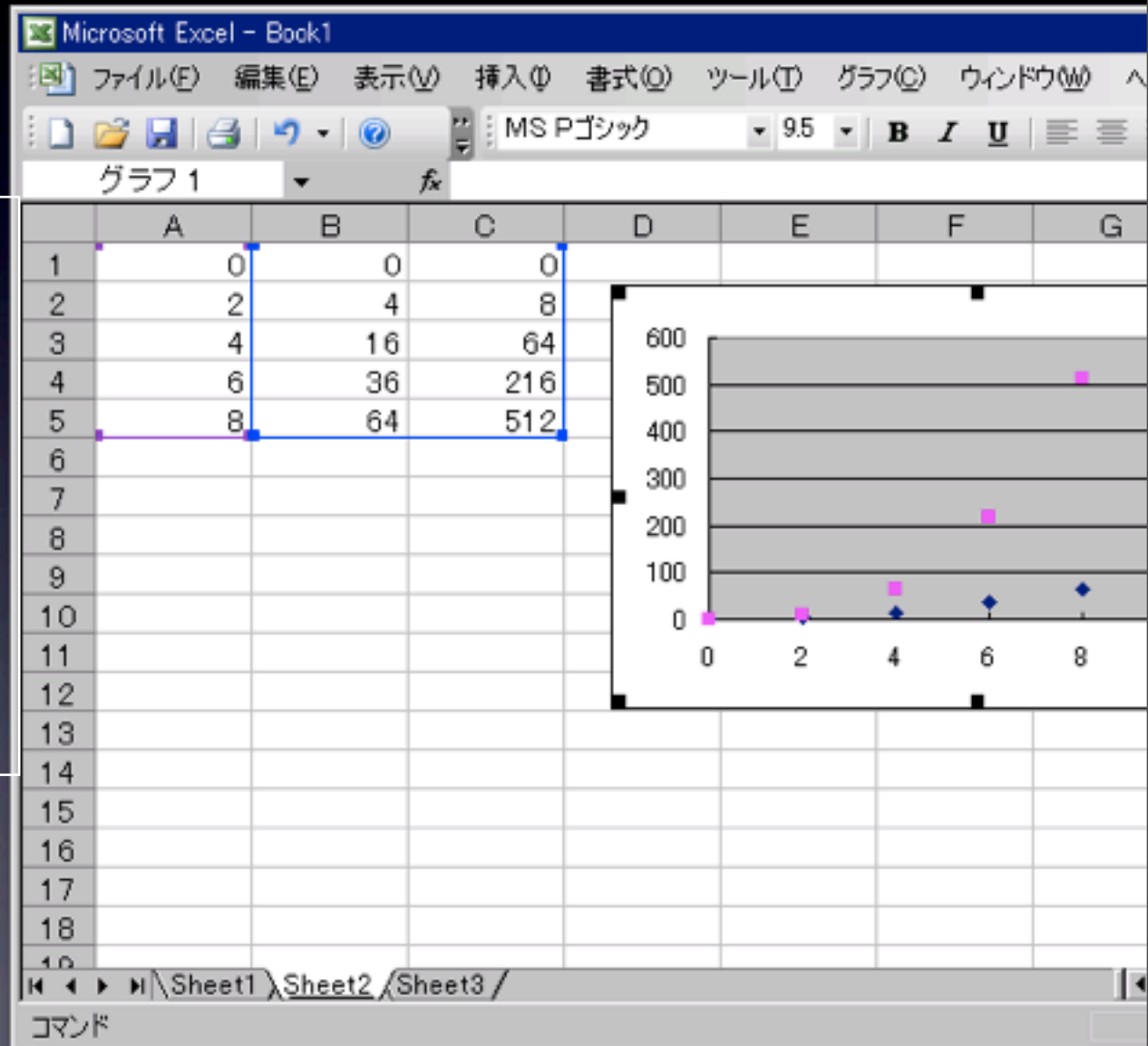
```
#!/usr/bin/perl  
  
print "0,0\n";  
print "2,4\n";  
print "4,16\n";  
print "6,36\n";  
print "8,64\n";
```



Excelでプロット

```
#!/usr/bin/perl

print "0,0,0\n";
print "2,4,8\n";
print "4,16,64\n";
print "6,36,216\n";
print "8,64,512\n";
```



gnuplotでプロット

```
perl script.pl
```

```
0,0
```

```
2,4
```

```
4,16
```

```
6,36
```

```
8,64
```

とりあえず実行して確認

ファイル(data.csv)に保存

```
#!/usr/bin/perl
```

```
print "0,0\n";
```

```
print "2,4\n";
```

```
print "4,16\n";
```

```
print "6,36\n";
```

```
print "8,64\n";
```

```
perl script.pl >! data.csv
```

```
gnuplot
```

```
gnuplot> set datafile separator ","
```

```
gnuplot> plot 'data.csv'
```

カンマ区切りだよ

gnuplotでプロット

```
perl script.pl >! data.csv
gnuplot
gnuplot> set datafile separator ","
gnuplot> plot 'data.csv'
gnuplot> replot 'data.csv' using 1:3
```

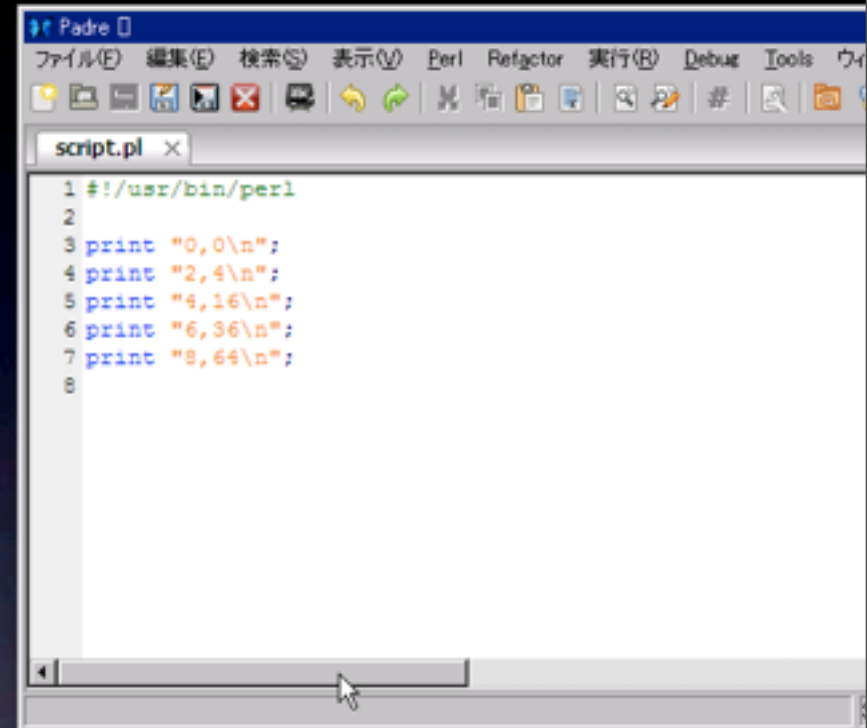
```
#!/usr/bin/perl

print "0,0,0\n";
print "2,4,8\n";
print "4,16,64\n";
print "6,36,216\n";
print "8,64,512\n";
```

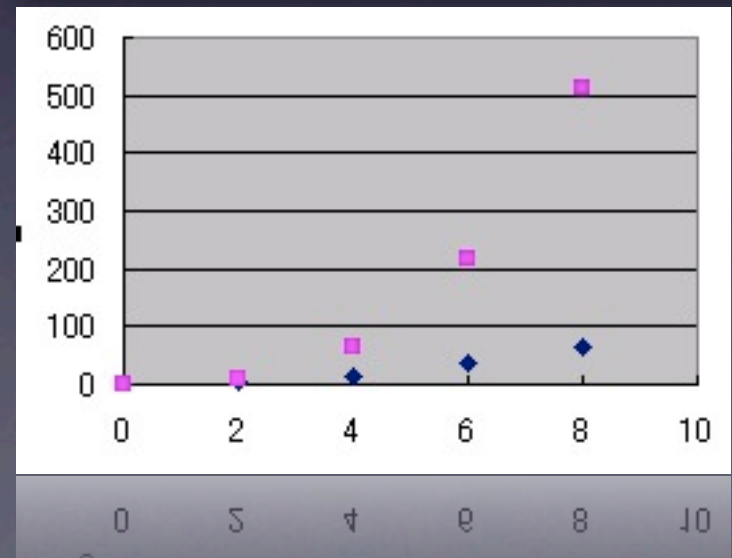
using 1:4
using 1:5
...

ここまでのまとめ

- Perlでプログラムを書く
 - Padre, Emacs等
 - print文で表示
 - CSV(カンマ区切り)で表示
- プロット
 - CSVを読み込む
 - Excel, gnuplot



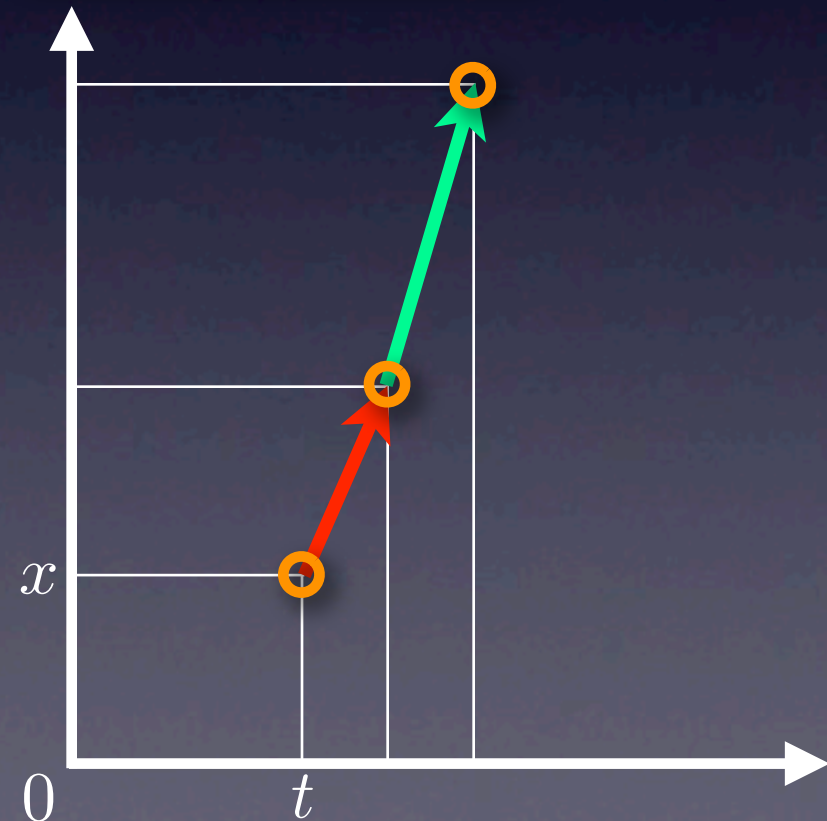
```
1 #!/usr/bin/perl
2
3 print "0,0\n";
4 print "2,4\n";
5 print "4,16\n";
6 print "6,36\n";
7 print "8,64\n";
8
```



足りないもの

- Perlでプログラムを書く
 - Padre, Emacs等
 - print文で表示
 - CSV(カンマ区切り)で表示
- プロット
 - CSVを読み込む
 - Excel, gnuplot

数値積分



ODE Simulator

- $\frac{dx}{dt} = 2x$ を解くシミュレータ $t = 0, x = 1.0$

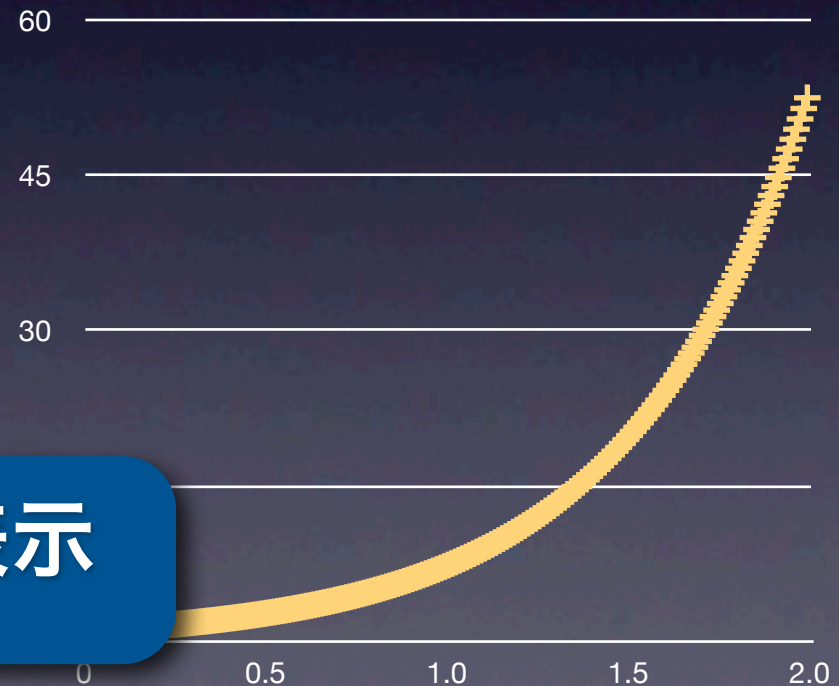
```
#!/usr/bin/perl
```

```
$dt = 0.01;  
$t = 0.0;  
$x = 1.0;
```

```
for ($i = 0; $i <= 200; $i++) {  
  print "$t,$x\n";  
  $dx = 2 * $x * $dt;  
  $x = $x + $dx;  
  $t = $t + $dt;  
}
```

CSVで表示

シミュレーション結果



ODE Simulator

- $\frac{dx}{dt} = 2x$ を解くシミュレータ $t = 0, x = 1.0$

```
#!/usr/bin/perl
```

Δt の値

```
$dt = 0.01;
```

```
$t = 0.0;
```

```
$x = 1.0;
```

x, tの初期値

```
for ($i = 0; $i <= 200; $i++) {
```

201回繰り返す

```
  print "$t,$x\n";
```

```
  $dx = 2 * $x * $dt;
```

```
  $x = $x + $dx;
```

```
  $t = $t + $dt;
```

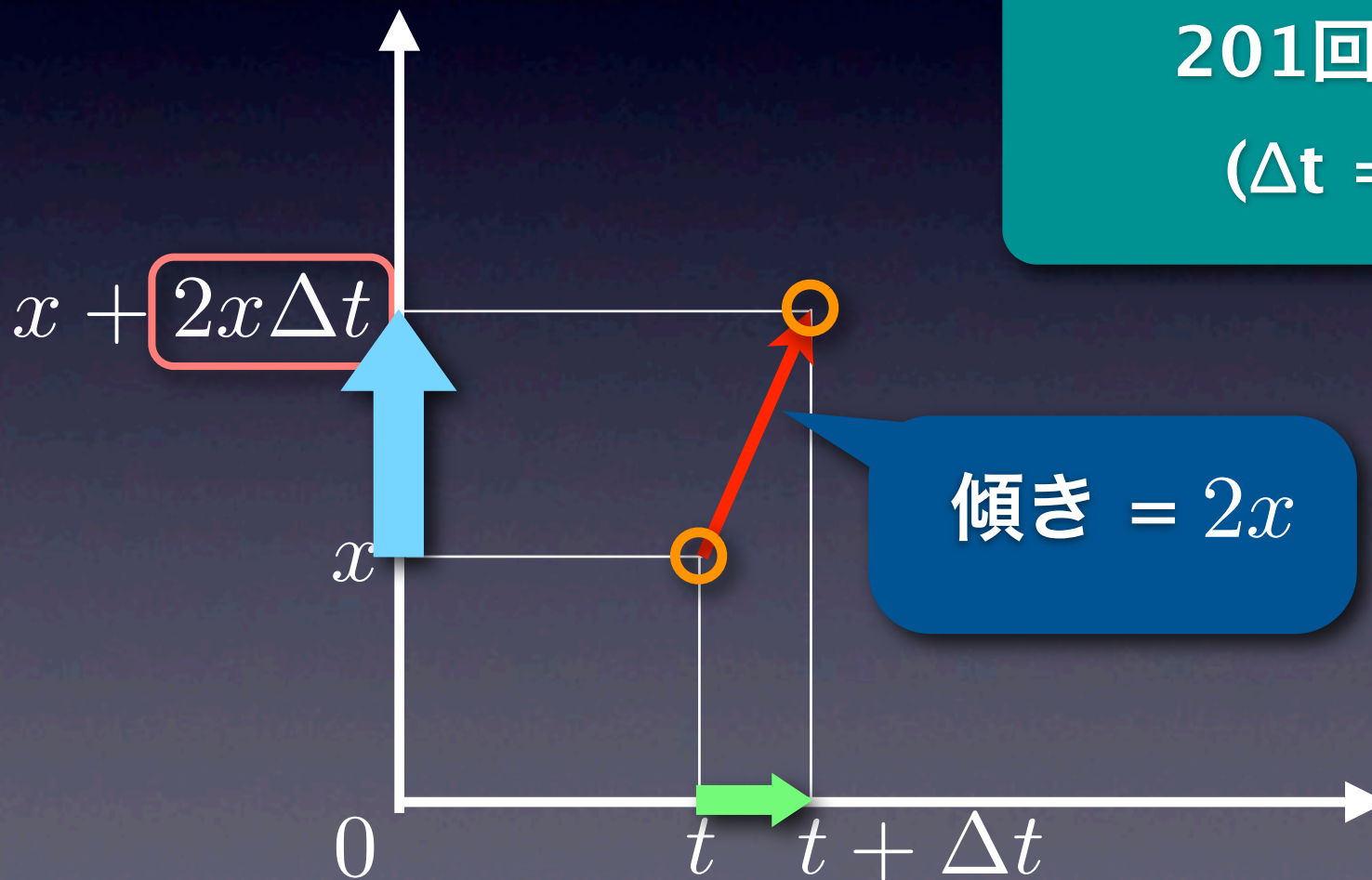
繰り返しの内容

```
}
```

```
for ($i = 0; $i <= 200; $i++) {  
    print "$t,$x\n";  
    $dx = 2 * $x * $dt;  
    $x = $x + $dx;  
    $t = $t + $dt;  
}
```

$$\frac{dx}{dt} = 2x$$

t=0 から t=2.0まで
201回繰り返す
($\Delta t = 0.01$)



ポイント

- $\frac{dx}{dt} = 2x$ を解くシミュレータ $t = 0, x = 1.0$

```
#!/usr/bin/perl
```

Δt の値

```
$dt = 0.01;
```

```
$t = 0.0;
```

```
$x = 1.0;
```

x, tの初期値

```
for ($i = 0; $i <= 200; $i++) {
```

```
  print "$t, $x\n";
```

```
  $dx = 2 * $x * $dt;
```

```
  $x = $x + $dx;
```

```
  $t = $t + $dt;
```

```
}
```

$dx/dt =$ の右辺を書く

xとtをちょっと増やす(減らす)

シミュレーション時間
と Δt によって繰り返し
回数を決める

```
1 #!/usr/bin/perl
2
3 $dt = 0.01;
4 $t = 0.0;
5 $x = 1.0;
6
7 for ($i = 0; $i <= 200; $i++) {
8     print "$t,$x\n";
9     $dx = 2 * $x * $dt;
10    $x = $x + $dx;
11    $t = $t + $dt
12 }
13
```

Scintilla Perl 5 WIN 13.0 92%

Padre

ファイル(F) 編集(E) 検索(S) 表示(V) Perl Refactor 実行(R) Debug Tools ウィンドウ(W) ヘルプ(H)

script.pl simulate.pl

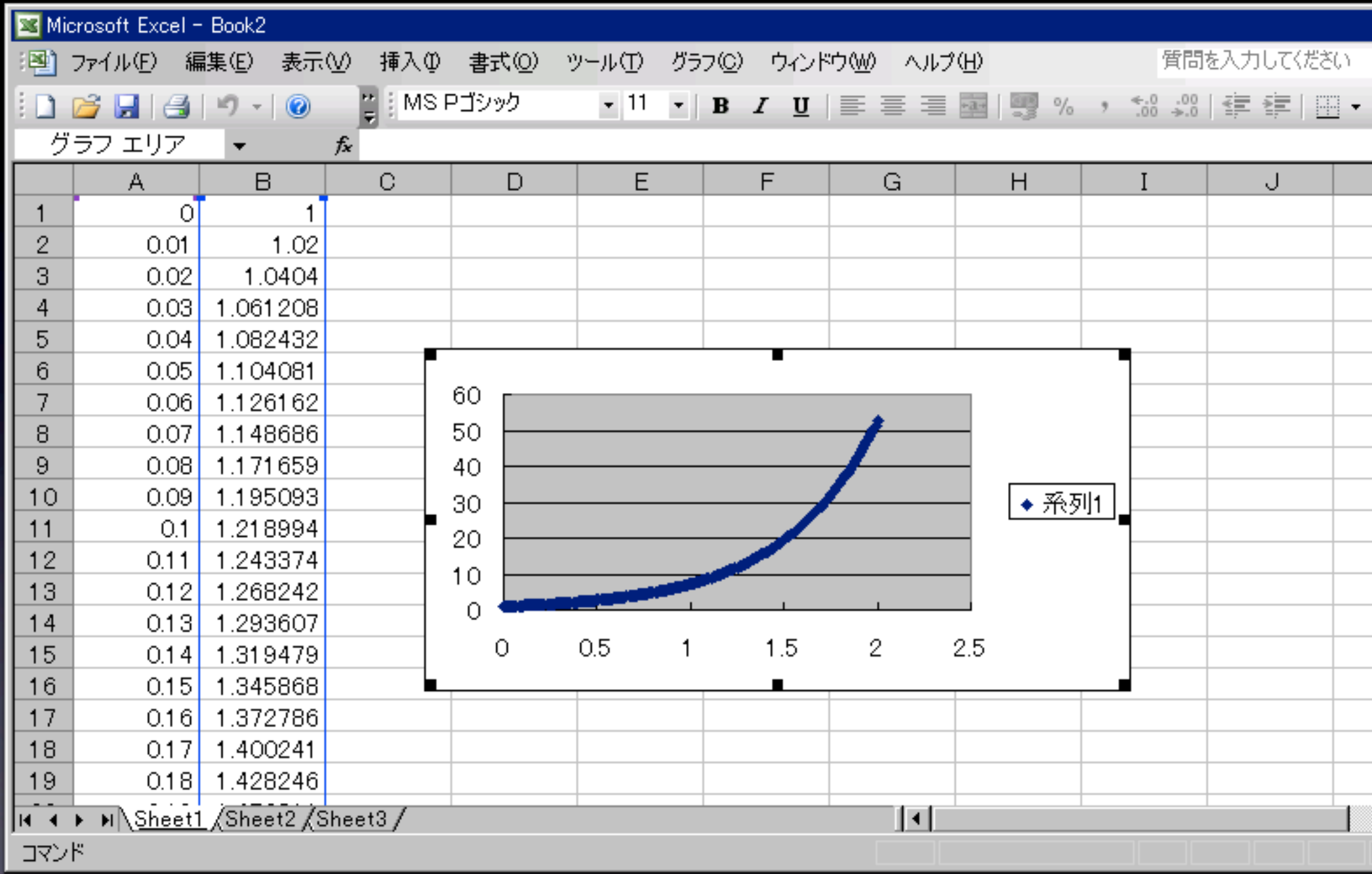
```
1 #!/usr/bin/perl
2
3 $dt = 0.01;
4 $t = 0.0;
5 $x = 1.0;
6
7 for ($i = 0; $i <= 200; $i++) {
8     print "$t,$x\n";
9     $dx = 2 * $x * $dt;
10    $x = $x + $dx;
11    $t = $t + $dt
12 }
13
```

出力

```
1.97,49.457691026366
1.98,50.446844846894
1.99,51.455781743832
2.52.4848973787087
```

Undo Insert Text Ctrl+Z
やり直す(R) Ctrl+Y
切り取り(C)
コピー(C)
貼り付け(P)

C:\Documents and Settings\... simulate.pl Scintilla Perl 5 WIN 13.0 92%



Mac, Linuxの人

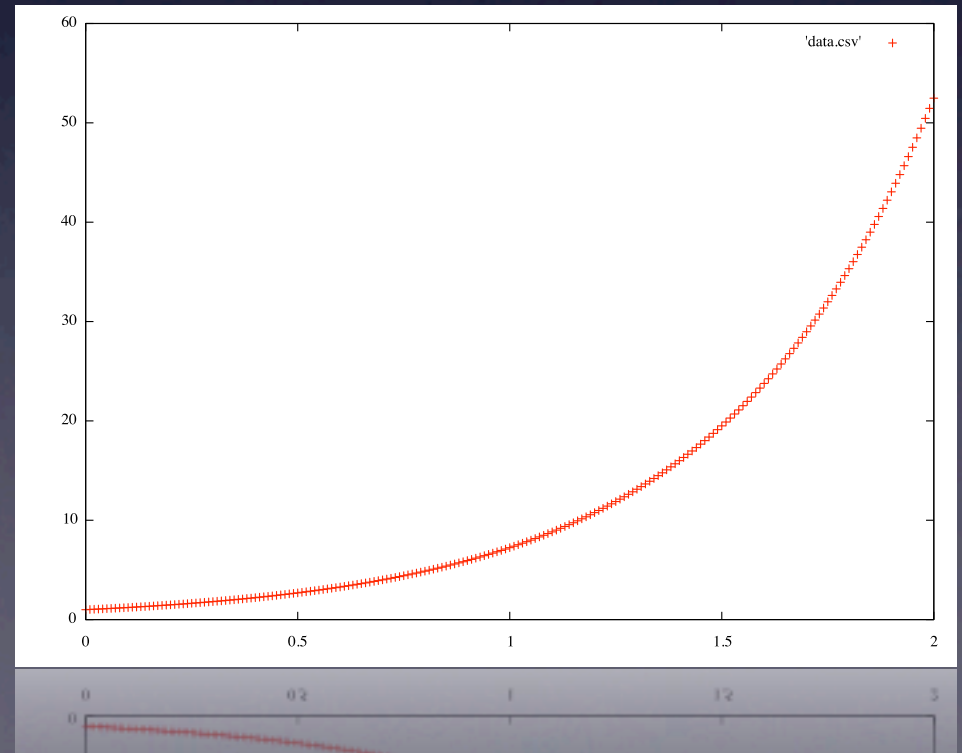
```
emacs simulate.pl
```

```
perl simulate.pl >! data.csv
```

```
gnuplot
```

```
gnuplot> set datafile separator ","
```

```
gnuplot> plot 'data.csv'
```



C言語版

- $\frac{dx}{dt} = 2x$ を解くシミュレータ $t = 0, x = 1.0$

```
#include<stdio.h>

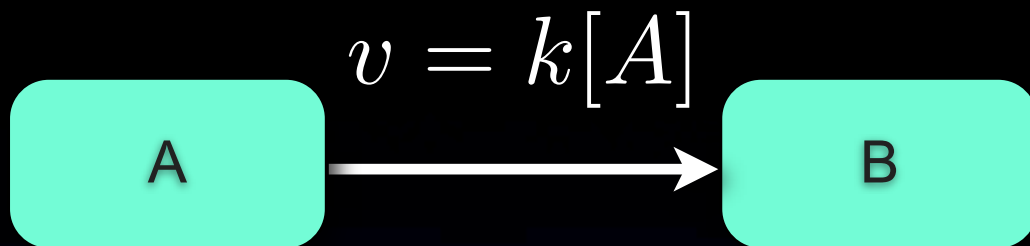
int main(void) {
    int i;
    double dx;
    double dt = 0.01;
    double t = 0.0;
    double x = 1.0;

    for (i = 0; i <= 200; i++) {
        printf("%lf,%lf\n", t, x);
        dx = 2.0 * x * dt;
        x = x + dx;
        t = t + dt;
    }
    return 0;
}
```

```
emacs simulate.c
gcc -Wall simulate.c
./a.out >! data.csv
(プロットはPerl版と同様)
```

Cコンパイラは gcc,
MSVC, LLVM Clang等

2変数モデル



```
#!/usr/bin/perl
```

```
$dt = 0.01;
```

```
$t = 0.0;
```

```
$A = 1.0;
```

```
$B = 0.0;
```

```
$k = 1.5;
```

```
for ($i = 0; $i <= 200; $i++) {
```

```
  print "$t, $A, $B\n";
```

```
  $dA = - $k * $A * $dt;
```

```
  $dB = $k * $A * $dt;
```

```
  $A = $A + $dA;
```

```
  $B = $B + $dB;
```

```
  $t = $t + $dt
```

```
}
```

$$\frac{d[A]}{dt} = -k[A]$$

$$\frac{d[B]}{dt} = k[A]$$

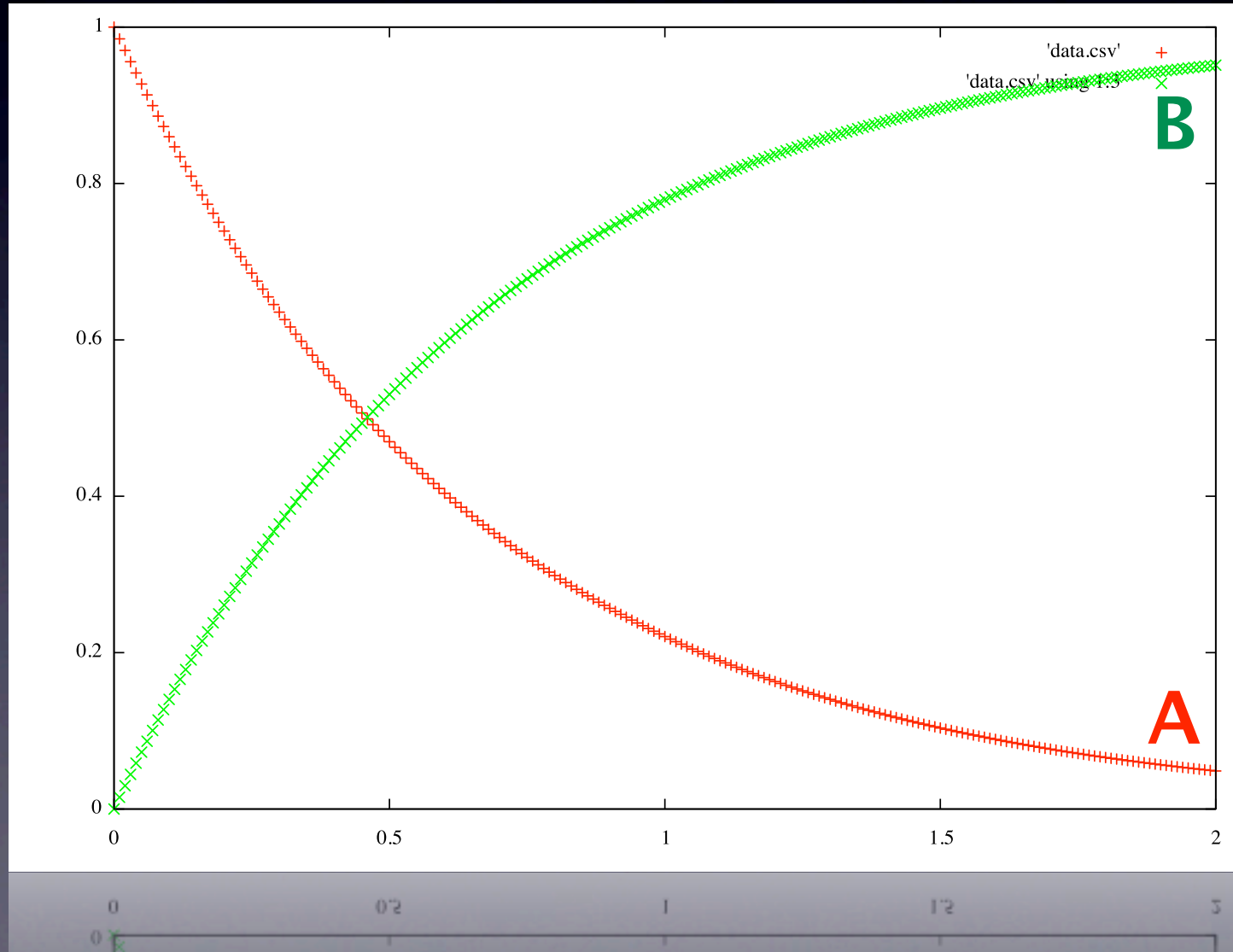
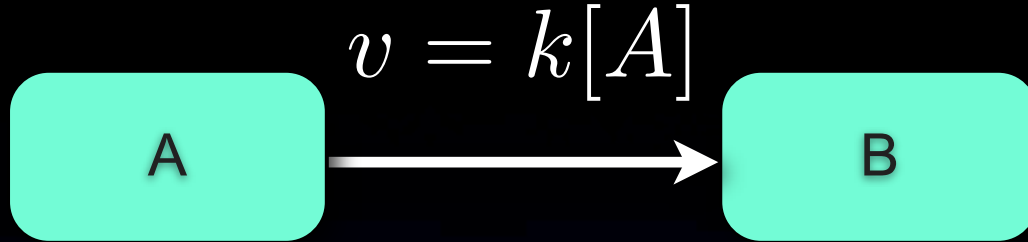
$$[A]_0 = 1.0$$

$$[B]_0 = 0.0$$

$$k = 1.5$$

A, B, t をちょっと増やす(減らす)

2変数モデル



C言語版

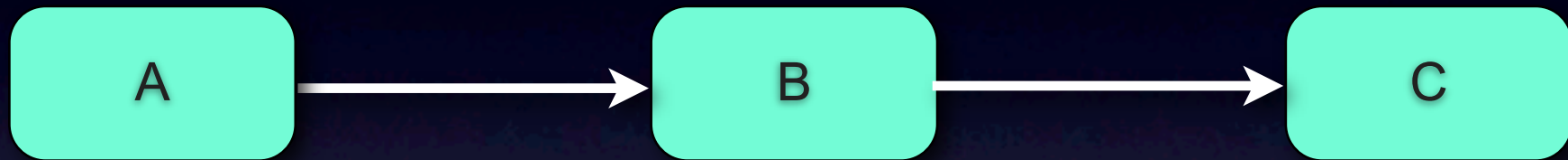
```
#include<stdio.h>
```

```
int main(void) {  
    int i;  
    double dA, dB;  
    double dt = 0.01;  
    double t = 0.0;  
    double A = 1.0;  
    double B = 0.0;  
    double k = 1.5;  
  
    for (i = 0; i <= 200; i++) {  
        printf("%lf,%lf,%lf\n", t, A, B);  
        dA = - k * A * dt;  
        dB =  k * A * dt;  
        A = A + dA;  
        B = B + dB;  
        t = t + dt;  
    }  
    return 0;  
}
```

```
emacs simulate.c  
gcc -Wall simulate.c  
./a.out >! data.csv  
(プロットはPerl版と同様)
```

3変数モデル

$$v_1 = k_1[A] \quad v_2 = k_2[B]$$



$$\frac{d[A]}{dt} = -v_1 \quad \frac{d[B]}{dt} = v_1 - v_2 \quad \frac{d[C]}{dt} = v_2$$

$$[A]_0 = 1.0, \quad [B]_0 = 0.5, \quad [C]_0 = 0.0$$

$$k_1 = 0.5, \quad k_2 = 0.8$$

```
#!/usr/bin/perl
```

Perl版

```
$dt = 0.1;  
$t  = 0.0;  
$A  = 1.0;  
$B  = 0.5;  
$C  = 0.0;  
$k1 = 0.5;  
$k2 = 0.8;
```

```
for ($i = 0; $i <= 100; $i++) {  
    print "$t,$A,$B,$C\n";  
    $v1 = $k1 * $A;  
    $v2 = $k2 * $B;  
    $dA = - $v1 * $dt;  
    $dB =  $v1 * $dt - $v2 * $dt;  
    $dC =  $v2 * $dt;  
    $A  = $A + $dA;  
    $B  = $B + $dB;  
    $C  = $C + $dC;  
    $t  = $t + $dt  
}
```

$$v_1 = k_1[A]$$

$$v_2 = k_2[B]$$

$$\frac{d[A]}{dt} = -v_1$$

$$\frac{d[B]}{dt} = v_1 - v_2$$

$$\frac{d[C]}{dt} = v_2$$

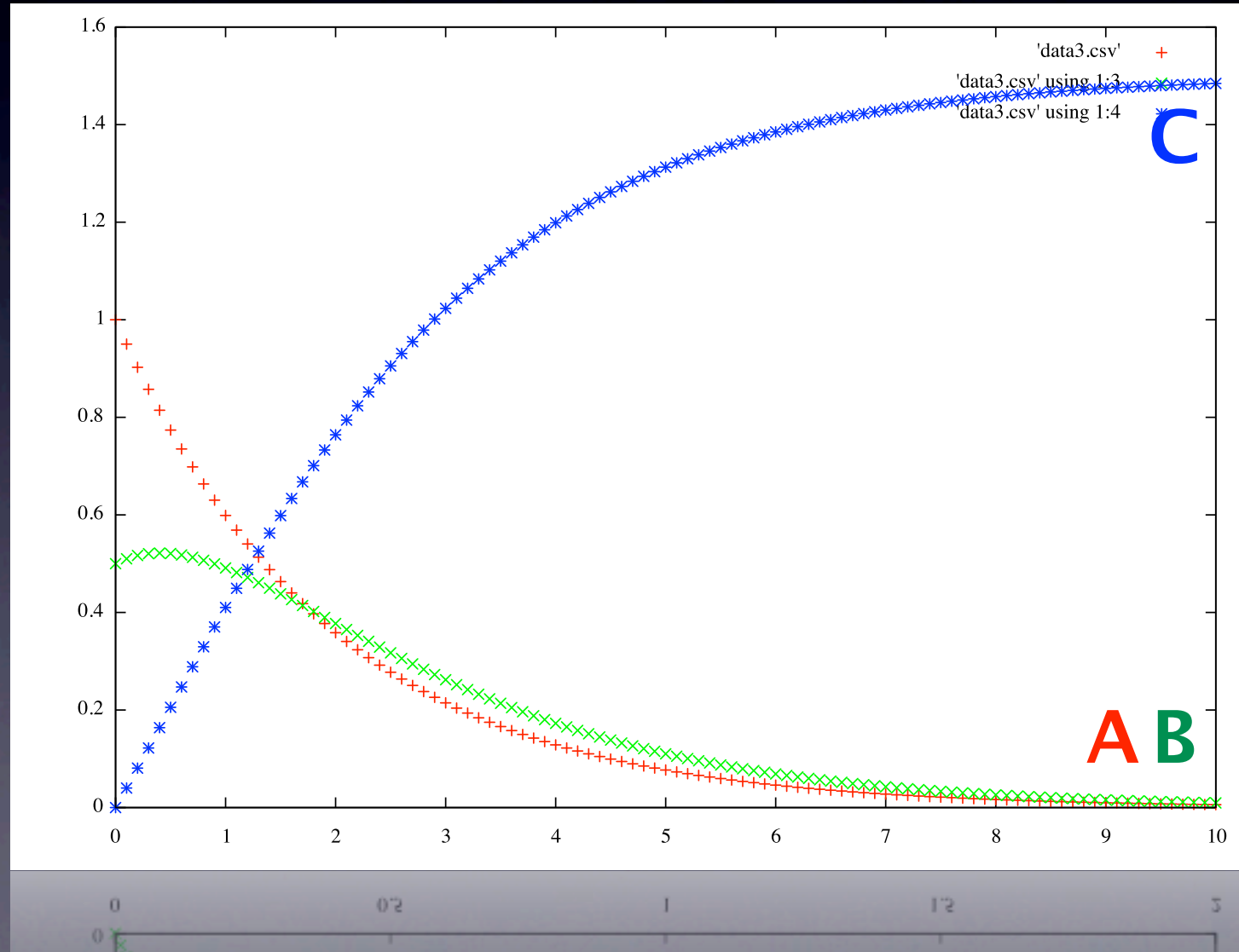
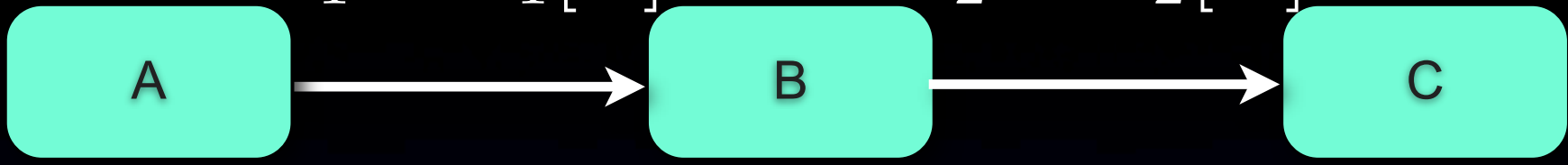
$$[A]_0 = 1.0, [B]_0 = 0.5, [C]_0 = 0.0$$

$$k_1 = 0.5, k_2 = 0.8$$

3変数モデル

$$v_1 = k_1 [A]$$

$$v_2 = k_2 [B]$$




```
#include<stdio.h>
```

C言語版

```
int main(void) {  
    int i;  
    double t = 0.0;  
    double dt = 0.1;  
    double v1, v2;  
    double A = 1.0, B = 0.5, C = 0.0;  
    double k1 = 0.5, k2 = 0.8;  
  
    for (i = 0; i <= 100; i++) {  
        printf("%lf, %lf, %lf, %lf\n", t, A, B, C);  
        v1 = k1 * A;  
        v2 = k2 * B;  
        A = A - v1*dt;          /* dA/dt = -v1      */  
        B = B + v1*dt - v2*dt; /* dB/dt =  v1 - v2 */  
        C = C + v2*dt;          /* dC/dt =  v2      */  
        t = t + dt;  
    }  
    return 0;  
}
```

```
function xdot = sim3(time,x)
```

```
xdot = zeros(3,1);
```

```
k1 = 0.5;
```

```
k2 = 0.8;
```

```
if ( nargin == 0 )
```

```
    xdot(1) = 1.0; % A
```

```
    xdot(2) = 0.5; % B
```

```
    xdot(3) = 0.0; % C
```

```
else
```

```
    A = x(1);
```

```
    B = x(2);
```

```
    C = x(3);
```

```
    v1 = k1 * A;
```

```
    v2 = k2 * B;
```

```
    xdot(1) = -v1; % dA/dt
```

```
    xdot(2) = v1 - v2; % dB/dt
```

```
    xdot(3) = v2; % dC/dt
```

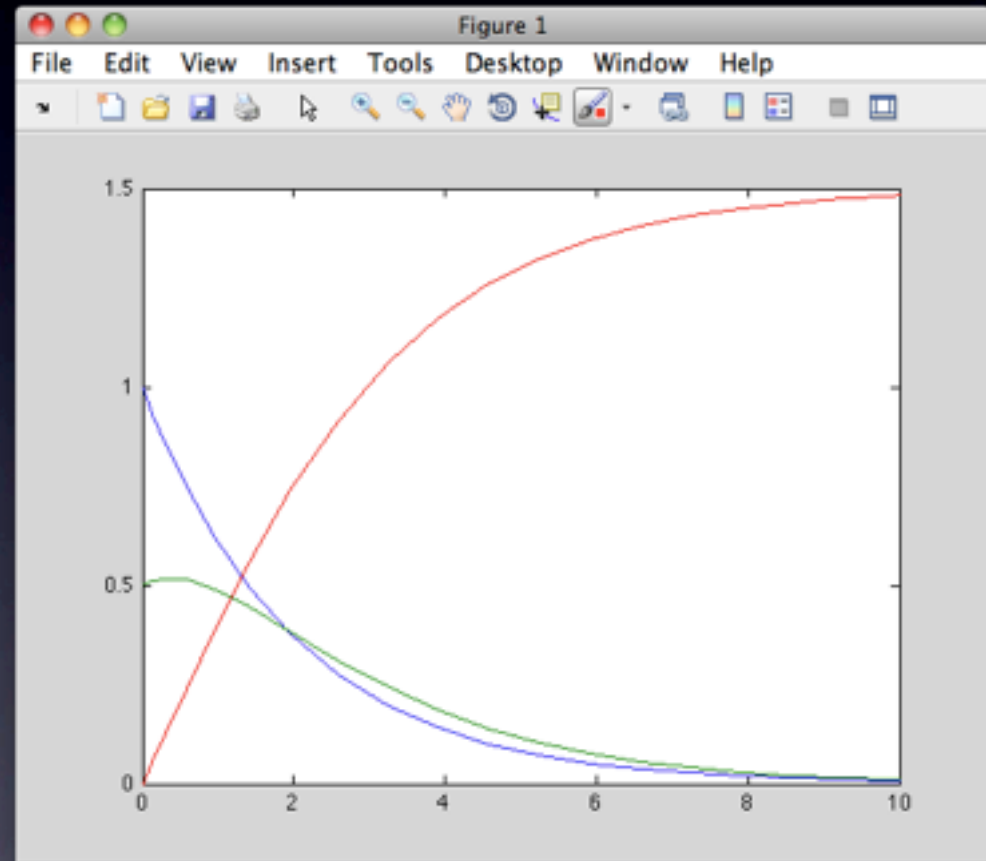
```
end;
```

変数の数

初期値

MATLAB版

sim3.m



```
>> [t,x] = ode23(@sim3,[0 10],sim3);
```

```
>> plot(t,x)
```

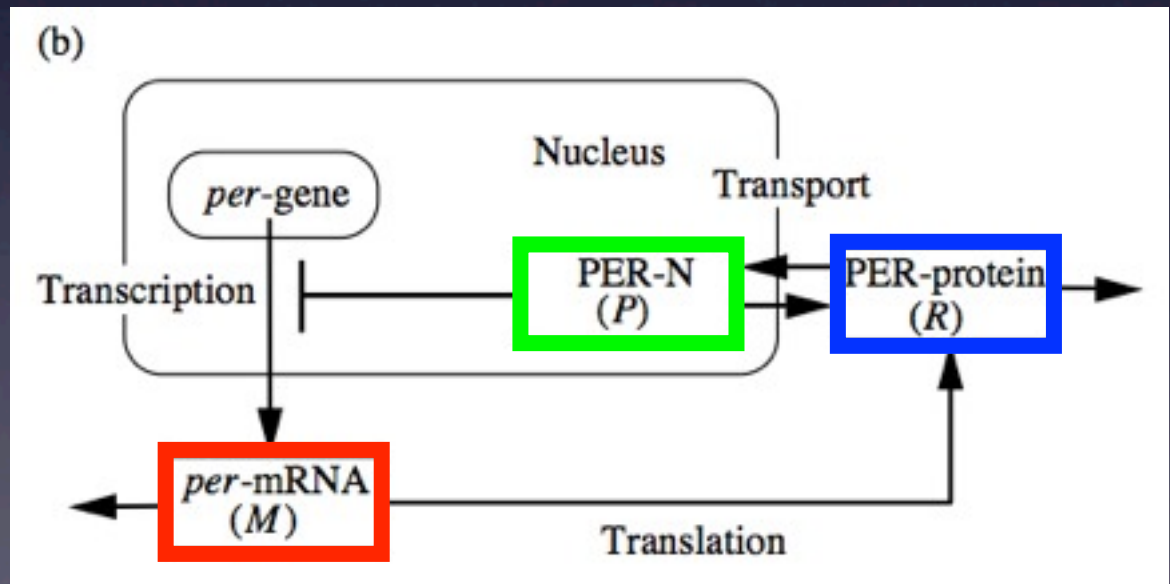
Octave-Forge
odepkg でも可

本格的なモデル

Circadian clock model

- mRNA (**M**) の生産は核内のタンパク質 (**P**) によって抑制される
- **M** は細胞質でタンパク質 (**R**) に翻訳される
- **P** / **R** はそれぞれ細胞質 / 核内に輸送される

$$\frac{dM}{dt} = \frac{1}{1 + (P/h)^n} - aM,$$
$$\frac{dR}{dt} = sM - (d + u)R + vP,$$
$$\frac{dP}{dt} = uR - vP.$$



J. theor. Biol. (2002) 216, 193-208

Circadian clock model

$$\frac{dM}{dt} = \frac{1}{1 + (P/h)^n} - aM,$$

$$\frac{dR}{dt} = sM - (d + u)R + vP,$$

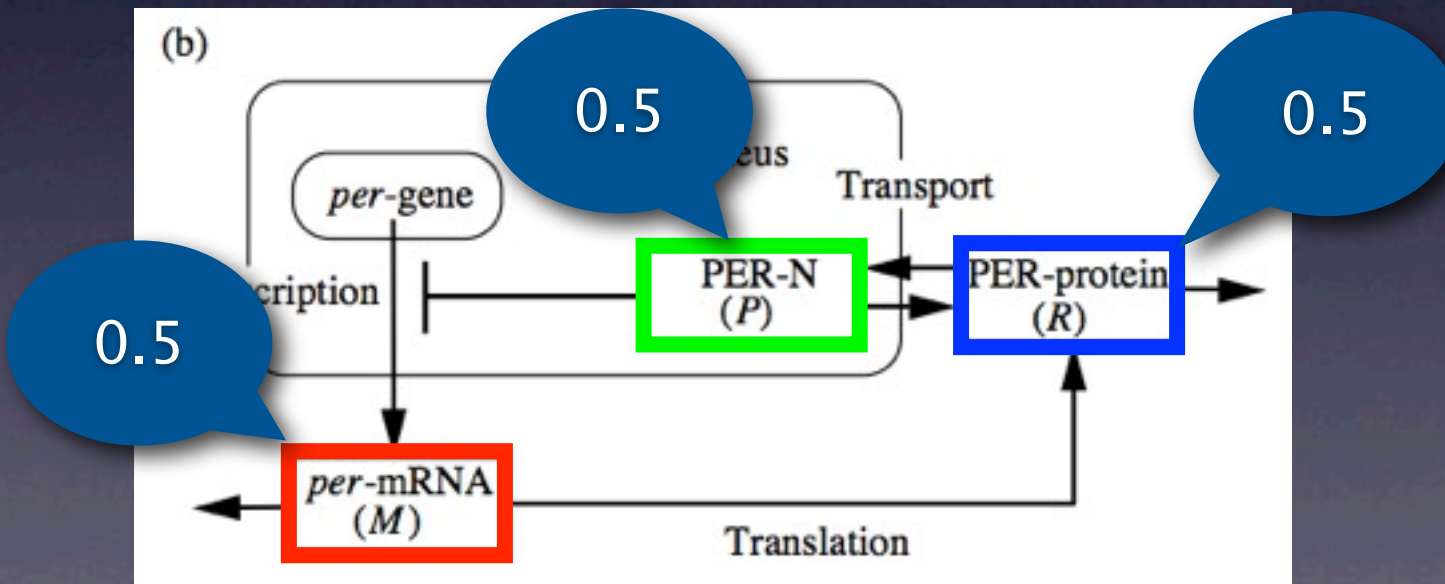
$$\frac{dP}{dt} = uR - vP.$$

$$a = s = d = v = 1.0$$

$$u = 0.1$$

$$h = 0.01$$

$$n = 40$$



Circadian clock model

$$\frac{dM}{dt} = \frac{1}{1 + (P/h)^n} - aM,$$

$$\frac{dR}{dt} = sM - (d + u)R + vP,$$

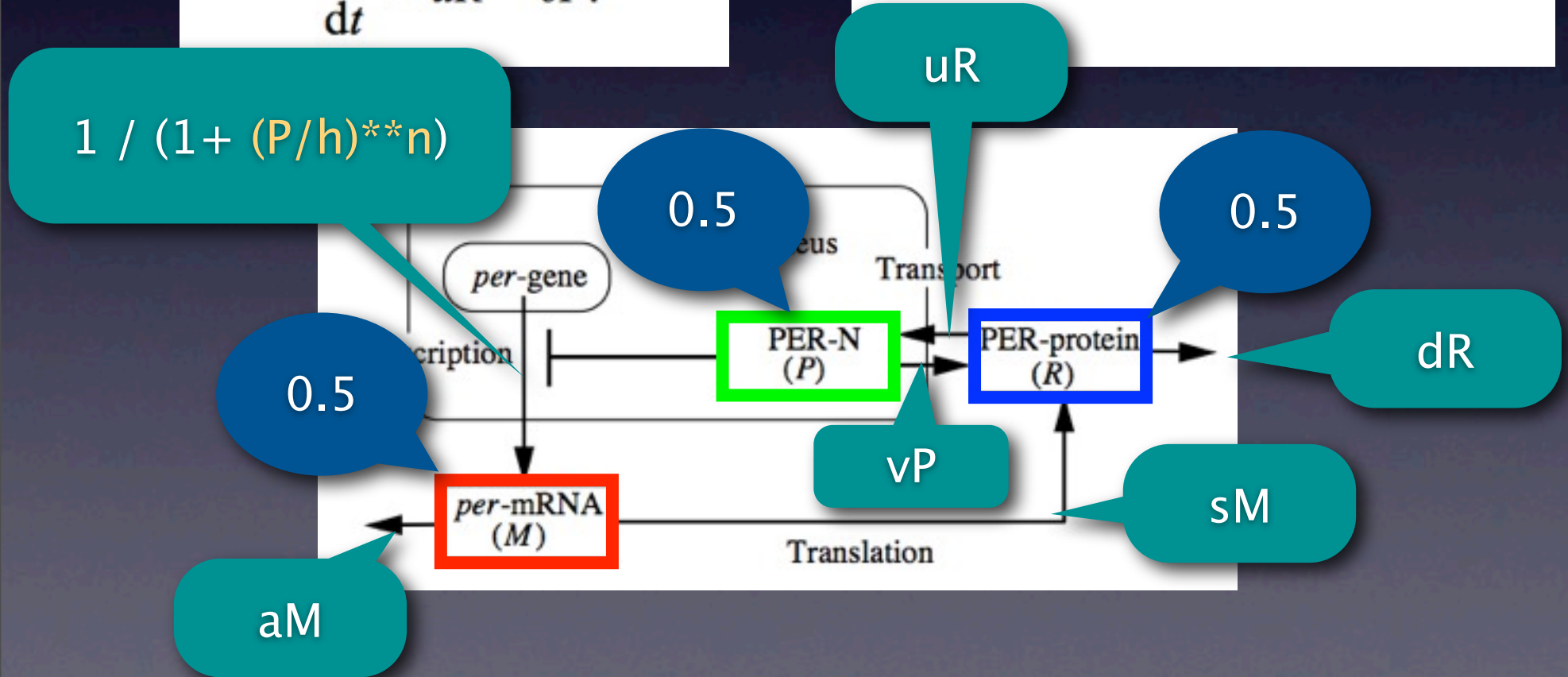
$$\frac{dP}{dt} = uR - vP.$$

$$a = s = d = v = 1.0$$

$$u = 0.1$$

$$h = 0.01$$

$$n = 40$$



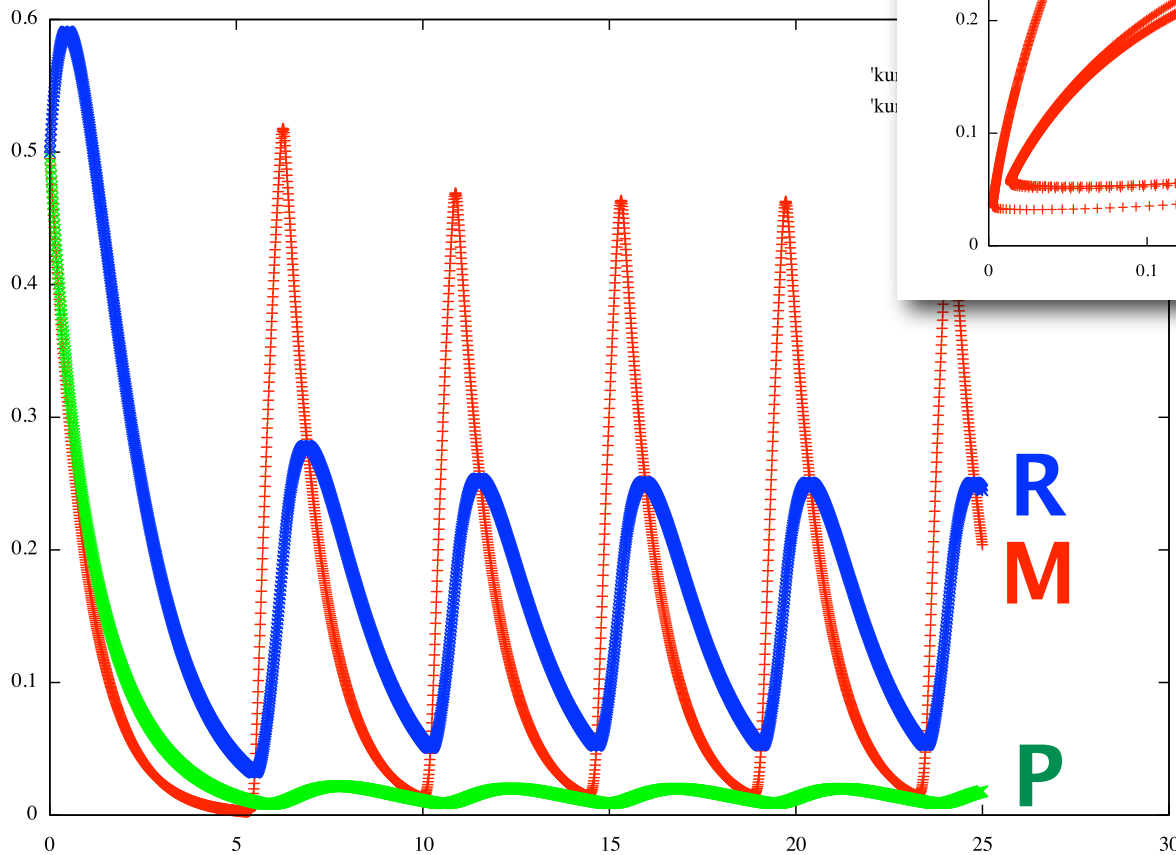
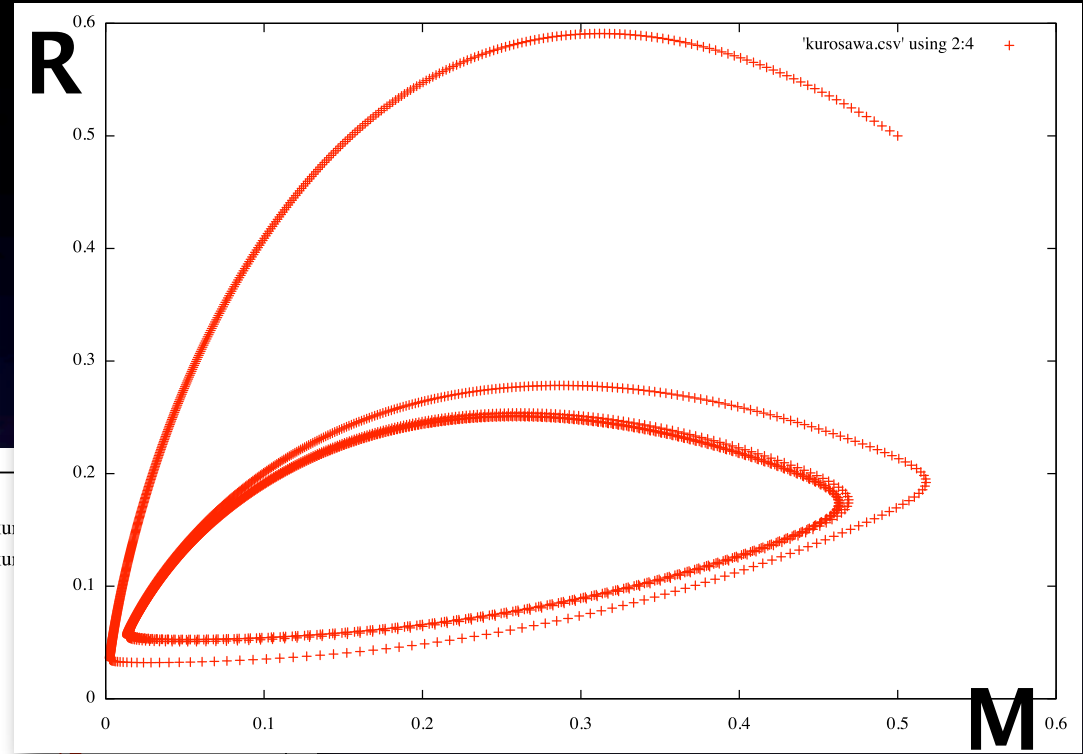
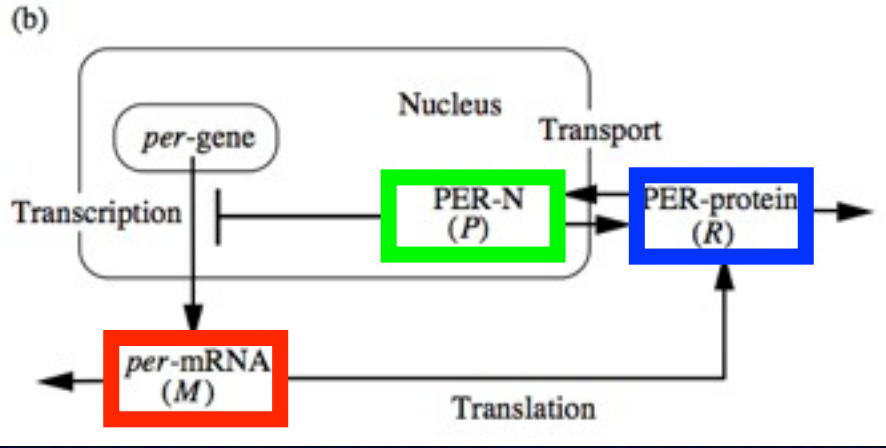
Perl版

```
#!/usr/bin/perl

# Time variables
$t = 0.0;
$dt = 0.01;
# Parameters
$a = 1.0; $s = 1.0; $d = 1.0; $v = 1.0;
$u = 0.1; $h = 0.01;
$n = 40;
# Molecules
$M = 0.5; $P = 0.5; $R = 0.5;

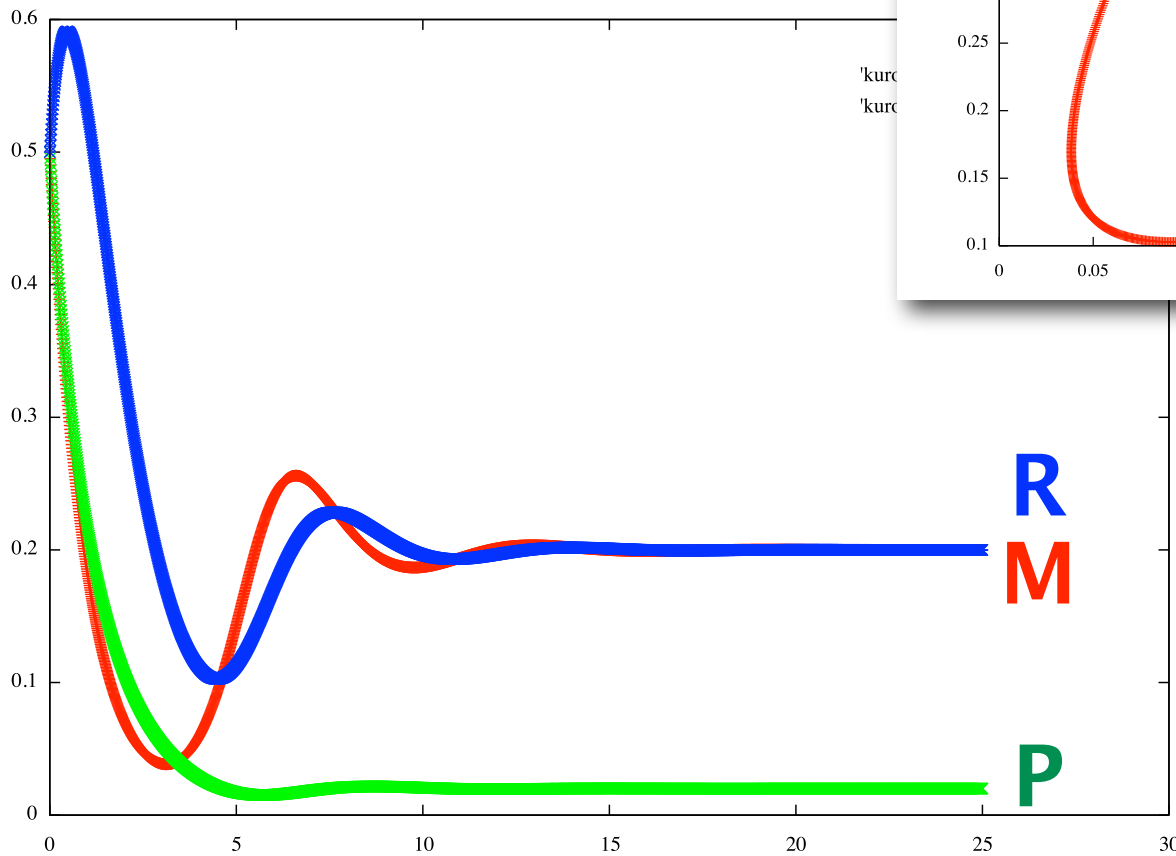
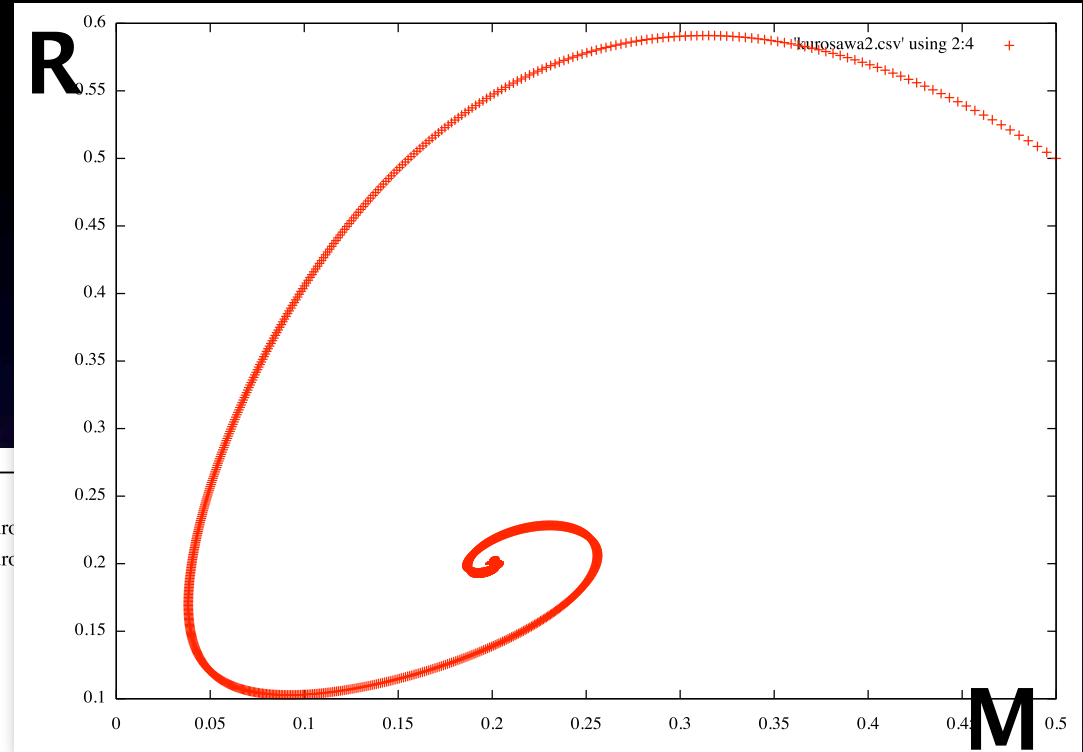
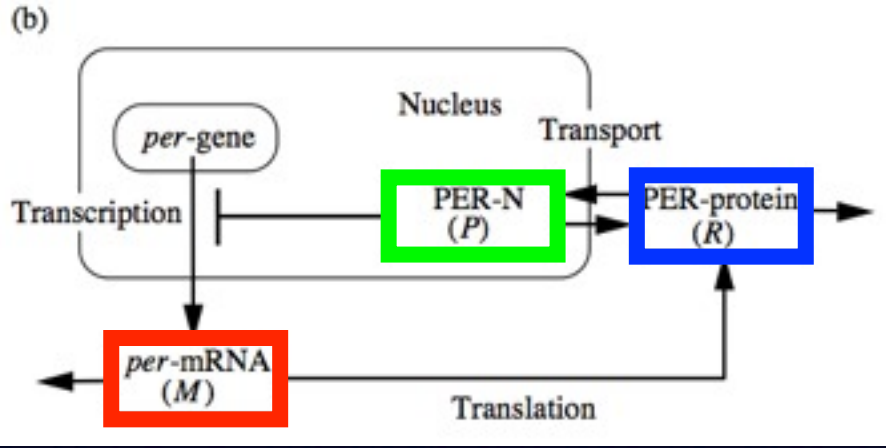
for ($i = 0; $i <= 2500; $i++) {
    print "$t, $M, $P, $R\n";
    $dMdt = 1 / (1 + ($P/$h)**$n) - $a * $M;
    $dRdt = $s*$M - ($d+$u)*$R + $v*$P;
    $dPdt = $u*$R - $v*$P;
    $M = $M + $dMdt * $dt;
    $R = $R + $dRdt * $dt;
    $P = $P + $dPdt * $dt;
    $t = $t + $dt;
}
```

Circadian clock model



$$n = 40$$

Circadian clock model



$$n = 2$$

C言語版

```
#include<stdio.h>
#include<math.h>
int main(void) {
    int i;
    /* Time variables */
    double t = 0.0, dt = 0.01;
    /* Parameters */
    double a = 1.0, s = 1.0, d = 1.0, v = 1.0;
    double u = 0.1, h = 0.01;
    double n = 40;
    /* Molecules */
    double M = 0.5, P = 0.5, R = 0.5;
    double dMdt, dRdt, dPdt;

    for (i = 0; i <= 2500; i++) {
        printf("%lf, %lf, %lf, %lf\n", t, M, P, R);
        dMdt = 1 / (1 + pow(P/h,n)) - a * M;
        dRdt = s*M - (d+u)*R + v*P;
        dPdt = u*R - v*P;
        M = M + dMdt * dt;
        R = R + dRdt * dt;
        P = P + dPdt * dt;
        t = t + dt;
    }
    return 0;
}
```


MATLAB版

```
function xdot = kurosawa(time,x)
```

```
xdot = zeros(3,1);
```

```
a = 1.0; s = 1.0; d = 1.0; v = 1.0;
```

```
u = 0.1; h = 0.01;
```

```
n = 40;
```

```
if ( nargin == 0 )
```

```
    xdot(1) = 0.5; % dM
```

```
    xdot(2) = 0.5; % dP
```

```
    xdot(3) = 0.5; % dR
```

```
else
```

```
    M = x(1);
```

```
    P = x(2);
```

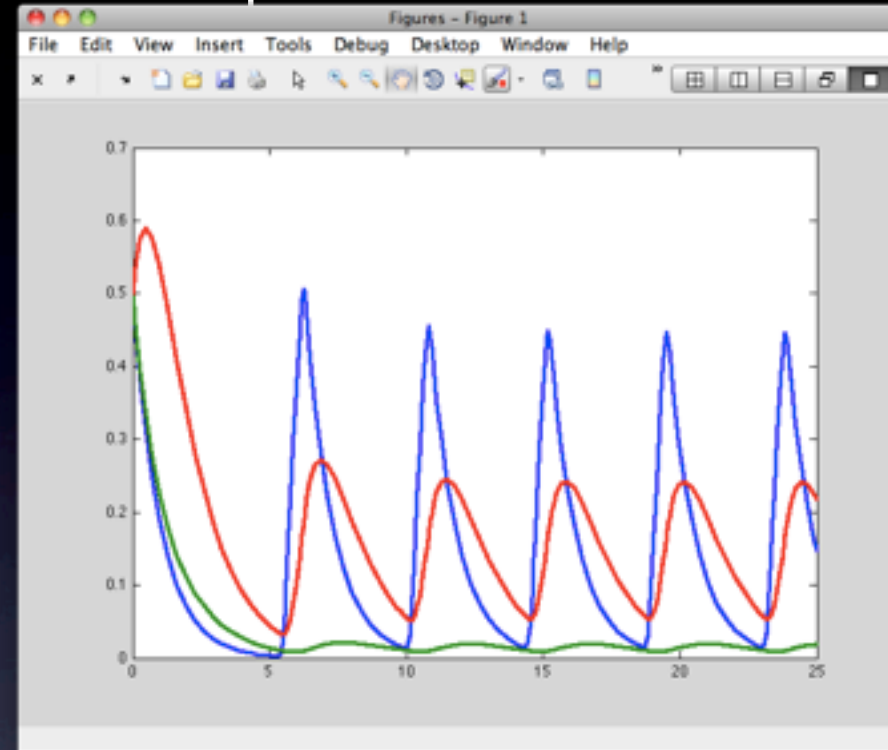
```
    R = x(3);
```

```
    xdot(1) = 1/(1+(P/h)^n) - a*M; % dM/dt
```

```
    xdot(2) = u*R - v*P; % dP/dt
```

```
    xdot(3) = s*M - (d+u)*R + v*P; % dR/dt
```

```
end;
```



kurosawa.m

Octave-Forge
odepkg でも可

```
>> [t,x] = ode23(@kurosawa,[0 10],kurosawa);  
>> plot(t,x)
```

今日の目標

- 数理モデルの構築
- シミュレータの実装
- シミュレーション

シミュレーション
してみたい!

どれを選ばいい?

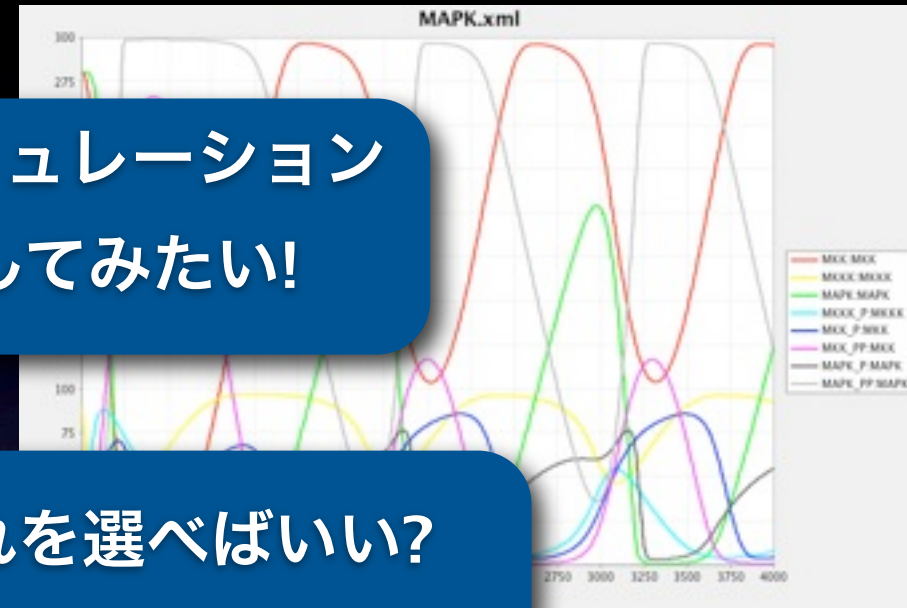
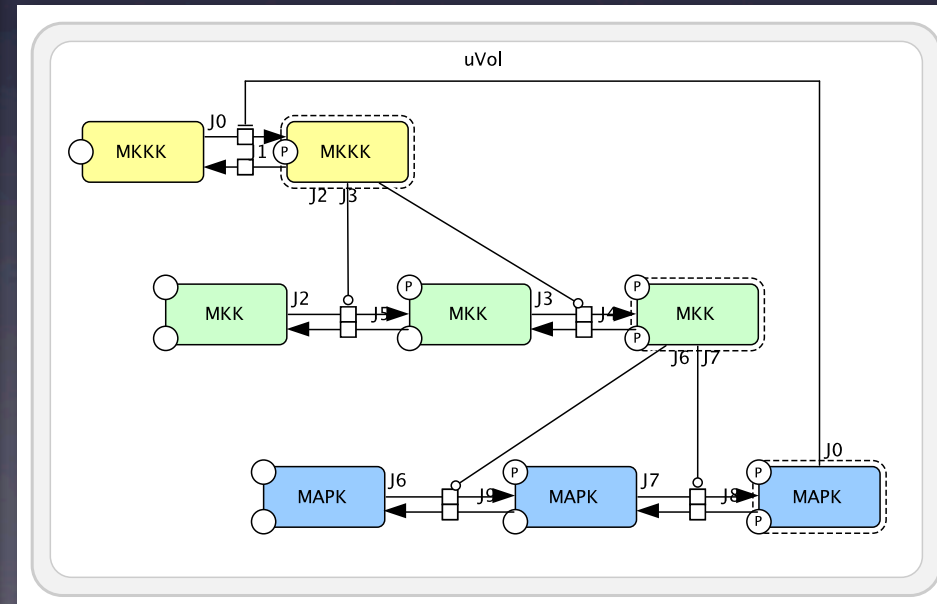


Table 1. Kinetic equations comprising the computational model of the MAPK cascade.

$$\begin{aligned}
 d[\text{MKKK}]/dt &= v_2 - v_1 \\
 d[\text{MKKK-P}]/dt &= v_1 - v_2 \\
 d[\text{MKK}]/dt &= v_6 - v_3 \\
 d[\text{MKK-P}]/dt &= v_3 + v_5 - v_4 - v_6 \\
 d[\text{MKK-PP}]/dt &= v_4 - v_5 \\
 d[\text{MAPK}]/dt &= v_{10} - v_7 \\
 d[\text{MAPK-P}]/dt &= v_7 + v_9 - v_8 - v_{10} \\
 d[\text{MAPK-PP}]/dt &= v_8 - v_9
 \end{aligned}$$

Moiety conservation relations:

$$\begin{aligned}
 [\text{MKKK}]_{\text{total}} &= [\text{MKKK}] + [\text{MKKK-P}] \\
 [\text{MKK}]_{\text{total}} &= [\text{MKK}] + [\text{MKK-P}] + [\text{MKK-PP}] \\
 [\text{MAPK}]_{\text{total}} &= [\text{MAPK}] + [\text{MAPK-P}] + [\text{MAPK-PP}]
 \end{aligned}$$



ここから少し難しくなります

数値計算での注意点

● 精度

- 浮動小数点数 (単精度と倍精度)

丸め誤差

● 数値積分

- 誤差とTaylor展開と刻み幅
- 硬い(stiff)微分方程式 (陽解法、陰解法)
- 乱数の周期性

打ち切り誤差

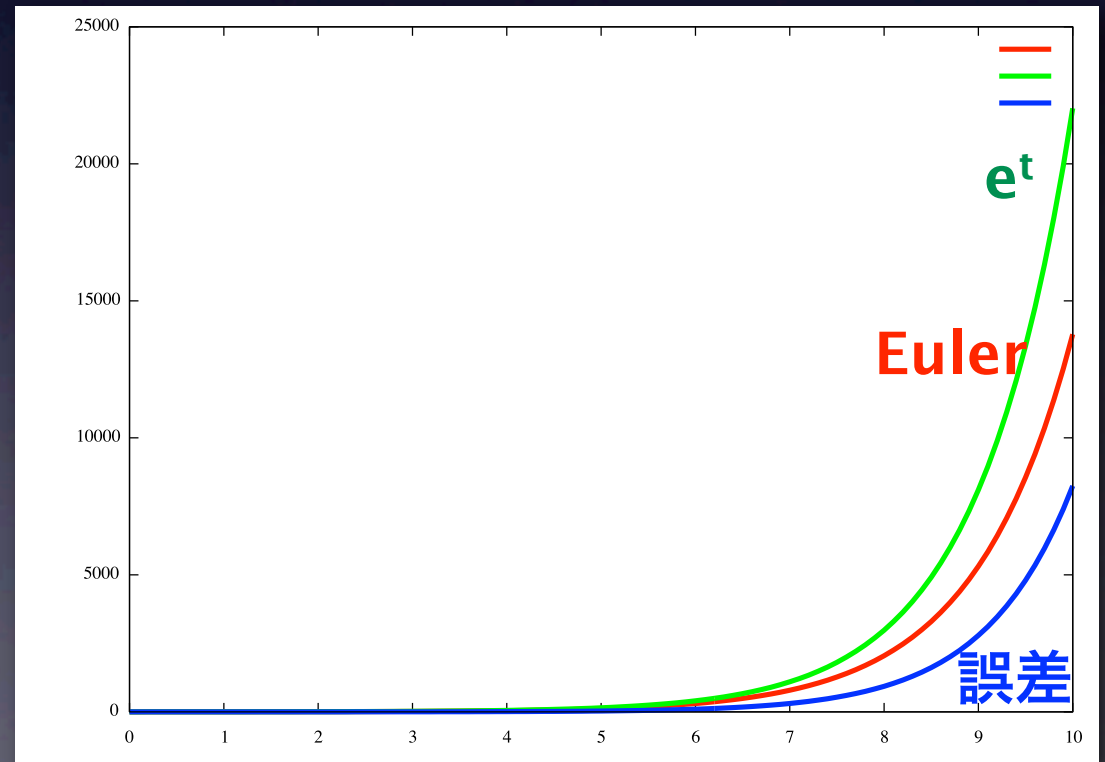
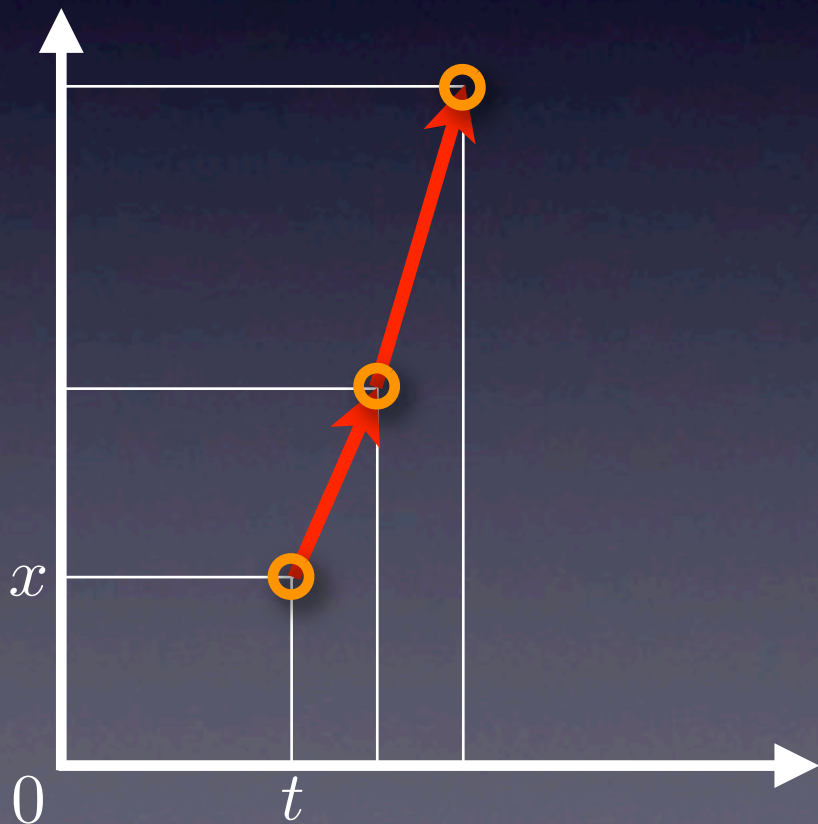
● 速度

- 刻み幅(精度と計算時間のトレードオフ)
- 高速化(アルゴリズム、言語、ハードウェア)

Euler法と誤差

$$\frac{dx}{dt} = x \quad (t = 0, x = 1)$$

$$x = e^t$$



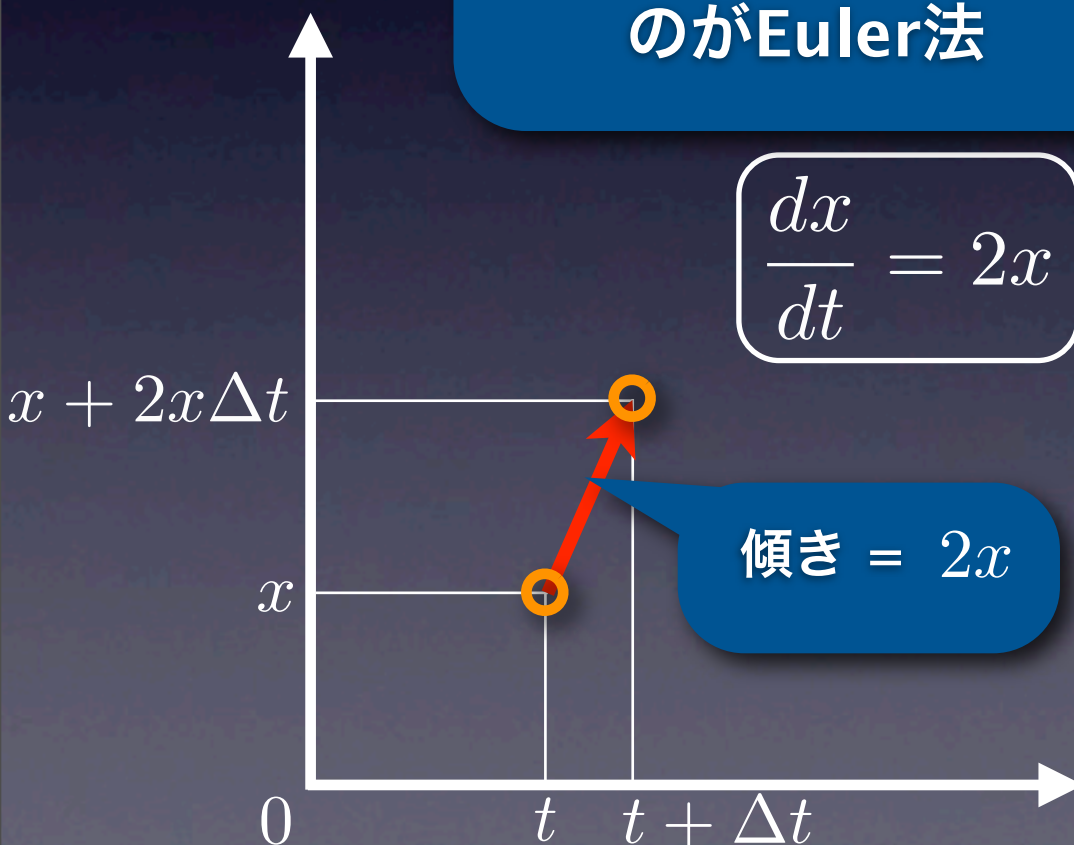
Taylor展開とEuler法

$$x(t + \Delta t) = x(t) + \Delta t x'(t) + \frac{1}{2} \Delta t^2 x''(t) + \frac{1}{3!} \Delta t^3 x^{(3)}(t) + \dots$$

ここまでで打ち切った
のがEuler法

$$\dots + \frac{1}{n!} \Delta t^n x^{(n)}(t)$$

打ち切り誤差



$$\text{打ち切り誤差} = O(\Delta t^2)$$

刻み幅を1/2にすれば
誤差は1/4

Ruge-Kutta法(4次)

$$x(t + \Delta t) = x(t) + \Delta t x'(t) + \frac{1}{2} \Delta t^2 x''(t) + \frac{1}{3!} \Delta t^3 x^{(3)}(t) + \dots$$

$$\dots + \frac{1}{n!} \Delta t^n x^{(n)}(t)$$

$$d_1 = \Delta t \cdot f(t, x)$$

$$d_2 = \Delta t \cdot f\left(t + \frac{\Delta t}{2}, x(t) + \frac{d_1}{2}\right)$$

$$d_3 = \Delta t \cdot f\left(t + \frac{\Delta t}{2}, x(t) + \frac{d_2}{2}\right)$$

$$d_4 = \Delta t \cdot f(t + \Delta t, x(t) + d_3)$$

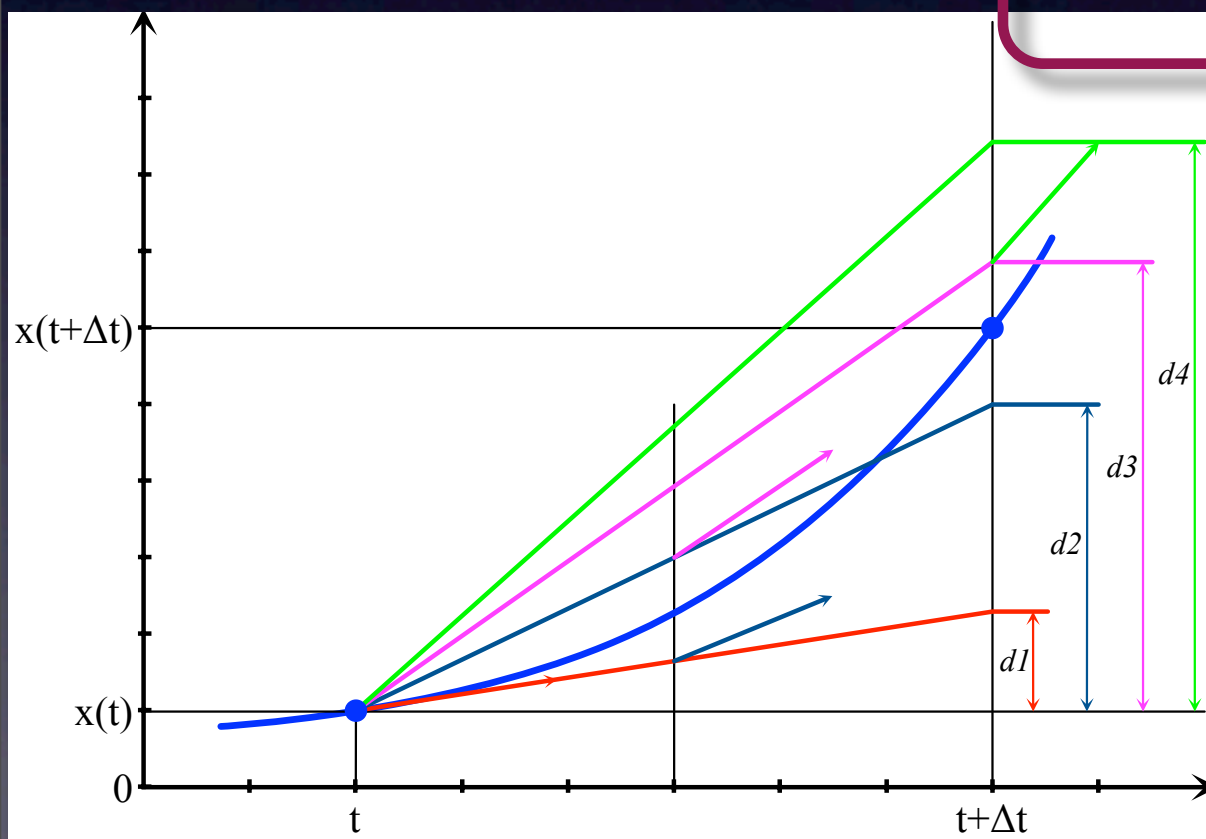
$$x(t + \Delta t) = x(t) + \frac{d_1 + 2d_2 + 2d_3 + d_4}{6}$$

打ち切り誤差

Ruge-Kutta法(4次)

$$x(t + \Delta t) = x(t) + \Delta t x'(t) + \frac{1}{2} \Delta t^2 x''(t) + \frac{1}{3!} \Delta t^3 x^{(3)}(t) + \dots$$

$$\dots + \frac{1}{n!} \Delta t^n x^{(n)}(t)$$



打ち切り誤差

$$\text{打ち切り誤差} = O(\Delta t^5)$$

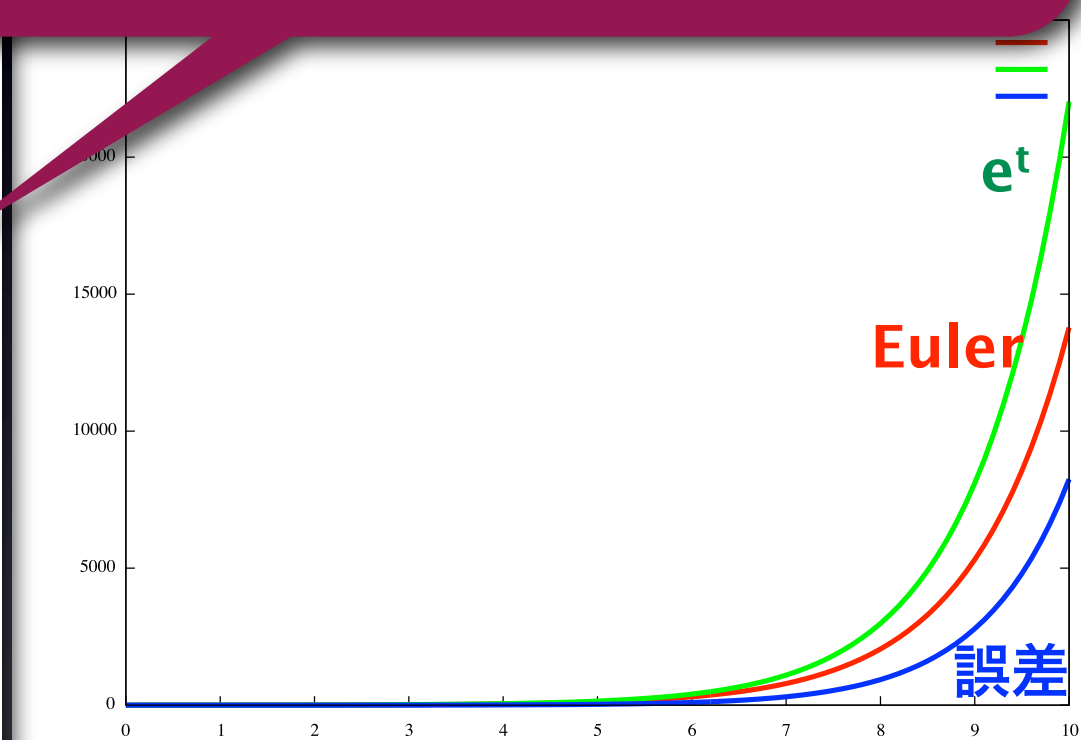
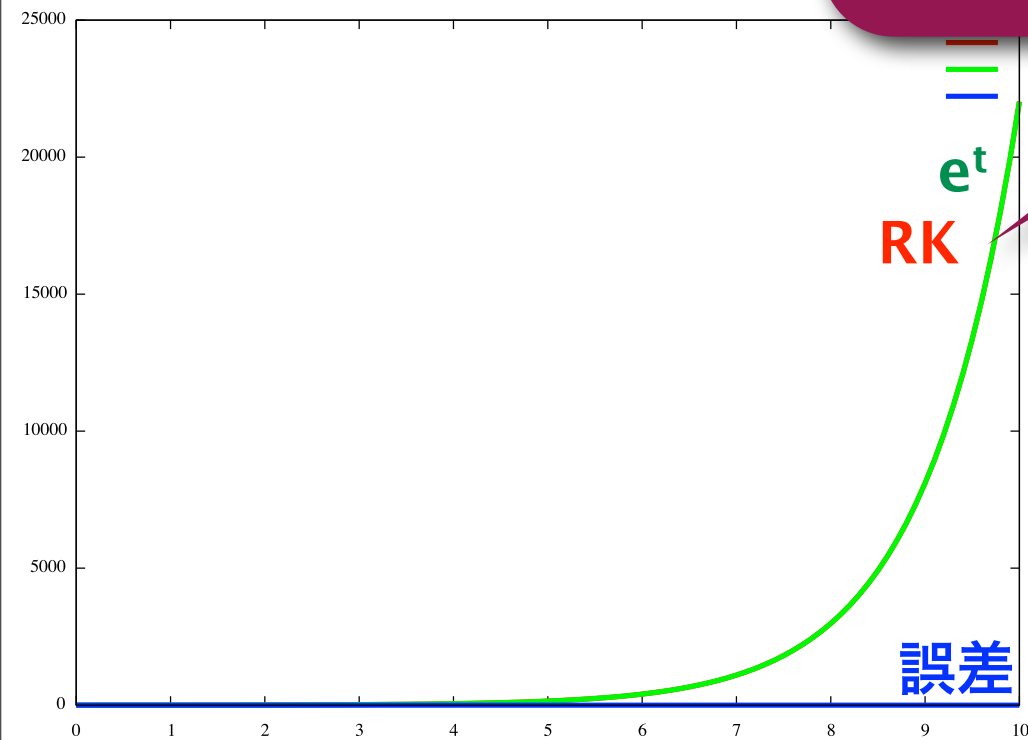
刻み幅を1/2にすれば
誤差は1/32

Ruge-Kutta法と誤差

$$\frac{dx}{dt} = x \quad (t = 0, x = 1)$$

$$x = e^t$$

計算時間は4倍になるが、打ち切り誤差を小さくできるため精度は劇的に向上



Perl版

```
#!/usr/bin/perl
```

```
$dt = 0.1;
$t = 0.0;
$x = 1.0;
for ($i = 0; $i <= 100; $i++) {
    print "$t,$x\n";
    $d1 = &dxdt($t, $x) * $dt;
    $d2 = &dxdt($t+$dt/2, $x+$d1/2) * $dt;
    $d3 = &dxdt($t+$dt/2, $x+$d2/2) * $dt;
    $d4 = &dxdt($t+$dt, $x+$d3) * $dt;
    $dx = ($d1 + 2*$d2 + 2*$d3 + $d4) / 6;
    $x = $x + $dx;
    $t = $t + $dt;
}
```

```
sub dxdt {
    my ($t, $x) = @_;
    my $dxdt = $x; # dx/dt = x
    return $dxdt;
}
```


C言語版

```
#include<stdio.h>
```

```
double dxdt(double t, double x) {  
    double dxdt = x; /* dx/dt = x */  
    return dxdt;  
}
```

```
int main(void) {  
    int i;  
    double t = 0.0, dt = 0.1;  
    double x = 1.0;  
    double dx, d1, d2, d3, d4;  
  
    for (i = 0; i <= 100; i++) {  
        printf("%lf,%lf\n", t, x);  
        d1 = dxdt(t, x) * dt;  
        d2 = dxdt(t+dt/2, x+d1/2) * dt;  
        d3 = dxdt(t+dt/2, x+d2/2) * dt;  
        d4 = dxdt(t+dt, x+d3) * dt;  
        dx = (d1 + 2*d2 + 2*d3 + d4) / 6;  
        x = x + dx;  
        t = t + dt;  
    }  
    return 0;  
}
```

陽解法と陰解法

- 陽解法 (時刻 t までの x の値で $x(t + \Delta t)$ が求まる)

- Euler法 Runge-Kutta法, Adams-Bashforth法

- 硬い微分方程式が解けない(発散する)

- 実装が容易

dx/dtが非常に大きく変動するモデル、
dx/dt, dy/dtのオーダーが大きく異なるモデル

- 陰解法 ($x(t + \Delta t)$ を求めるのに $x(t + \Delta t)$ が必要)

- 後退Euler法, Adams-Moulton法, Gear法

- 硬い微分方程式に対応

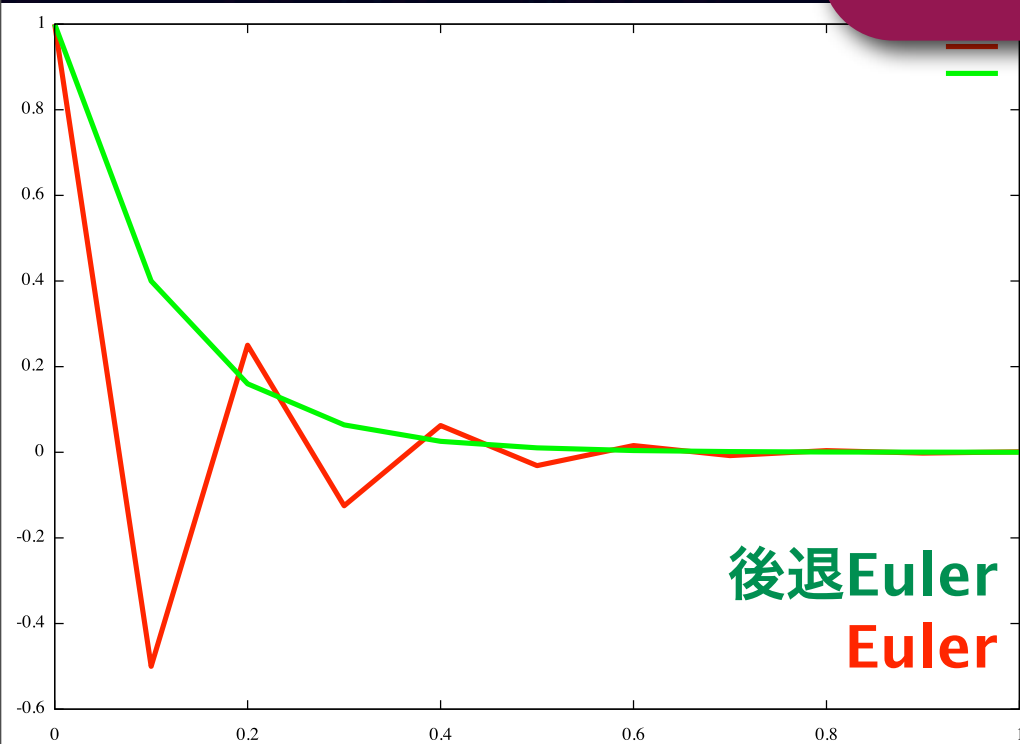
- 実装が面倒 (例: 数値微分, LU分解, ニュートン法, etc.)

Euler法と後退Euler法

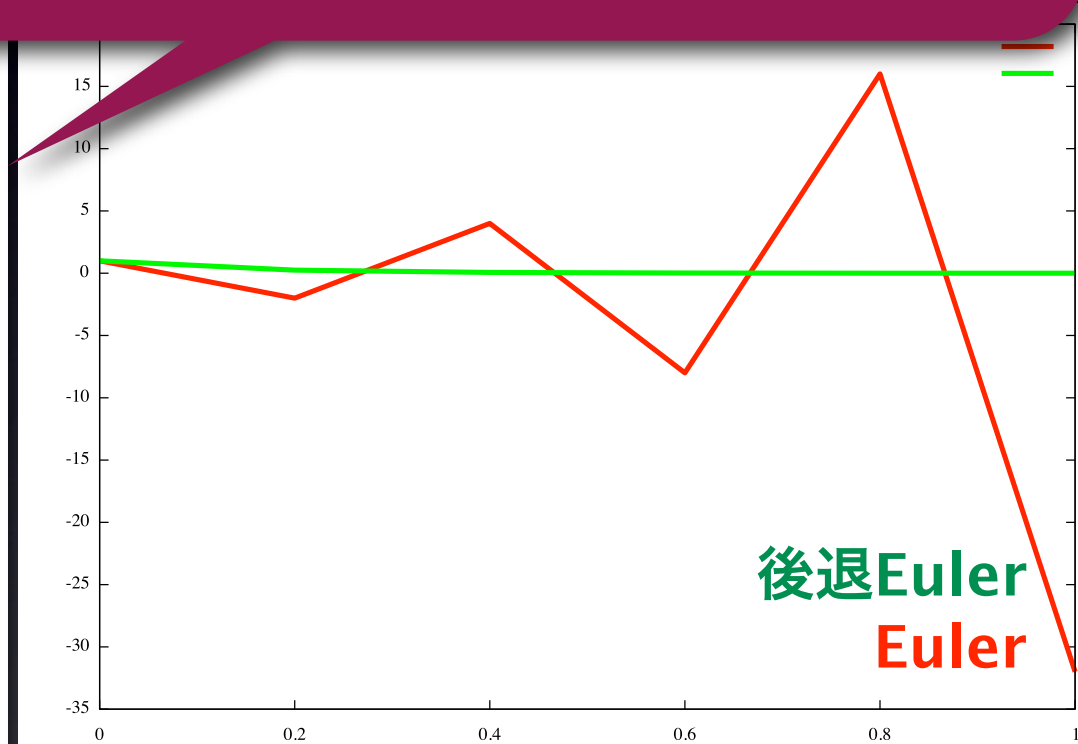
$$\frac{dx}{dt} = -15x \quad (t = 0, x = 1)$$

$$x = e^{-15t}$$

刻み幅によっては不安定になる、発散する



$\Delta t = 0.1$



$\Delta t = 0.2$

数値計算での注意点

● 精度

- 浮動小数点数 (単精度と倍精度)

丸め誤差

● 数値積分

- 誤差とTaylor展開と刻み幅

打ち切り誤差

- 硬い(stiff)微分方程式 (陽解法、陰解法)

● 乱数の周期性

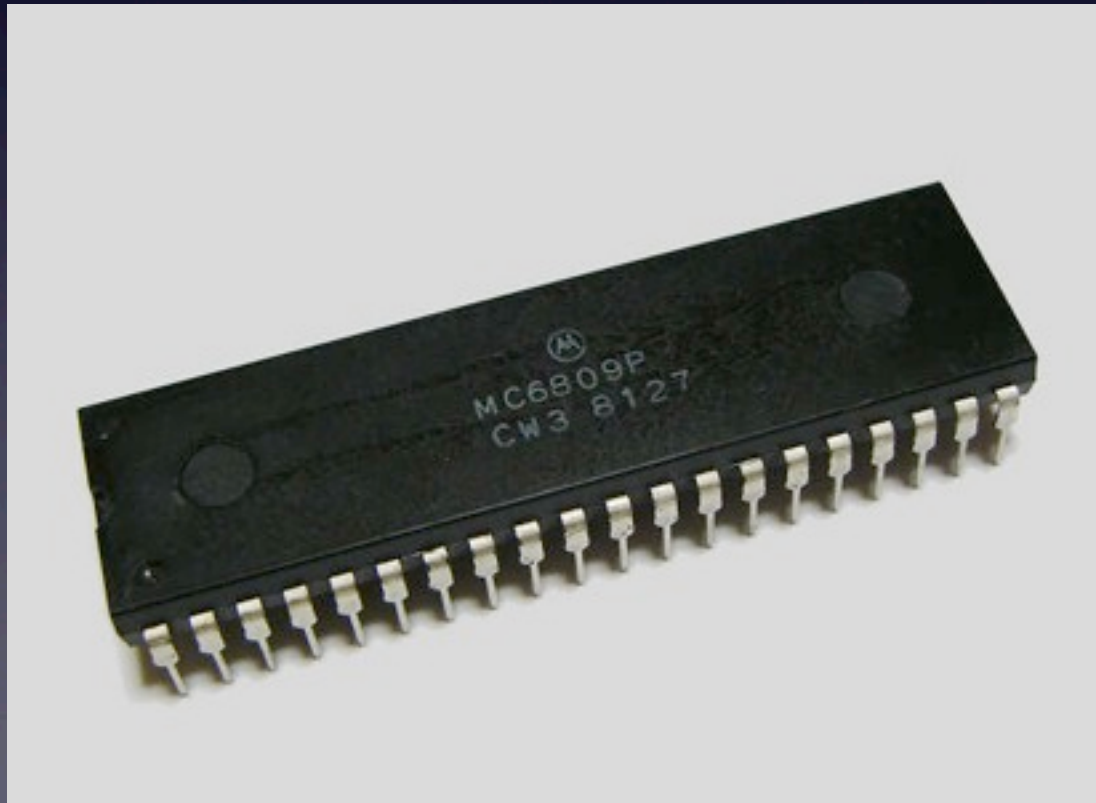
● 速度

- 刻み幅(精度と計算時間のトレードオフ)

- 高速化(アルゴリズム、言語、ハードウェア)

浮動小数点数

コンピュータの中で数字はどう表現される？



2進法

 2^3 2^2 2^1 2^0

1	0	1	1
---	---	---	---

- $1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 11$

実数はどう表現される?

```
$pi = 3.14;  
$sqr = 1.4142;
```

```
float pi = 3.14;  
double sqr = 1.4142;
```

まずは実験

```
#include<stdio.h>

int main(void) {
    int i;
    float sum = 0;
    float f = 1.1;

    for (i = 0; i < 10; i++) {
        sum = sum + f;
    }
    printf("sum = %f\n", sum);
    return 0;
}
```

1.1を10回足すだけの
プログラム

?!

```
./a.out
sum = 11.000001
```

浮動小数点表記

● 10進数

● $123400000 = 1.234 \times 10^8$

● $0.001234 = 1.234 \times 10^{-3}$

仮数

指数

● 2進数

● $11011000000 = 1.1011 \times 2^{10}$

● $0.0000010101 = 1.0101 \times 2^{-6}$

float型

● 2進数

● $11011000000 = 1.1011 \times 2^{10}$

● $0.0000010101 = 1.0101 \times 2^{-6}$

仮数

指数

符号部 (1bit)

float型 = 32bit

0 0 1 1 1 1 0 0 1 0 1 0 1 0

指数部 (8bit)

仮数部 (23bit)

$$\text{floatの値} = (-1)^{\text{符号部}} \times 2^{\text{指数部} - 127} \times 1.\text{仮数部}$$

仮数部の計算方法

仮数

指数

$$0.0000010101 = 1.0101 \times 2^{-6}$$

$$2^{-1} \quad 2^{-2} \quad 2^{-3} \quad 2^{-4} \quad 2^{-5} \quad \dots \quad 2^{-22} \quad 2^{-23}$$

		0	1	0	1	0	...	0	0
--	--	---	---	---	---	---	-----	---	---

$$\begin{aligned} 2^{-2} + 2^{-4} &= 0.25 + 0.0625 \\ &= 0.3125 \end{aligned}$$

double型

- 2進数

- $11011000000 = 1.1011 \times 2^{10}$

- $0.0000010101 = 1.0101 \times 2^{-6}$

仮数

指数

符号部 (1bit)

double型 = 64bit

指数部 (11bit)

仮数部 (52bit)

0.1を2進数で表すと

$$0.1_{(10)} = 0.00011001100110011\dots_{(2)}$$

```
#include<stdio.h>

int main(void) {
    int i;
    float sum = 0;
    float f = 1.1;

    for (i = 0; i < 10; i++) {
        sum += f;
    }
    printf("sum = %f\n", sum);
    return 0;
}
```

10進数の 0.1 は
2進数だと循環小数

丸め誤差

```
./a.out
sum = 11.000001
```

倍精度(double)で再挑戦

$$0.1_{(10)} = 0.00011001100110011\dots_{(2)}$$

```
#include<stdio.h>

int main(void) {
    int i;
    double sum = 0;
    double f = 1.1;

    for (i = 0; i < 10; i++) {
        sum += f;
    }
    printf("sum = %lf\n", sum);
    return 0;
}
```

10進数の 0.1 は
2進数だと循環小数

OK

```
./a.out
sum = 11.000000
```

浮動小数点数: まとめ

$$0.1_{(10)} = 0.00011001100110011\dots(2)$$

```
float f = 1.1;  
double f = 1.1;
```

- 単精度(float)は精度が悪い(有効数字6 or 7桁)
- 0.1より0.125の方が精度が高い($1/2^n$ は高精度)
- C言語を使う場合、floatは避ける
- floatとdoubleの計算時間の差は(今は)ない
- Perl等のスクリプト型言語は大概倍精度
- シミュレータが単精度か倍精度か確認すべき

余談: Javaのdouble

```
volatile double x, y, z, d;  
public void doTest() {  
    x = 9007199254740994.0; /* 2^53 + 2 */  
    y = 1.0 - 1/65536.0;  
    z = x + y;  
    d = z - x;  
    System.out.println("z = " + z);  
    System.out.println("d = " + d);  
}
```

```
z = 9.007199254740994E15  
d = 0.0
```

OK

gij 4.3.3 on
Linux/x86

```
z = 9.007199254740996E15  
d = 2.0
```

NG

数値計算での注意点: 結論

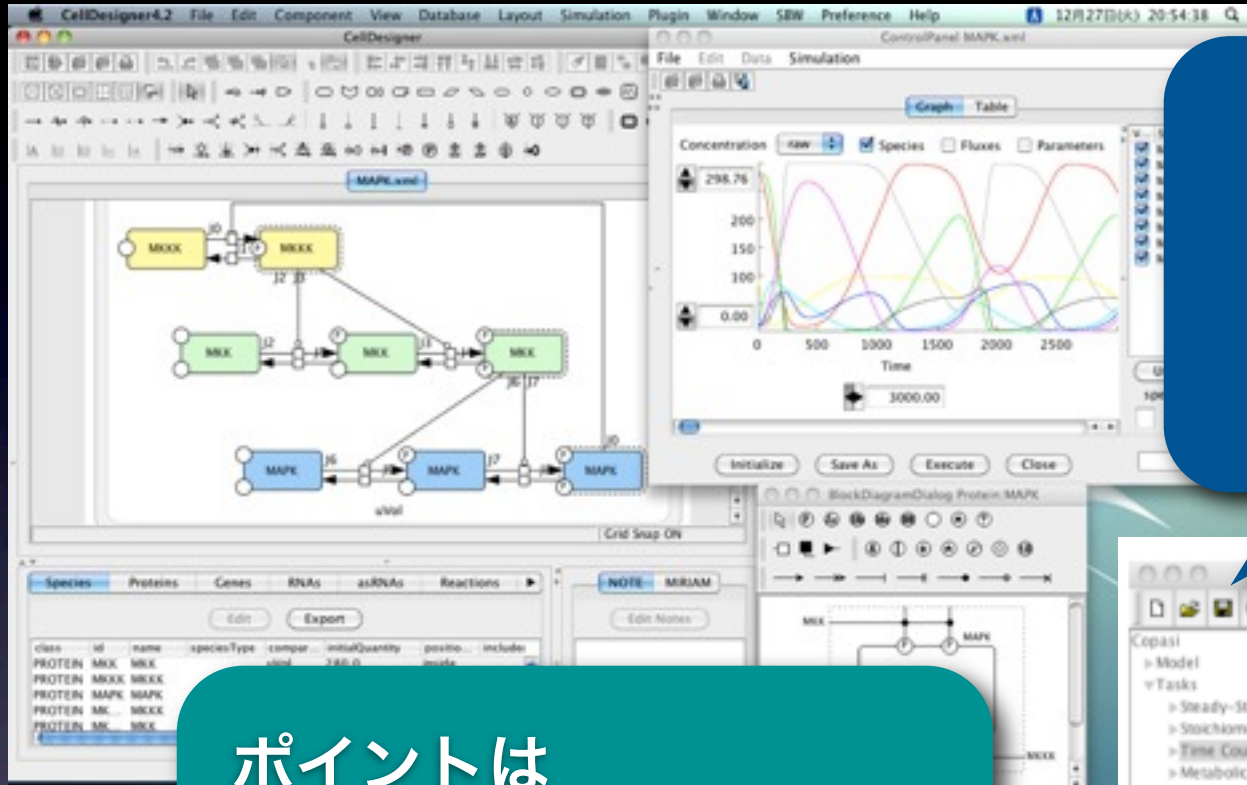
● 精度

- 単精度はだめ (C言語なら float ×, double ◎)
- Euler法で満足してはだめ Runge-Kutta法等を実装
- 陰解法もがんばって実装するとgood
- C言語標準の rand() はかなりいまいち
MT(Mersenne Twister)等の長周期性、均等性をもつ
高速な擬似乱数生成器を利用すべき
- SUNDIALS, LSODA, GSL などのライブラリを利用

● 速度

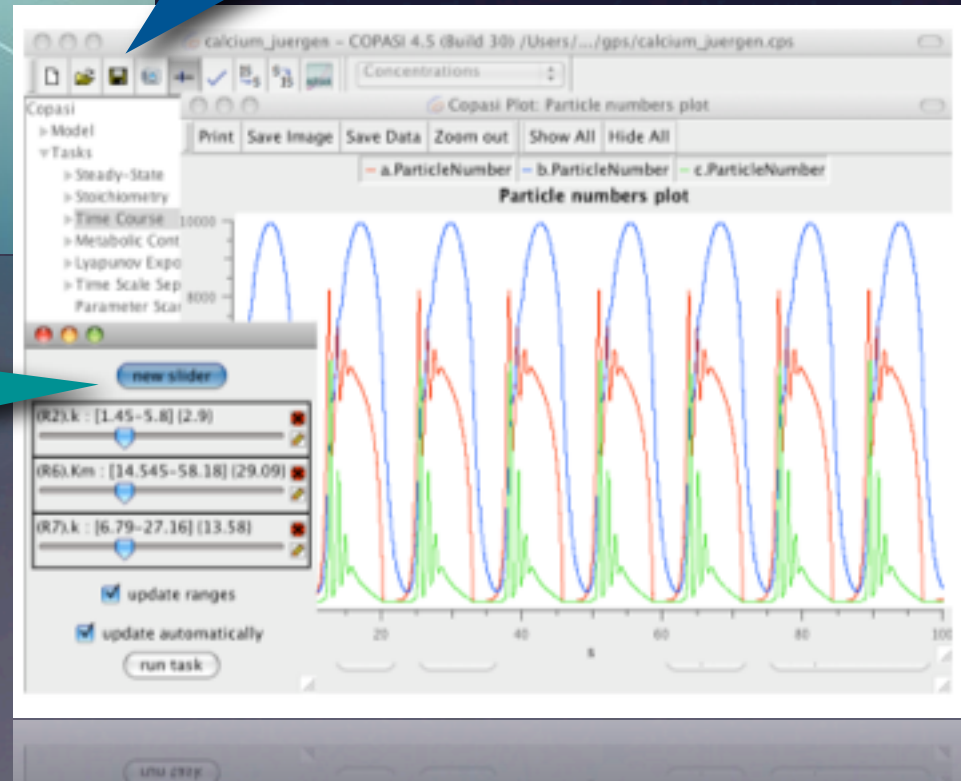
- 高速化の前に精度の保証を!
- 全部自前で実装しないと大して速くならない

シミュレータの選び方



1. 数理モデル構築
2. シミュレーション
3. プロット
4. 解析(あれば)

- ポイントは
1. 精度
 2. 使いやすさ
 3. モデルを再利用可能か

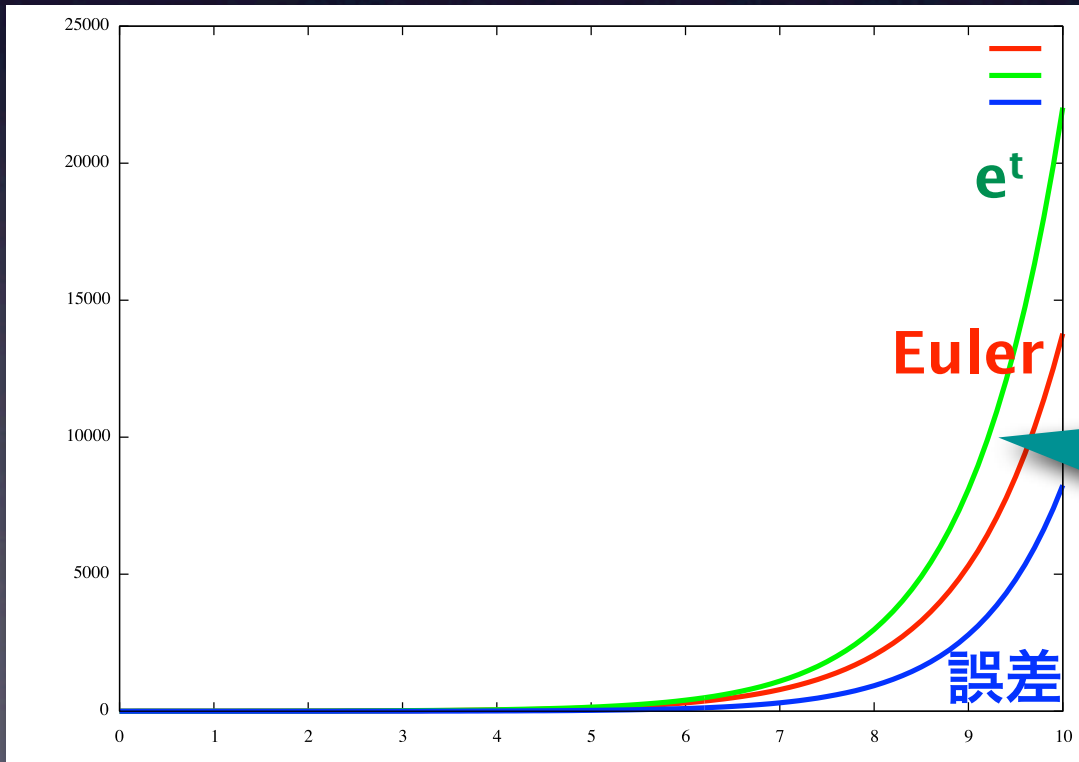


精度

$$\frac{dx}{dt} = x \quad (t = 0, x = 1) \quad \Delta t = 0.1$$

$$x = e^t$$

この条件でシミュレーション
を実行、 $x = e^t$ と比較



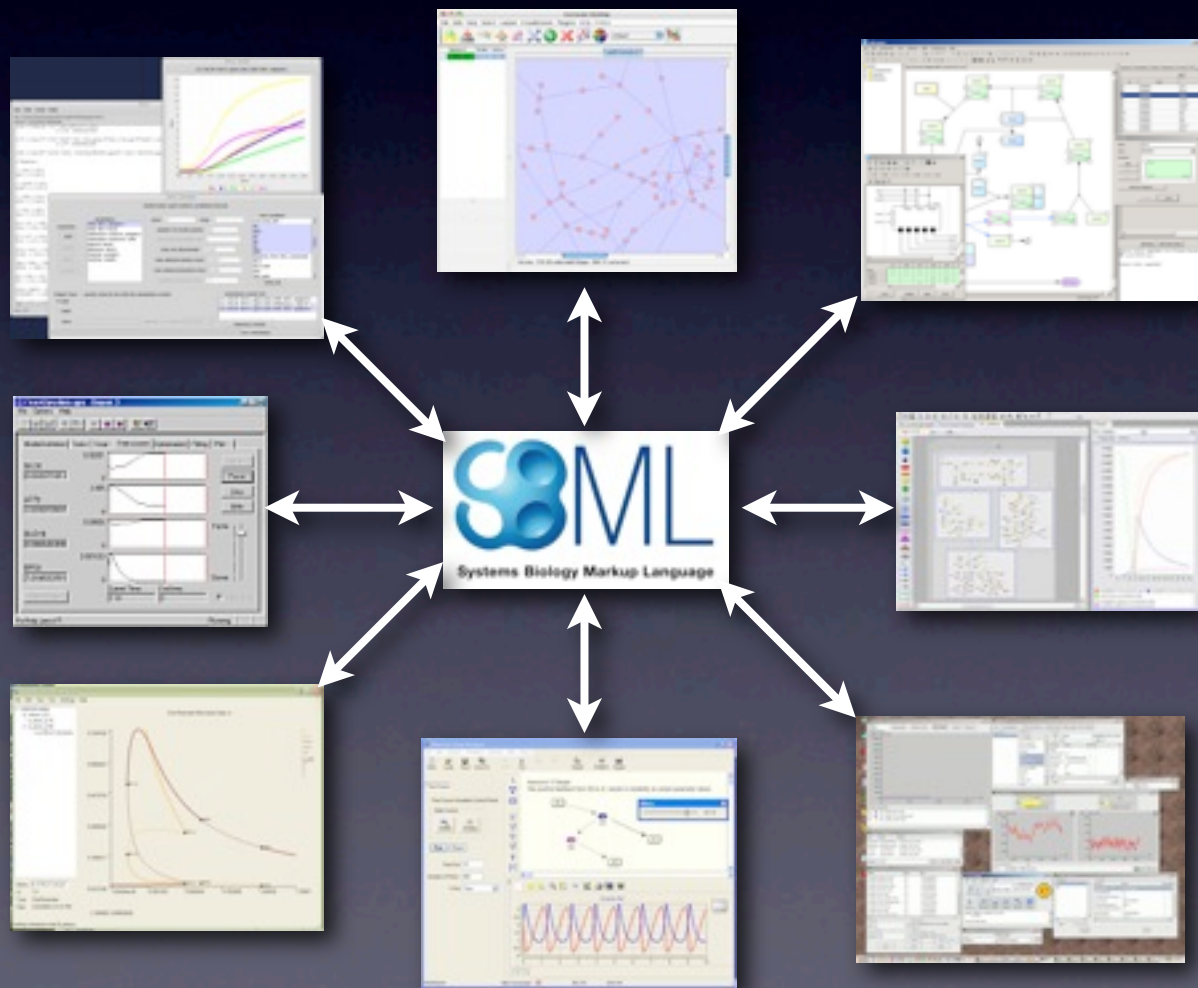
- * SUNDIALS (CVODE)
- * LSODA
- * GSL 等

使用している数値計算
ライブラリが明記されて
いれば大丈夫

モデルの再利用

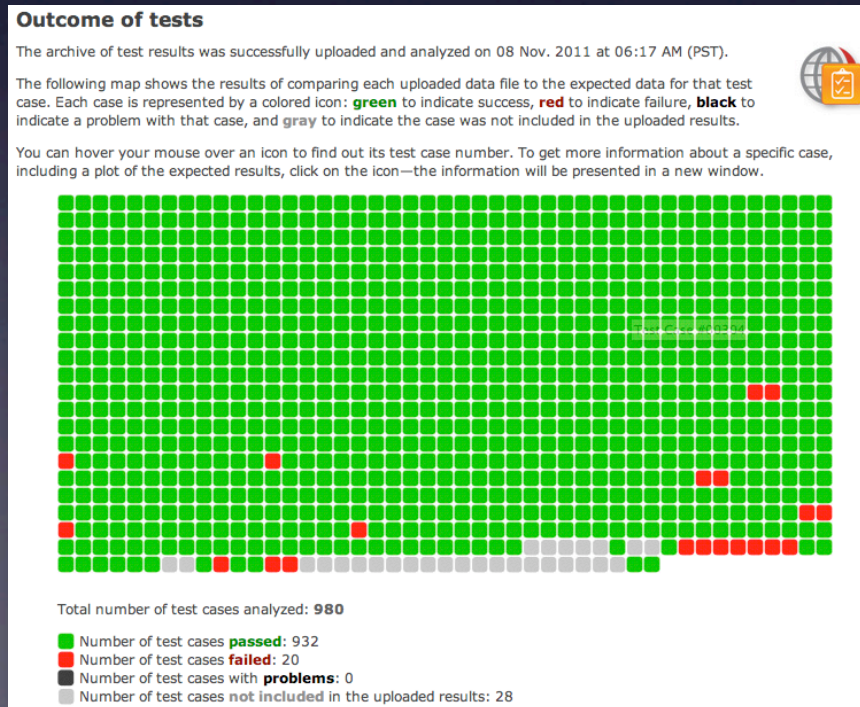
● 230以上のSBML対応ソフトウェア

● <http://sbml.org>



機能

- 常微分方程式の初期値問題を解くだけじゃない
- 代数微分方程式、イベント、遅延微分方程式、etc.
- SBMLのすべての仕様を満たす(テストをパスした)シミュレータは存在しない



SBML.org The Systems Biology Markup Language

News Documents Downloads Forums Facilities Community Events About

Parent page: SBML.org / Facilities

Online SBML Test Suite

The SBML Test Suite allows you to evaluate the degree and correctness of SBML support implemented in SBML-compatible software. The system supports specifications of SBML up through **Level 3 Version 1 Core**. This page is the interface to the online version of the Suite; it allows you to upload test results and have them evaluated by our server.

There are three steps to using the online interface:

1. **Select and download test cases.** You can download all cases or select a subset using the interface provided on our online test case selection page.
2. **Run simulations** of the models in the software package you are testing, and collect the results. How you run the cases is up to you and the software you are testing.
3. **Upload the simulation results.** This online service will compare them to the expected results and provide you with a report of the outcome.

Limitations: (1) This online service works only with the semantic portion of the SBML Test Suite. It does not evaluate the results of syntactic tests available with the full SBML Test Suite. (2) Only HTTP is supported as the protocol for uploading results; other protocols such as <https://>, <ftp://>, etc. are not supported.

This Online SBML Test Suite server-based software was written by Michael Hucka and Kimberly Begley. The test cases were developed by Sarah M. Keating, Lucian Smith, and Michael Hucka. Please visit the main page of the **SBML Test Suite** for more information and other resources.

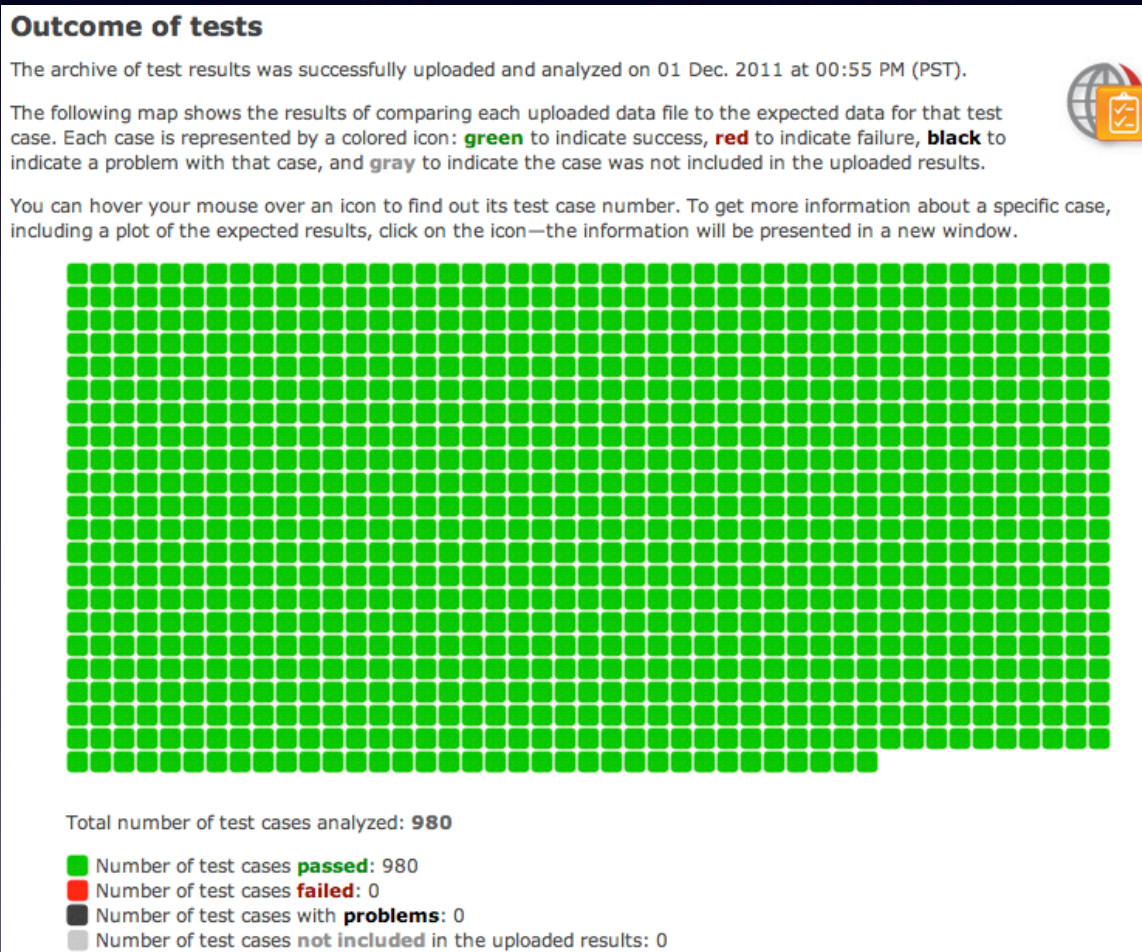
Please use our [issue tracking system](#) for any questions or suggestions about this website. This page was last modified 16:29, 6 June 2011.

newシミュレータ

- SBML Level 3 version 1 tests

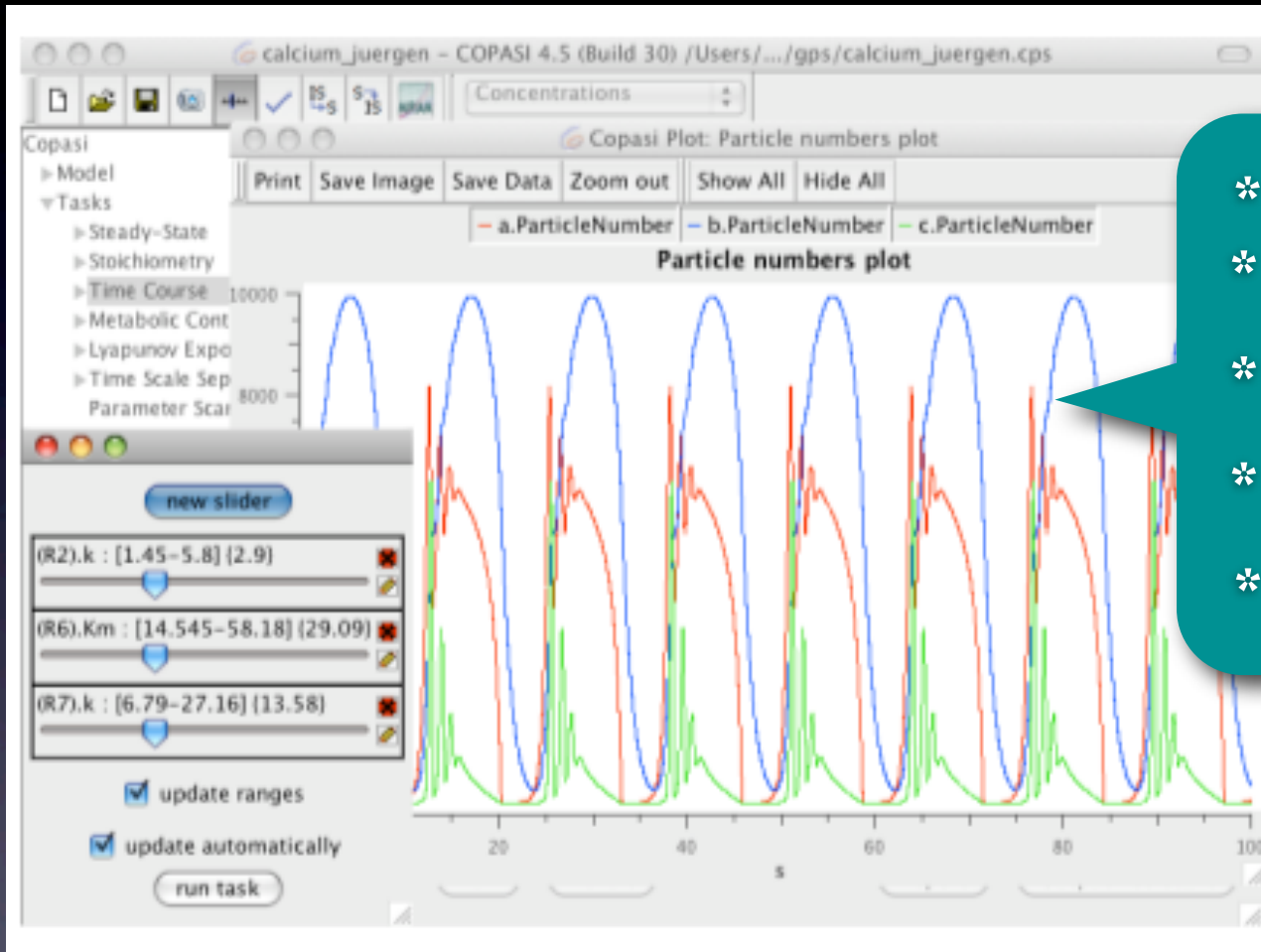
- 980 SBML tests (passed 100%)

http://sbml.org/Facilities/Online_SBML_Test_Suite



シミュレータの紹介

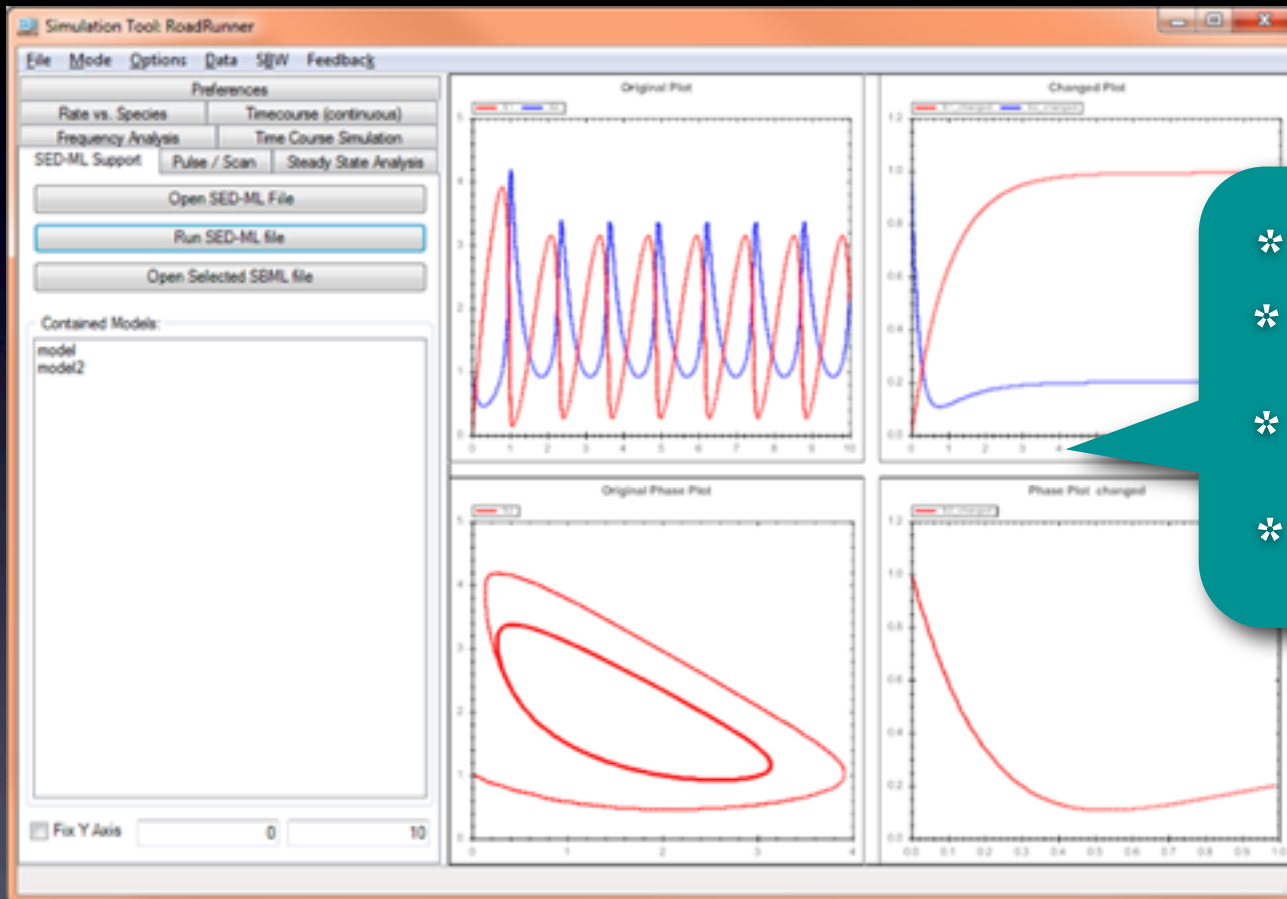
COPASI



- * ODE, SSA
- * LSODA (陰解法もOK)
- * パラメータフィッティング
- * 解析
- * C++

Univ. of Manchester, Univ. of Heidelberg, VBI
<http://copasi.org/>

RoadRunner



- * ODE
- * CVODE (陰解法もOK)
- * 解析ツールと連動
- * C# (.NET, Mono)

Univ. of Washington, Caltech
<http://sf.net/projects/roadrunner>

PhysioDesigner

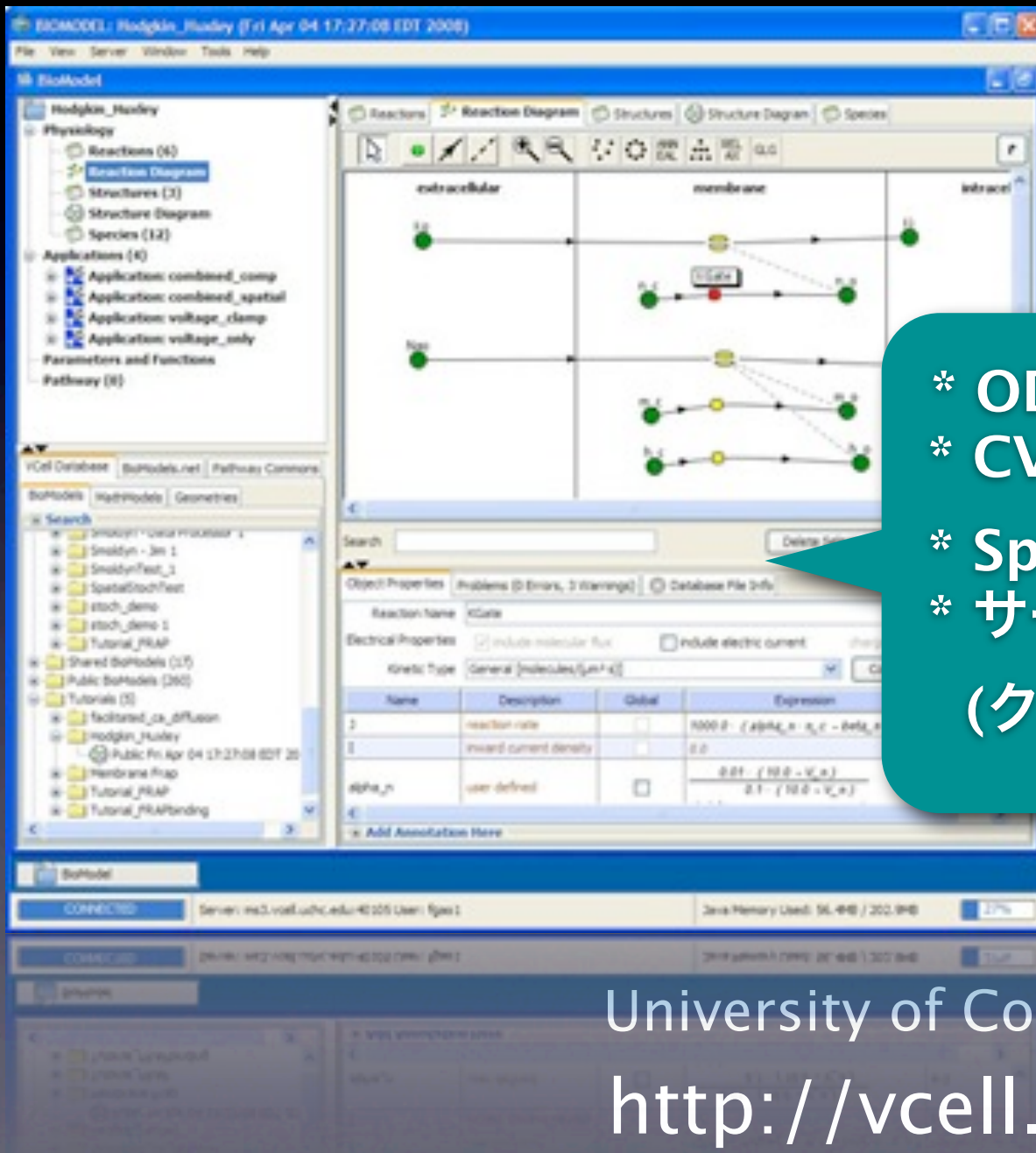
The screenshot displays the PhysioDesigner software interface. It features several panels: a Component Tree on the left, a Tree Diagram in the center, a Physical-Quantity Table and Port Table at the bottom left, and a Nesting Diagram at the bottom center. A simulation window in the bottom right shows a plot of voltage (V) over time, with a yellow bar indicating the simulation period and a purple line showing the resulting signal.

- * ODE, PDE(FEM)
- * モジュール構造
- * C++, Java

Osaka University, OIST

<http://physiodesigner.org/>

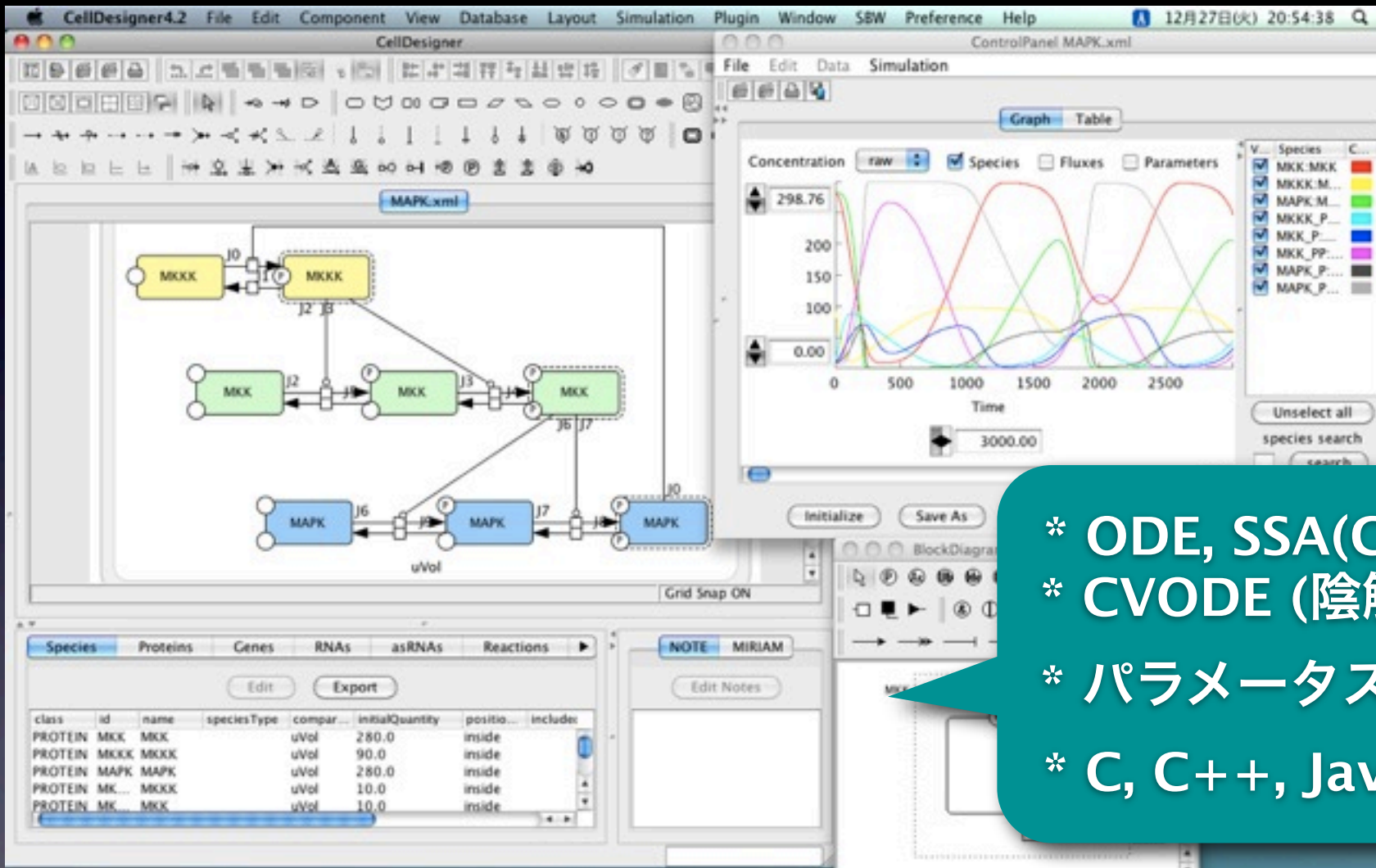
VCell



- * ODE, SSA, PDE, DAE
- * CVODE/IDA (陰解法もOK)
- * Spatial Stochastic (Smoldyn)
- * サーバサイド
(クライアント版もリリース)

University of Connecticut Health Center
<http://vcell.org/>

CellDesigner

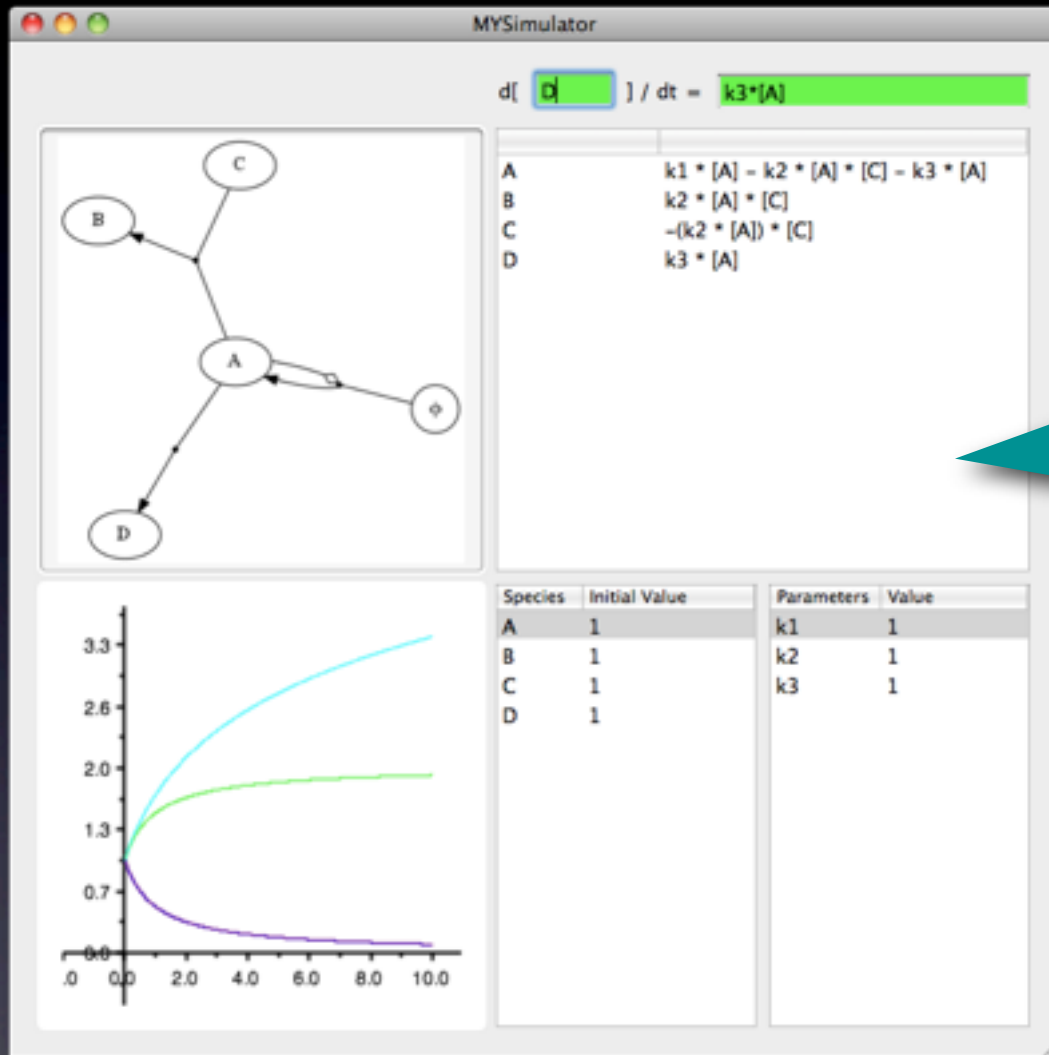


- * ODE, SSA(COPASI)
- * CVODE (陰解法もOK)
- * パラメータスキャン
- * C, C++, Java

JST, SBI, Keio University

<http://celldesigner.org/>

最速モデル構築



- * ODE
- * 陰解法もOK
- * 微分方程式 → ネットワーク
- * C++, Objective-C

Keio University
Coming soon!

数式处理

● Maxima, wxMaxima

● Wolfram Alpha

wxMaxima interface showing the following commands and results:

- (%i4) 'diff(y(x),x,2) + 'diff(y(x),x) + y(x) = 0;
- (%o4) $\frac{d^2}{dx^2}y(x) + \frac{d}{dx}y(x) + y(x) = 0$
- (%i5) desolve('diff(y(x),x,2) + 'diff(y(x),x) + y(x) = 0, y(x));
- (%o5) $y(x) = %e^{-\frac{x}{2}} \left(\frac{\sin\left(\frac{\sqrt{3}x}{2}\right) \left(2 \left(\frac{d}{dx}y(x) \Big|_{x=0} + y(0) \right) - y(0) \right)}{\sqrt{3}} + y(0) \cos\left(\frac{\sqrt{3}x}{2}\right) \right)$
- (%i6) x^k;
- (%o6) x^k
- (%i7) diff(x^k, x);
- (%o7) kx^{k-1}
- (%i8) diff(x^k, x, 2);
- (%o8) $(k-1)kx^{k-2}$
- (%i9) integrate(x^k, x);
- Is k+1 zero or nonzero?n;
- (%o9) $\frac{x^{k+1}}{k+1}$
- (%i10) integrate(sin(x), x);
- (%o10) $-\cos(x)$

Zoom set to 110%

WolframAlpha interface showing the solution to the differential equation $y'' + y = 0$.

Input: $y''(x) + y(x) = 0$

ODE classification: second-order linear ordinary differential equation

Alternate form: $y''(x) = -y(x)$

Differential equation solution: $y(x) = c_2 \sin(x) + c_1 \cos(x)$

Plots of sample individual solutions:

- Plot 1: $y(0) = 1, y'(0) = 0$. Shows a cosine wave.
- Plot 2: $y(0) = 0, y'(0) = 1$. Shows a sine wave.

Sample solution family: (sampling $y(0)$ and $y'(0)$)

Possible Lagrangian: $\mathcal{L}(y', y) = \frac{1}{2} (y'^2 - y^2)$

謝辞

- 東京大学 生産技術研究所 小林徹也 准教授
- 慶應義塾大学 理工学部 広井賀子 助教

松井達広
偏微分方程式

中村和成
陰解法
高速化(GPU)

瀧沢大夢
陰解法、イベント処理、
遅延・代数微分方程式

田平章人
構文解析、陰解法
高速化(CPU)

