定量生物に効く数値計算

2012年11月23日

慶應義塾大学 舟橋 啓

Keio University
1858
CALAMVS
GLADIO
FORTIOR

Windowsの人

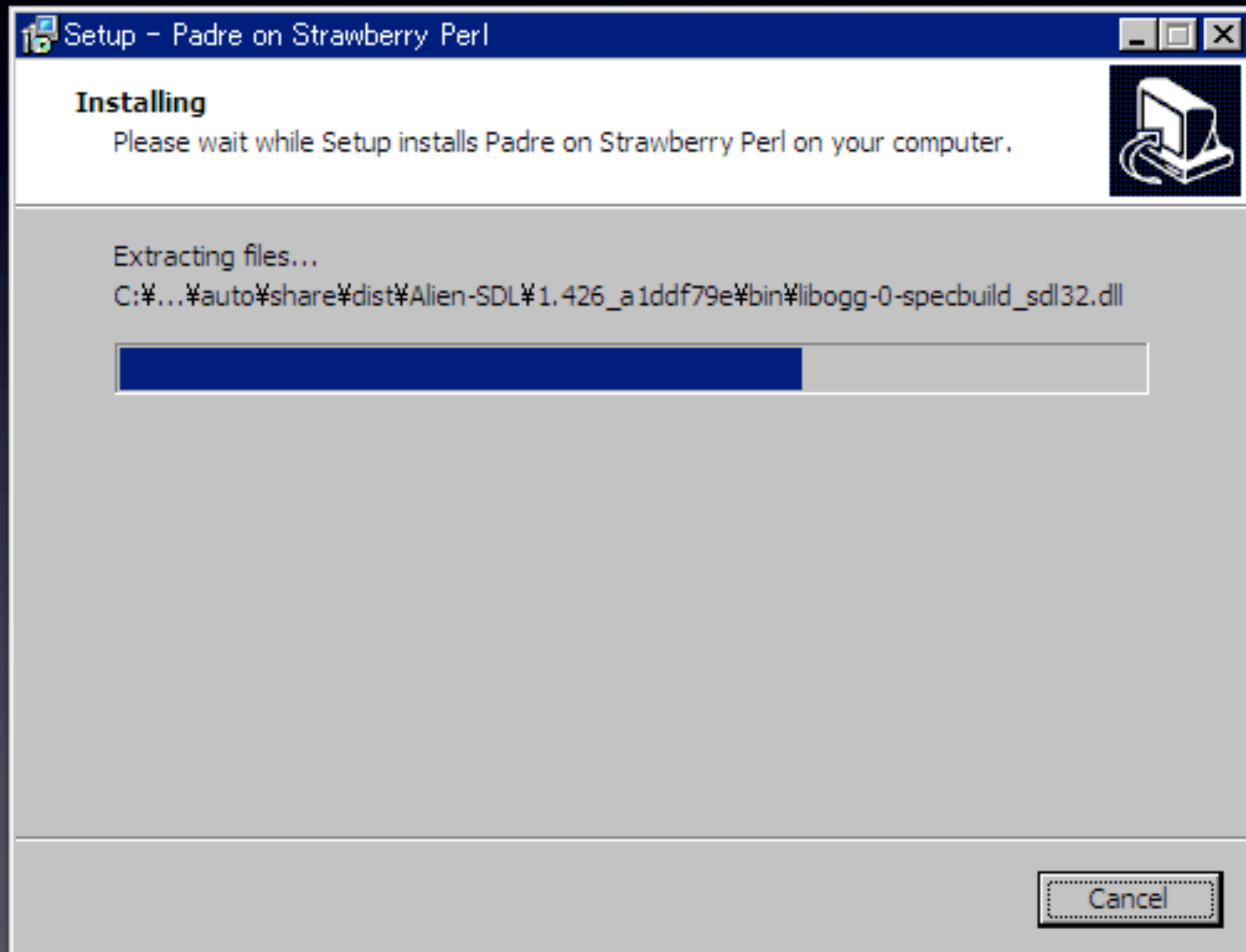
http://padre.perlide.org

Padreをダウンロード

http://padre.perlide.org

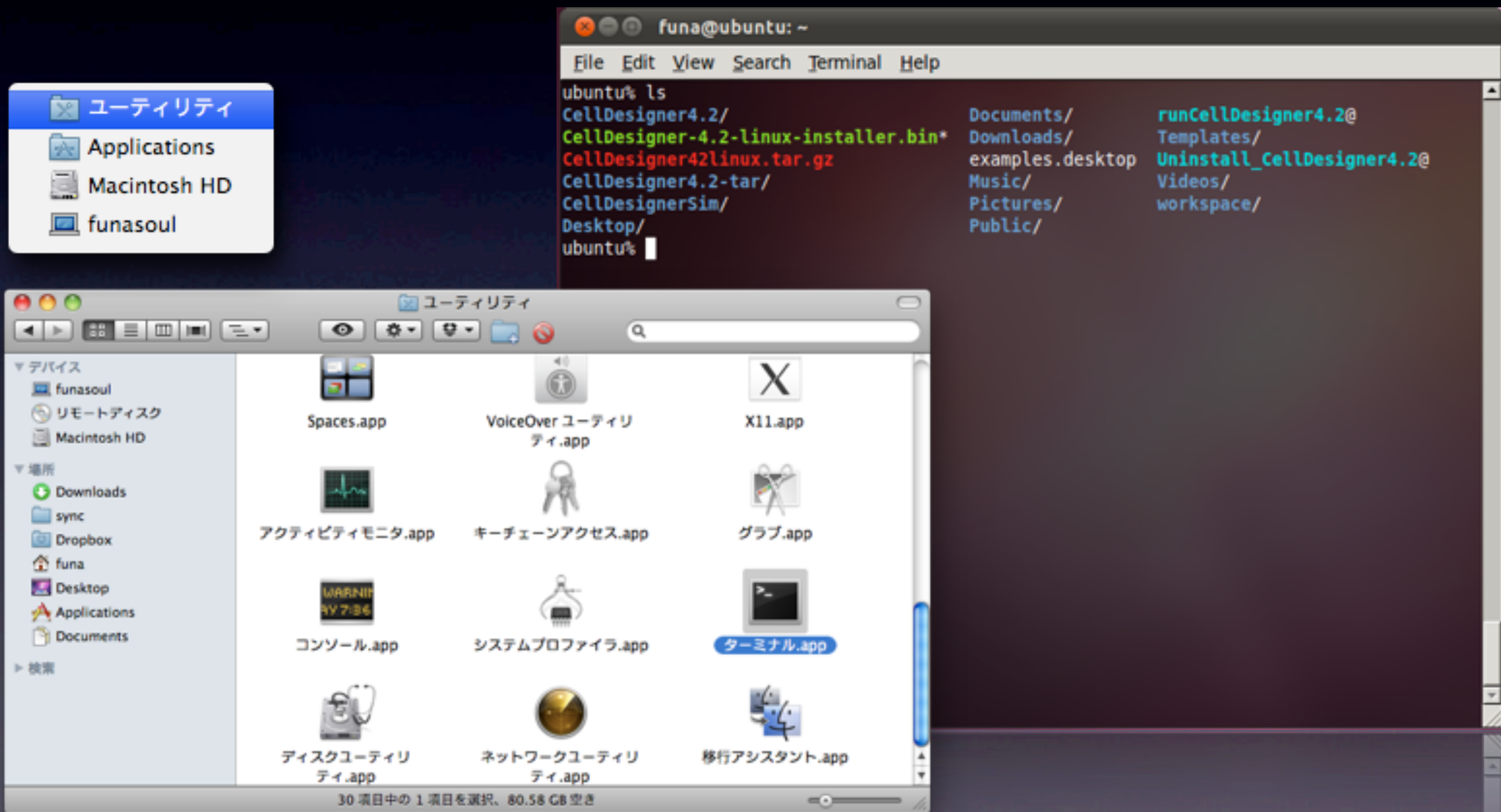
PadreをinstallІ

http://padre.perlide.org



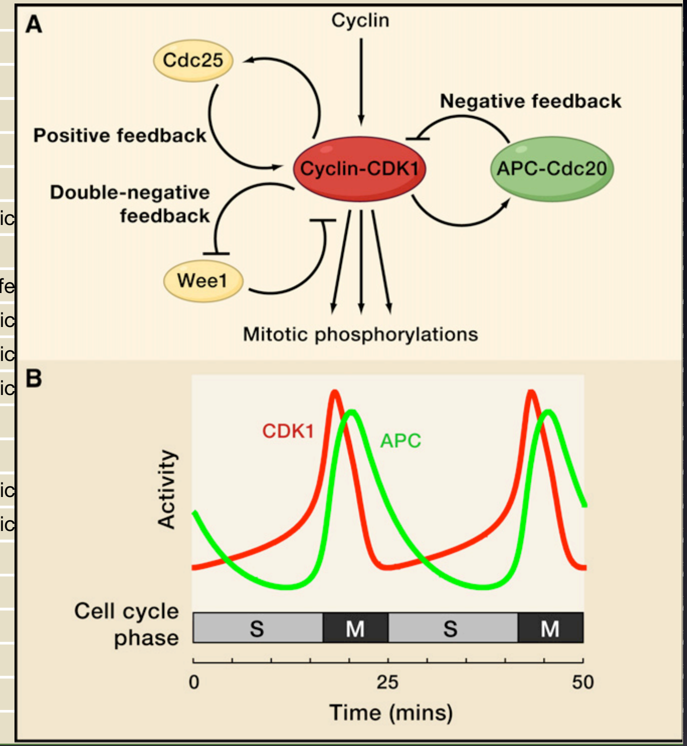
Mac, Linuxの人

Terminalを起動



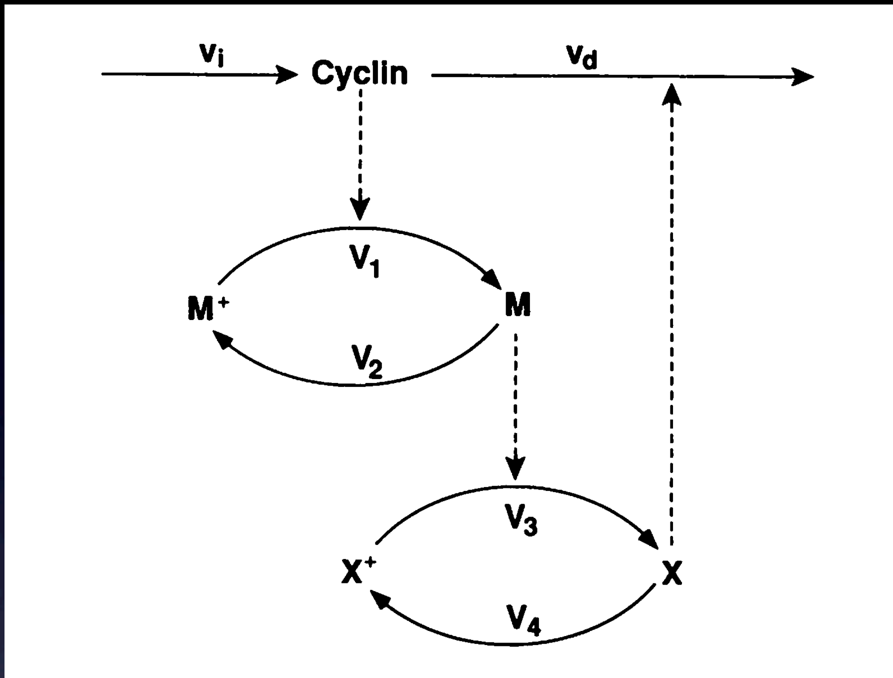
数理モデルを扱った論文

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



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

数理モデルを扱った論文



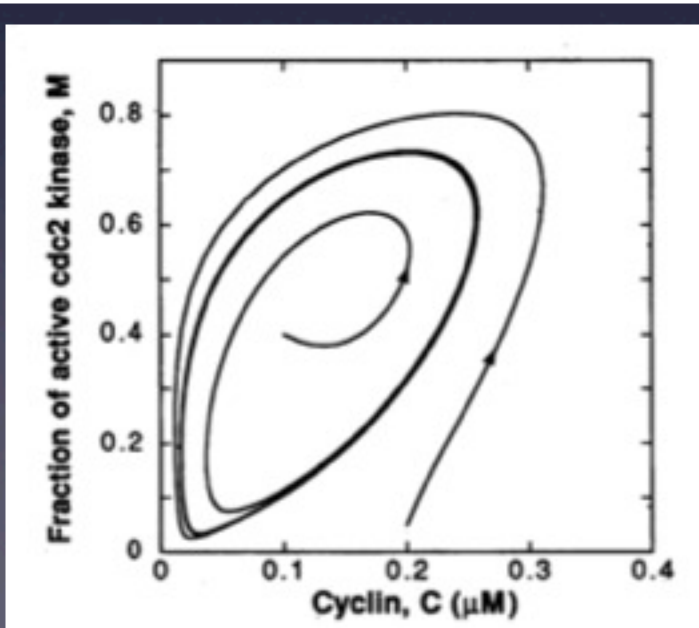
$$\frac{dC}{dt} = v_i - v_d X \frac{C}{K_d + C} - k_d C$$

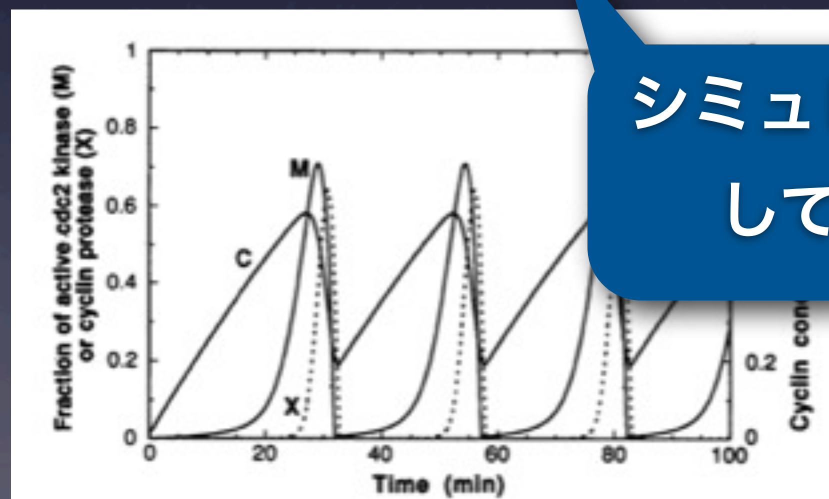
$$\frac{dM}{dt} = V_1 \frac{(1-M)}{K_1 + (1-M)} - V_2 \frac{M}{K_2 + M},$$

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

with

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

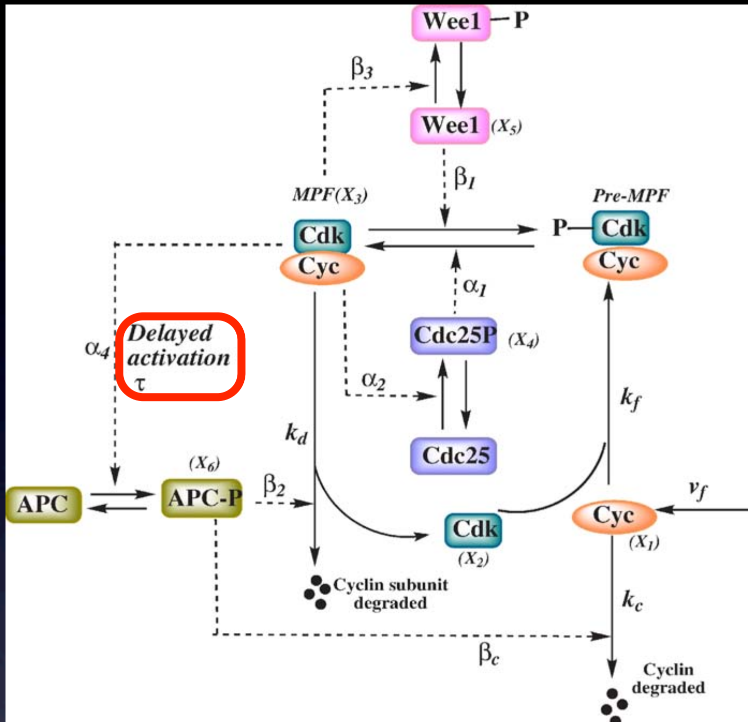




シミュレーション
してみたい!

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

数理モデルを扱った論文



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

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

$$\frac{dX_3}{dt} = v_{M1}\left[1 + \alpha_1 \frac{X_4}{K_a + X_4}\right] \frac{(C - X_2 - X \cdots}{J_1 + (C - X \cdots X_3)} \cdots \left[\cdots \frac{\cdots}{K_a + X_5}\right] \frac{\cdots}{J'_1 + X_3} \cdots X_6]X$$

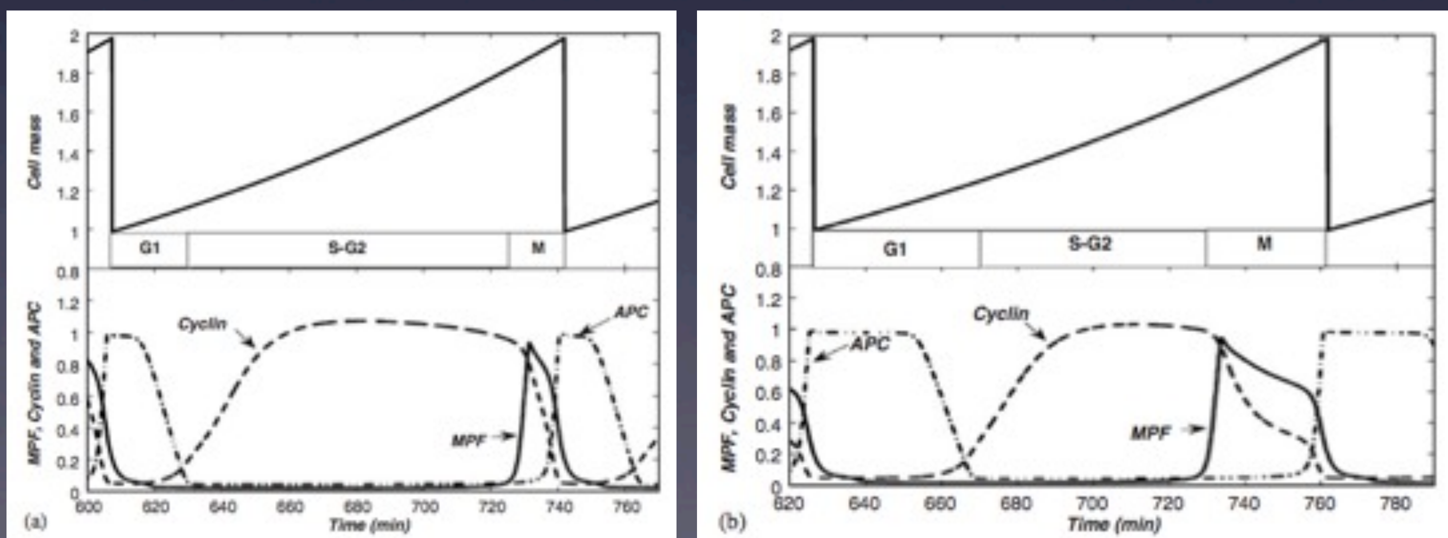
$$\frac{dX_4}{dt} = v_{M2}\left[1 + \alpha_2 \frac{mX_3}{K_a + mX_3}\right] \frac{(1 - X_4)}{J_2 + (1 - X_4)} - v'_{M2} \frac{X_4}{J'_2 + X_4}$$

$$\frac{dX_5}{dt} = v_{M3} \frac{(1 - X_5)}{J_3 + (1 - X_5)} - v'_{M3}\left[1 + \beta_3 \frac{mX_3}{K_a + mX_3}\right] \frac{X_5}{J'_3 + X_5}$$

$$\frac{dX_6}{dt} = v_{M4}\left[1 + \alpha_4 \frac{mX_3(t - \tau)}{K_a + mX_3(t - \tau)}\right] \frac{(1 - X_6)}{J_4 + (1 - X_6)} - v'_{M4} \frac{X_6}{J'_4 + X_6}$$

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

シミュレーション
してみたい!



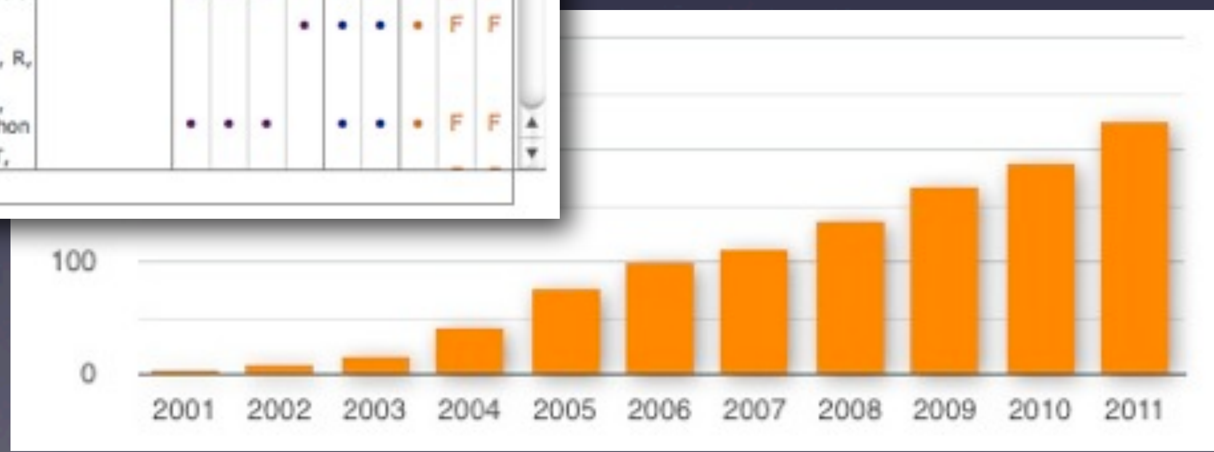
J Theor Biol, 2006, 241(3): 617-27

シミュレータの現状

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

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!

どれを選べばいい?

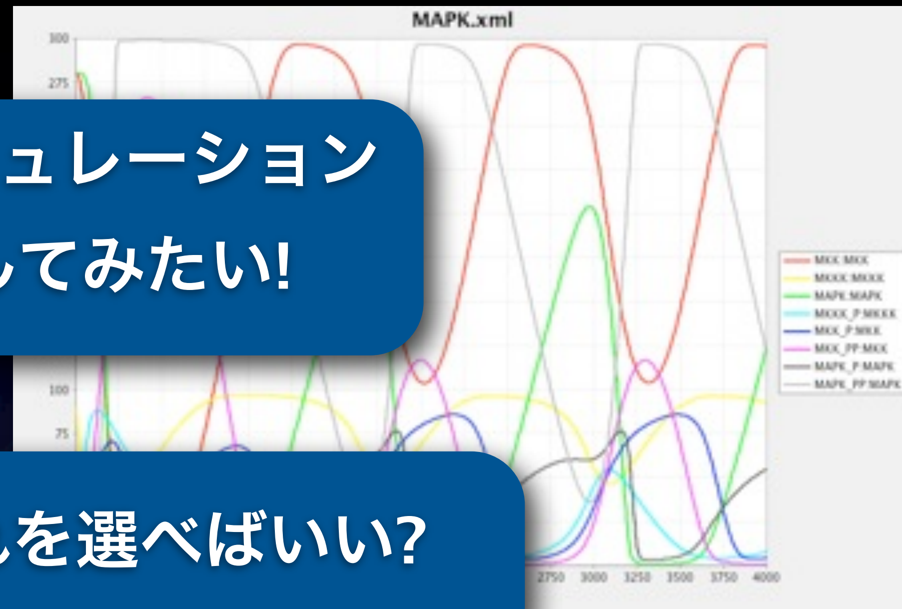
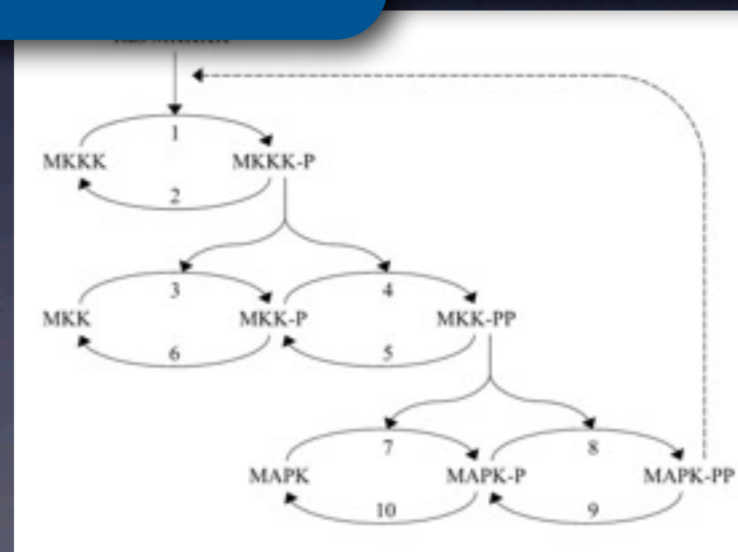


Table 1. Kinetic equations comprising the computation

Reaction number	Rate equation
1	$V_1 \cdot [\text{MKKK}]/((1 + ([\text{MAPK-}PP]/K_I)^n) \cdot (K_1 + [\text{MKKK}]))$
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: $[\text{MKKK}]_{\text{total}} = 100$; $[\text{MKK}]_{\text{total}} = 300$; $[\text{MAPK}]_{\text{total}} = 300$

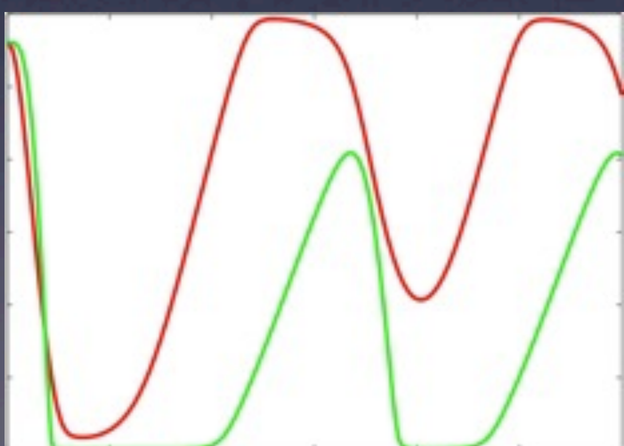
$$[\text{MKK}]_{\text{total}} = [\text{MKK}] + [\text{MKK-}P] + [\text{MKK-}PP]$$
$$[\text{MAPK}]_{\text{total}} = [\text{MAPK}] + [\text{MAPK-}P] + [\text{MAPK-}PP]$$



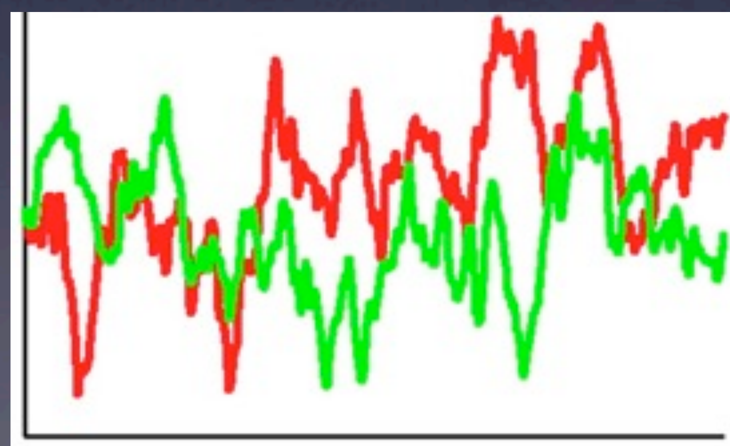
シミュレーション

- 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)

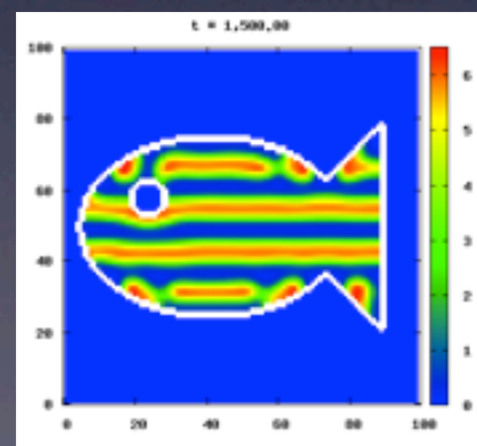
ODE



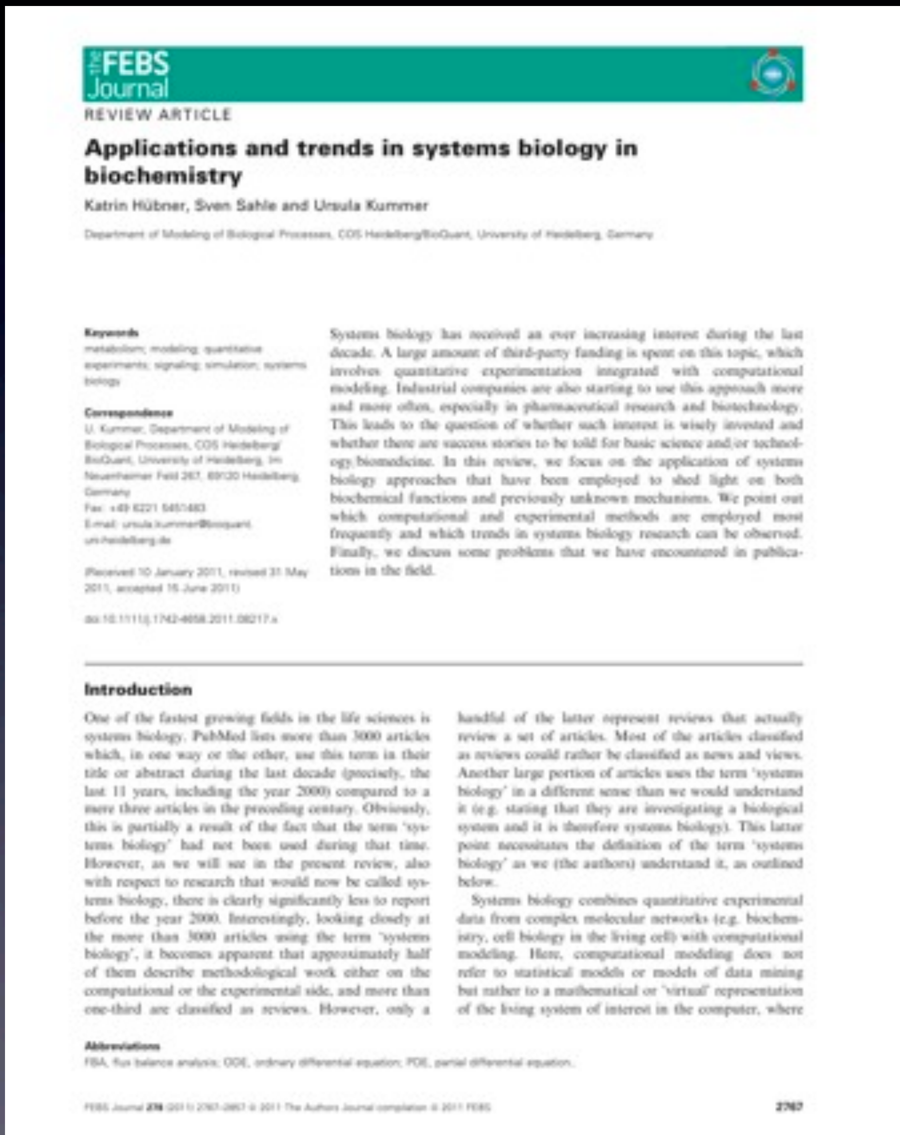
SSA



PDE



ODE? SSA? PDE?



the FEBS Journal

REVIEW ARTICLE

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

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.

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 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.

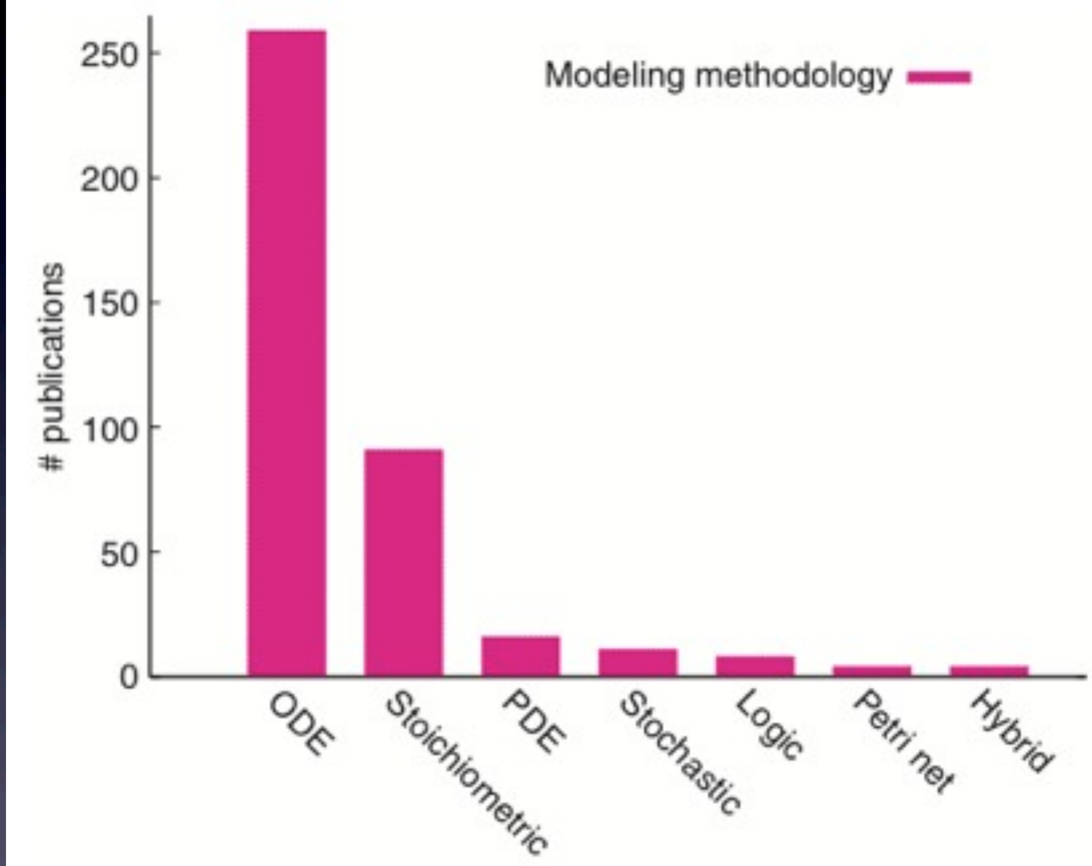


Fig. 7. Number of publications describing systems biology applied to biochemistry in the years 2000–2010 using a specific computational modeling approach.

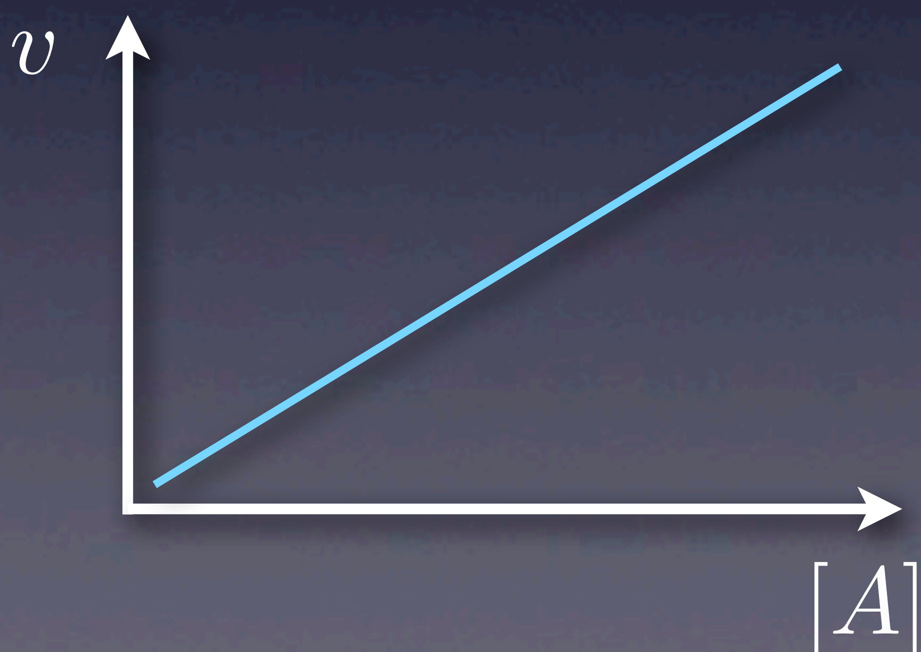
FEBS Journal 278 (2011) 2767–2857

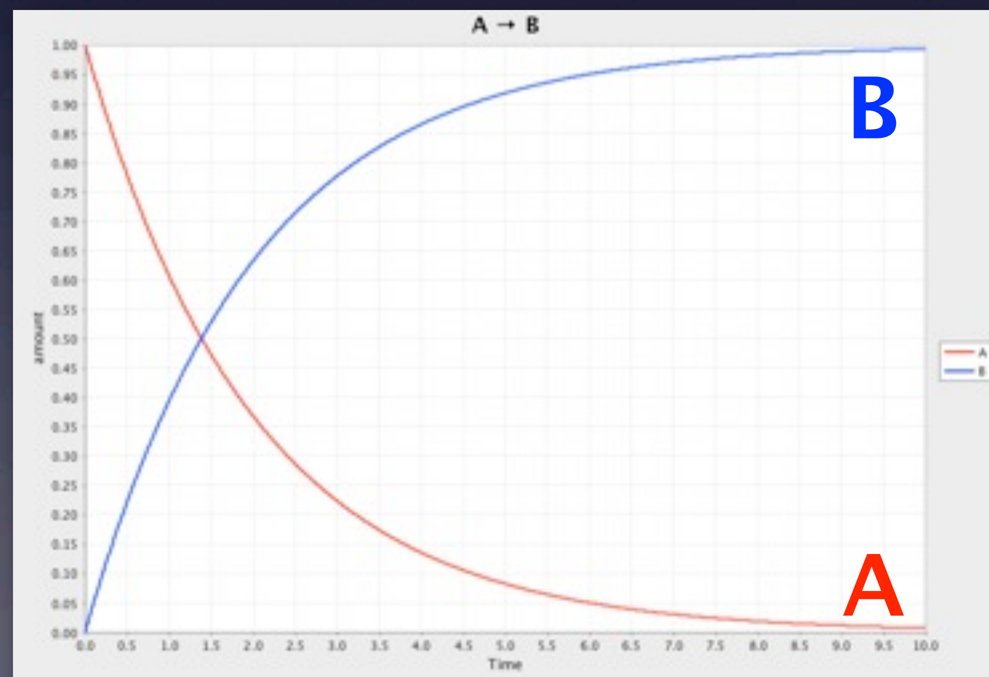
ODEで書かれた反応方程式

$$v = k[A]$$

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

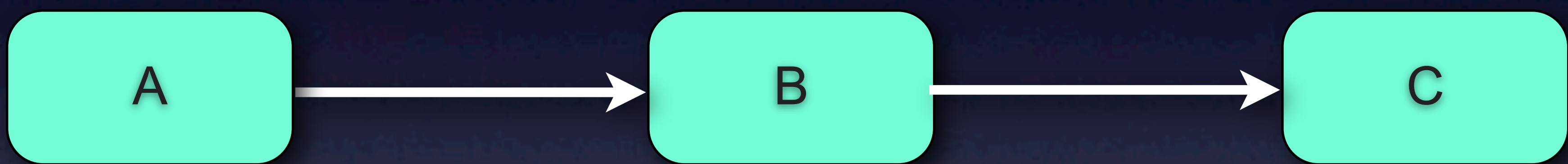
$$\frac{d[B]}{dt} = k[A]$$





Mass-action

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



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

大事なこと

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

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

- 和、差で反応毎に分解できる

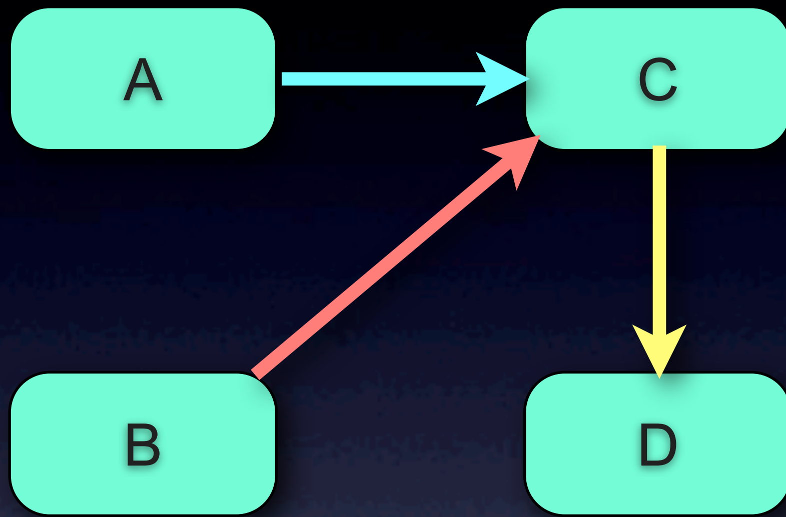
ODE ↔ ネットワーク

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

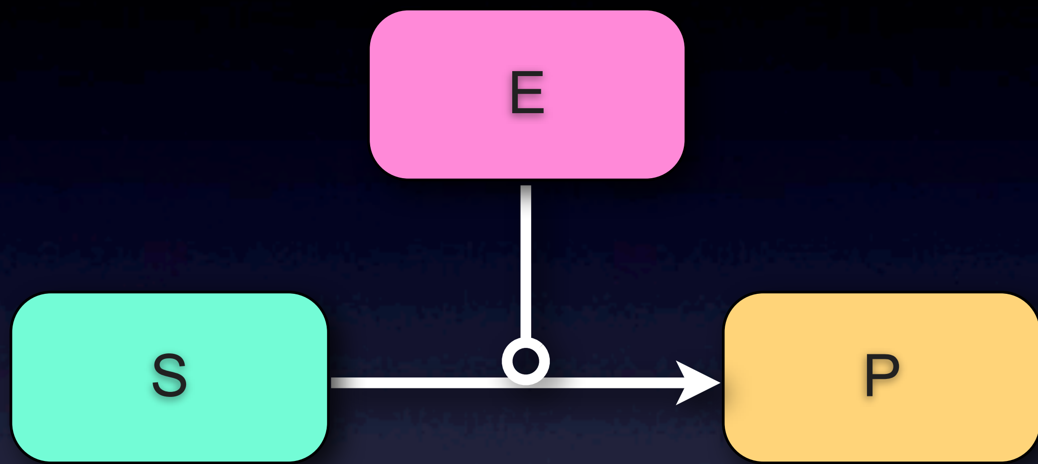
$$\frac{dB}{dt} = -k_2 B$$

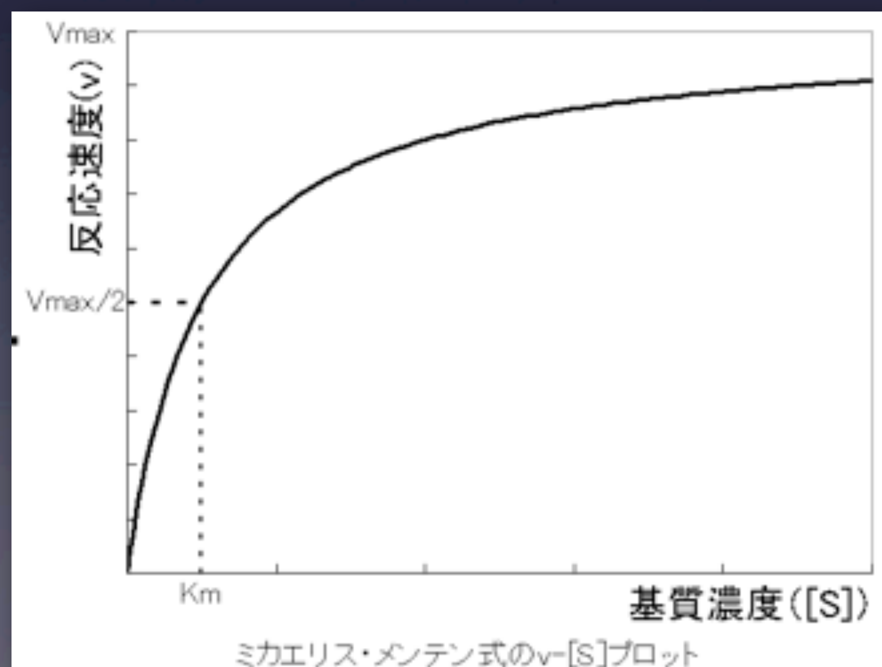
$$\frac{dC}{dt} = k_1 A + k_2 B - k_3 C$$

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



Michaelis-Menten





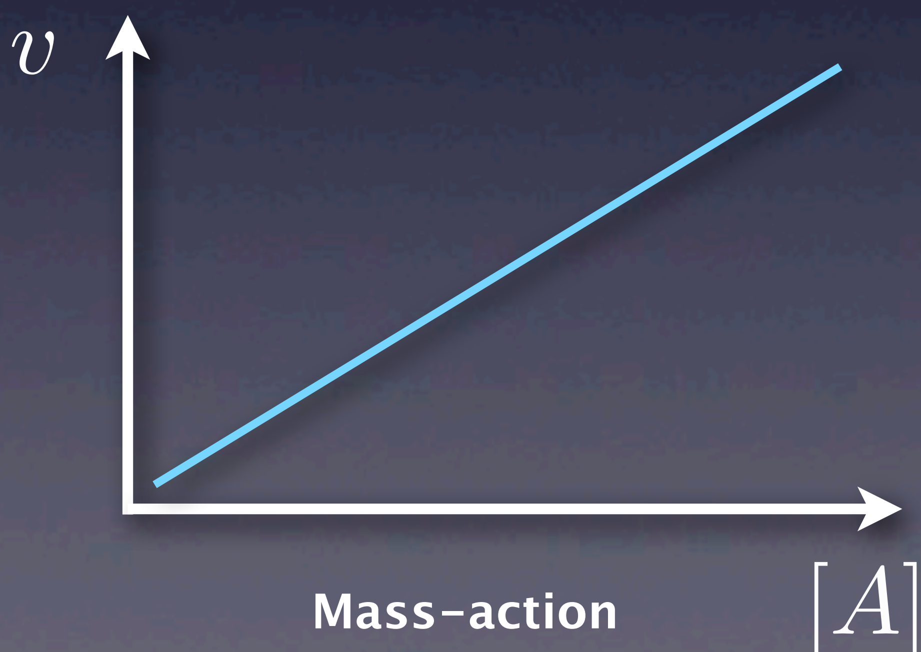
$$\frac{d[P]}{dt} = \frac{V_{max}[S]}{K_m + [S]}$$

Mass-action

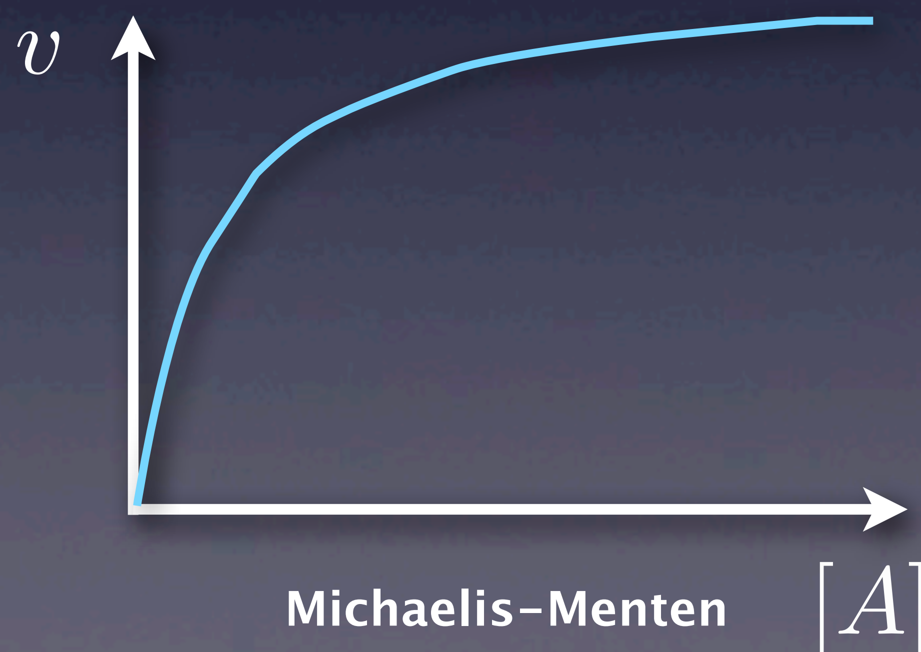
$$v = k[A]$$

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

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

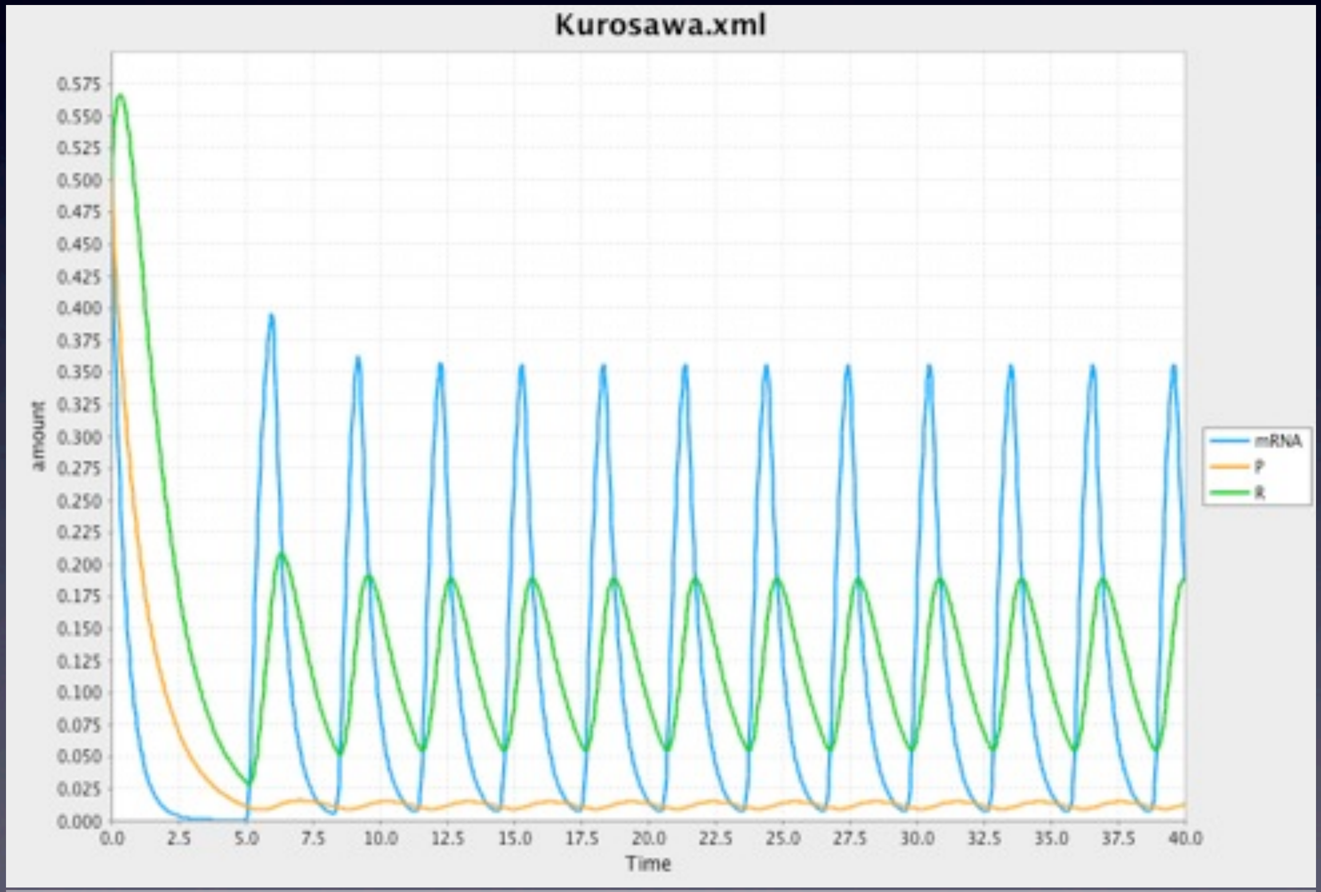


Mass-action



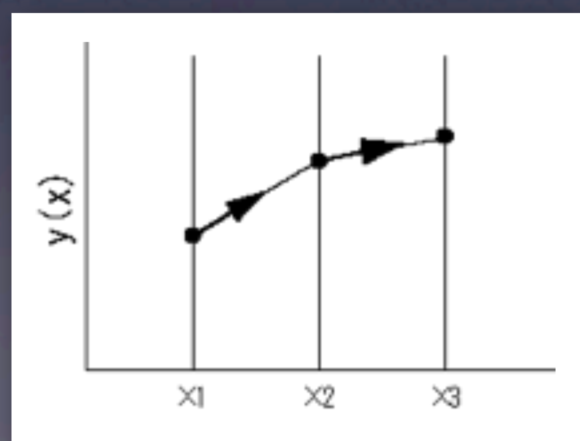
Michaelis-Menten

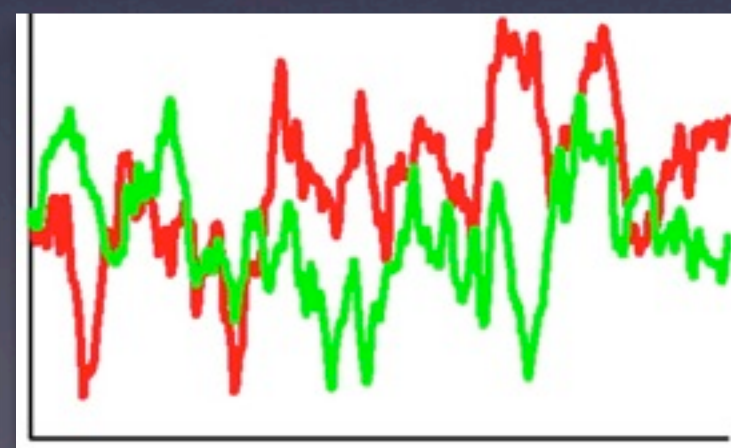
ODEシミュレータの実装



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





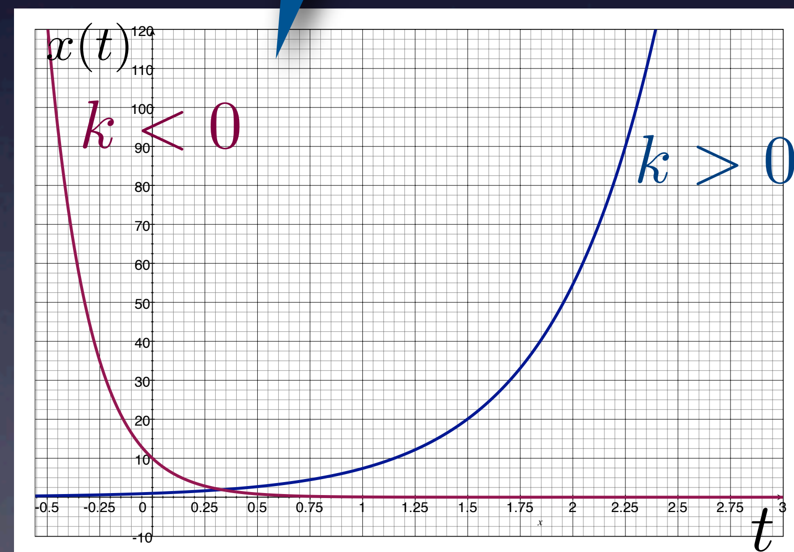
未来の予測

- 未来 (ある時間: t) の状態を予測

状態1 → 状態2 → 状態3

解析的に未来を予測

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

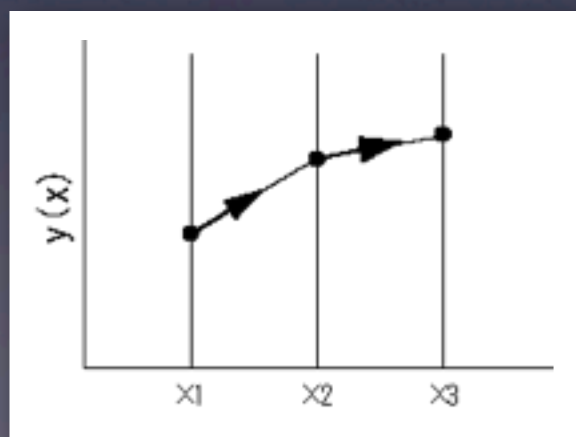


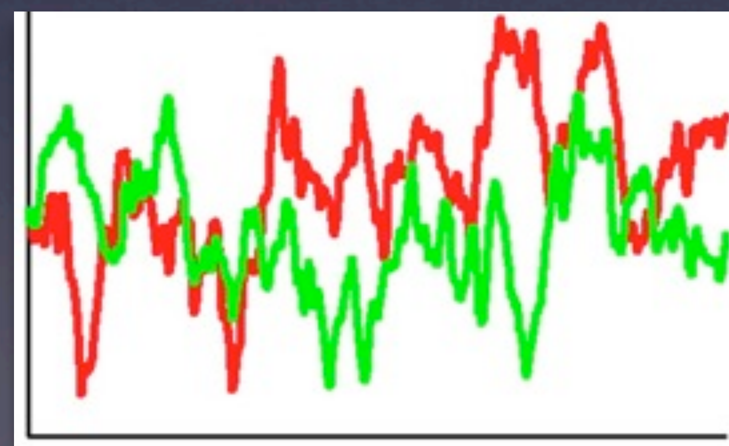
$$x = C_1 e^{kt}$$

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

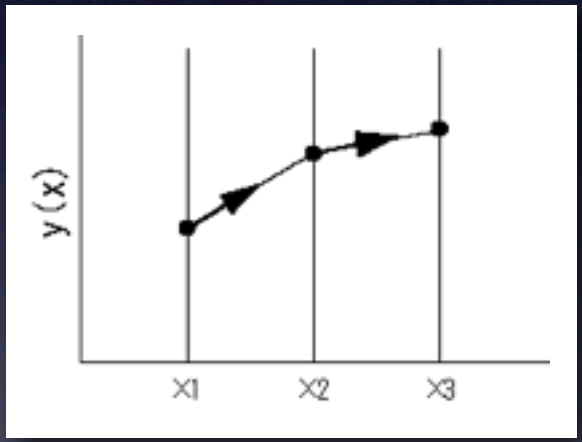




数値計算が必要な場合

- 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

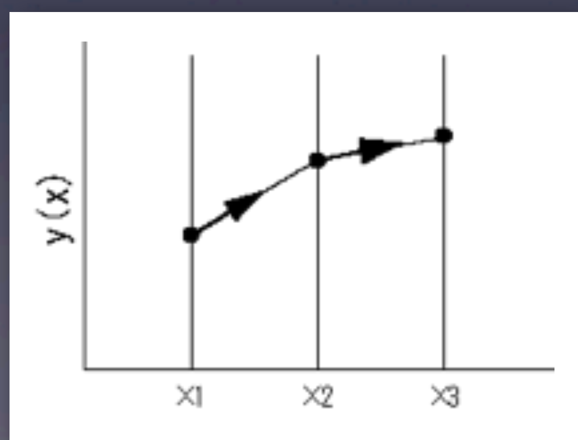
ODEシミュレータは
数値積分を行なっている



ODEの数値積分

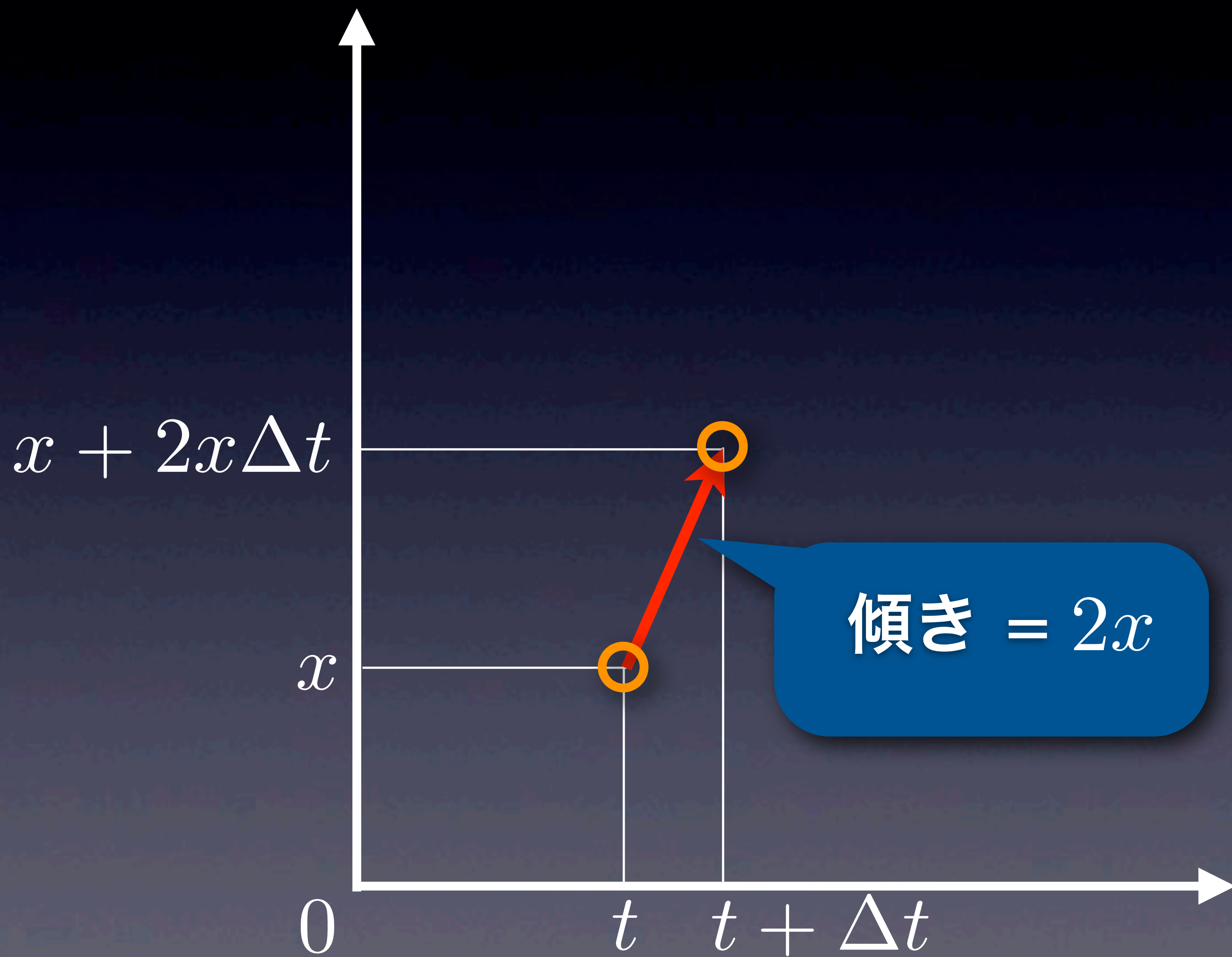
$$\frac{dx}{dt} = f(t, x)$$

$$x_0 \xrightarrow{f(t,x)} x_1 \xrightarrow{f(t,x)} x_2 \longrightarrow \cdots \xrightarrow{f(t,x)} x_t$$



ODE Simulatorの作り方

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

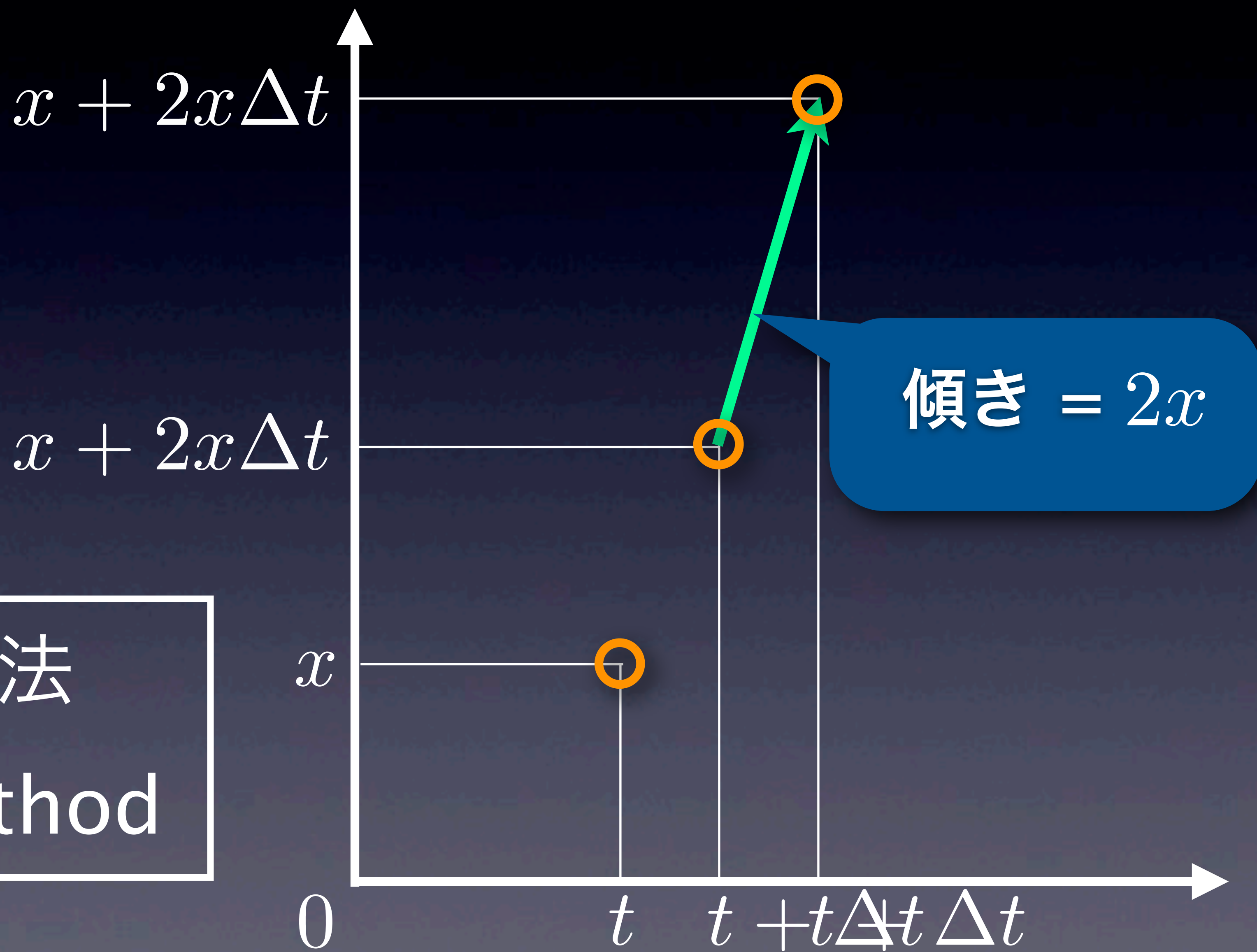


ODE Simulatorの作り方

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

オイラー法

Euler's Method

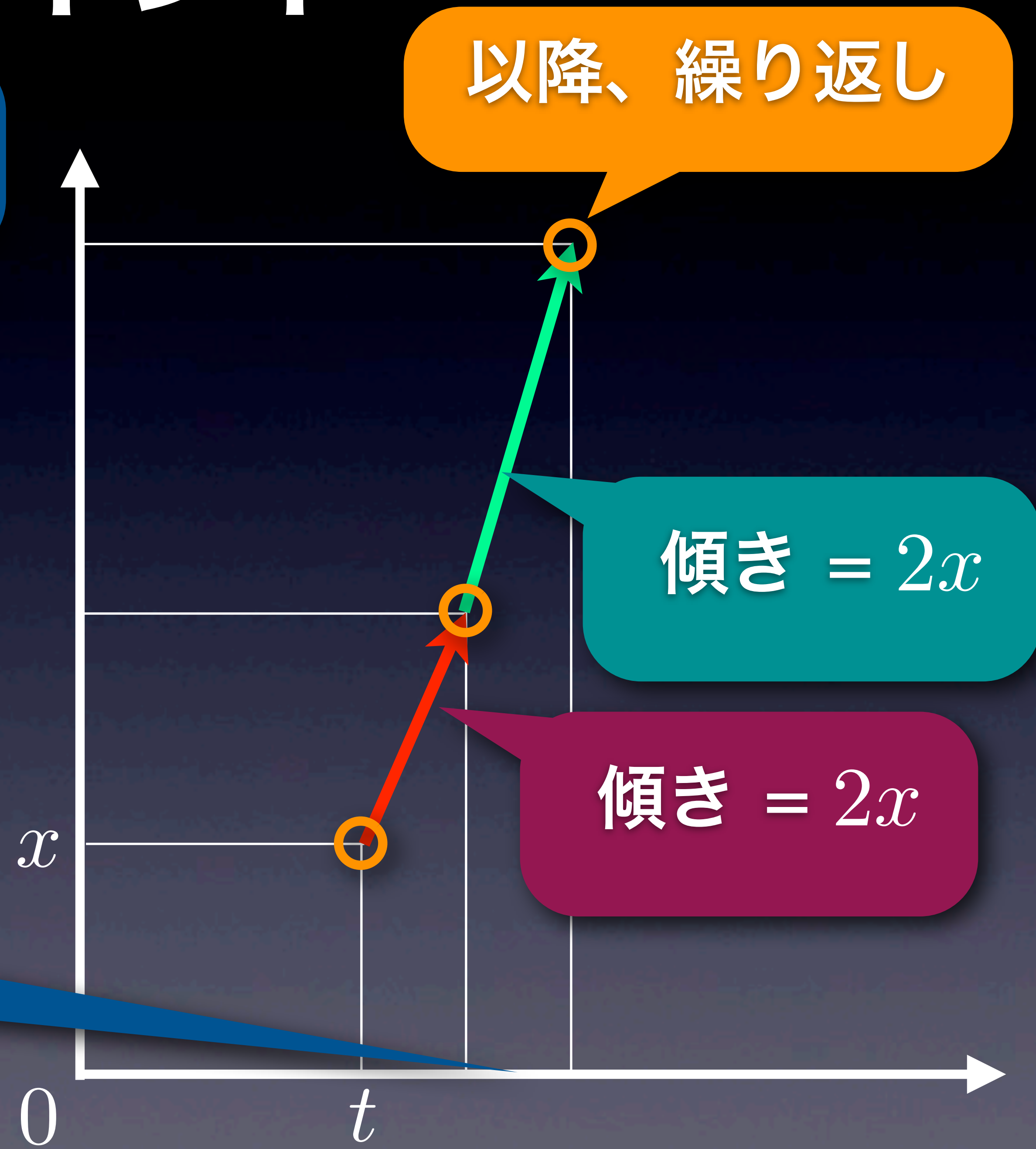


ポイント

x は dx に応じて変化

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

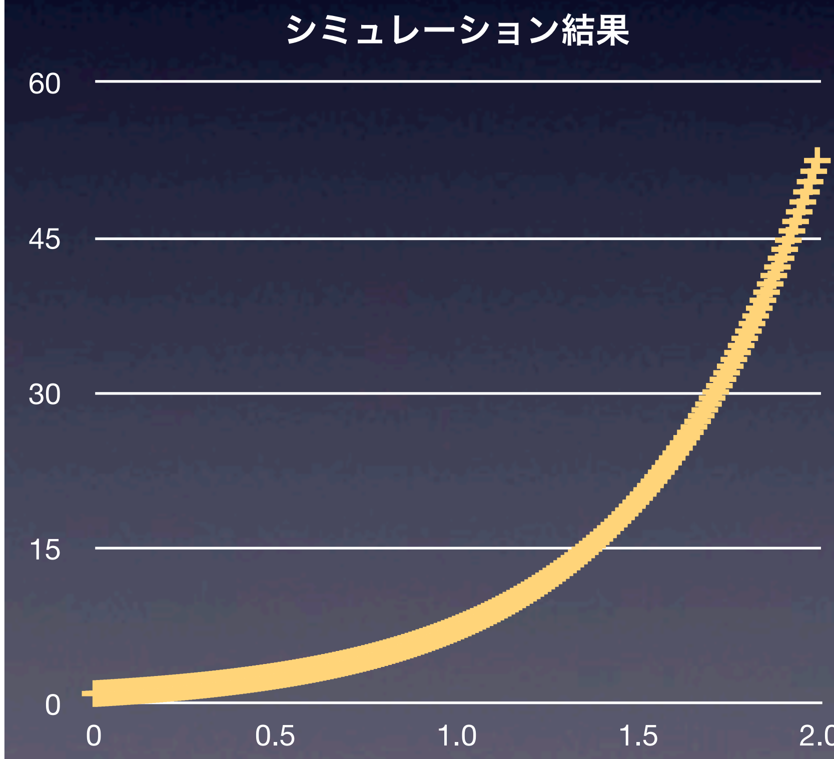
t は Δt で増加



ODE Simulator

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

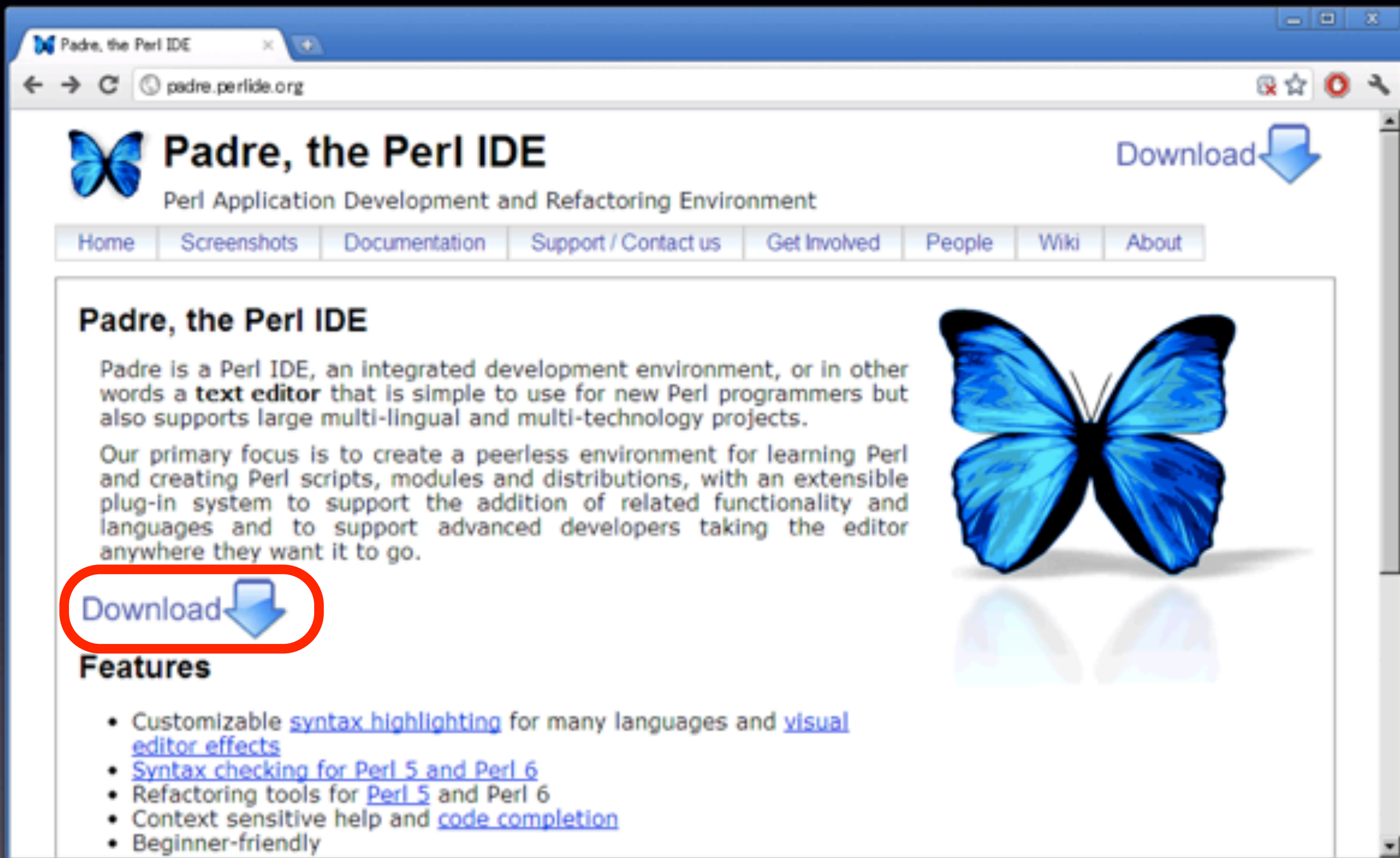
```
#!/usr/bin/perl

$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;
}
```



PerlでODEシミュレータ

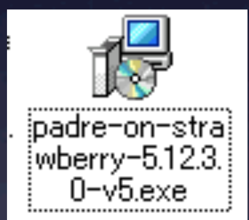
Windowsの人

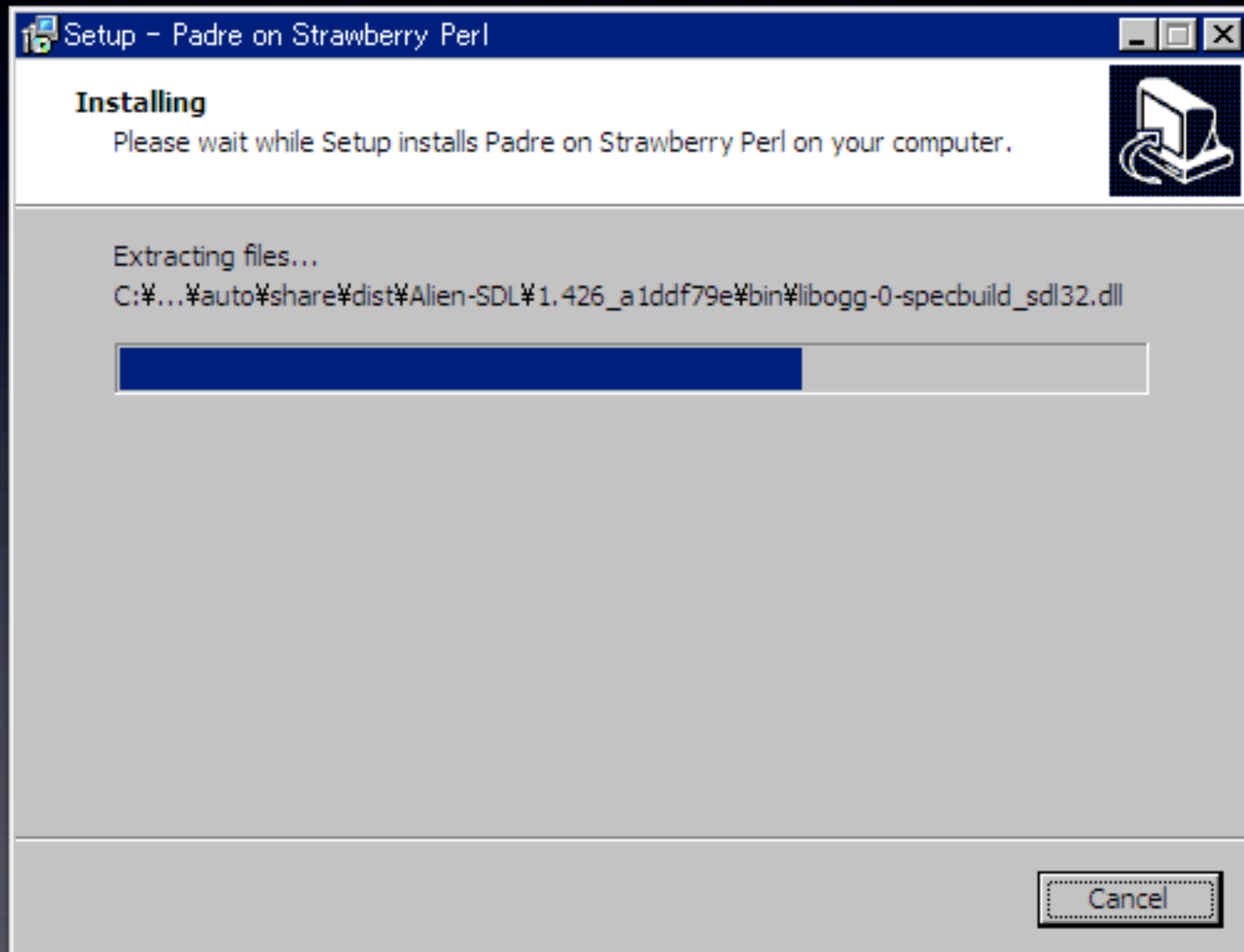
http://padre.perlide.org



PadreをInstall

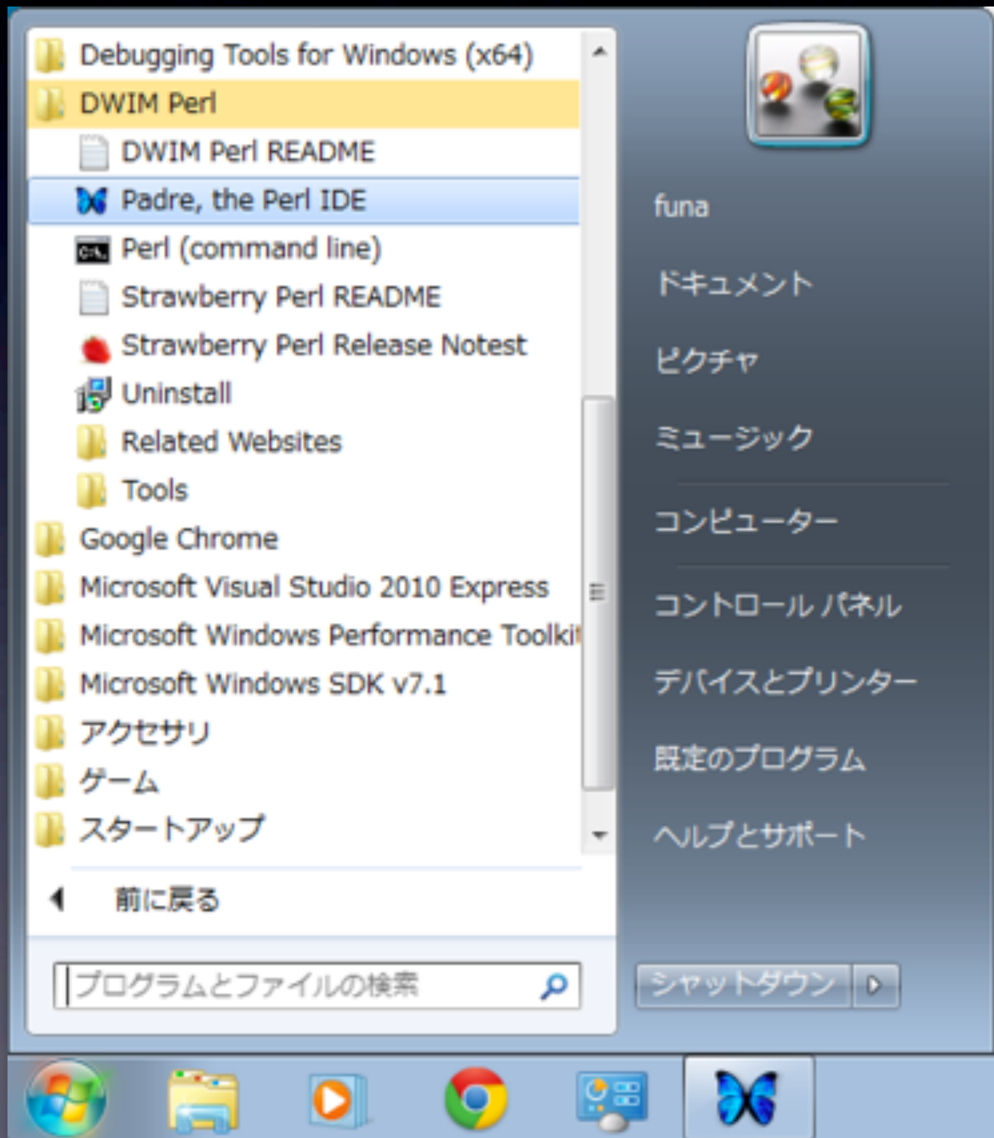
http://padre.perlide.org



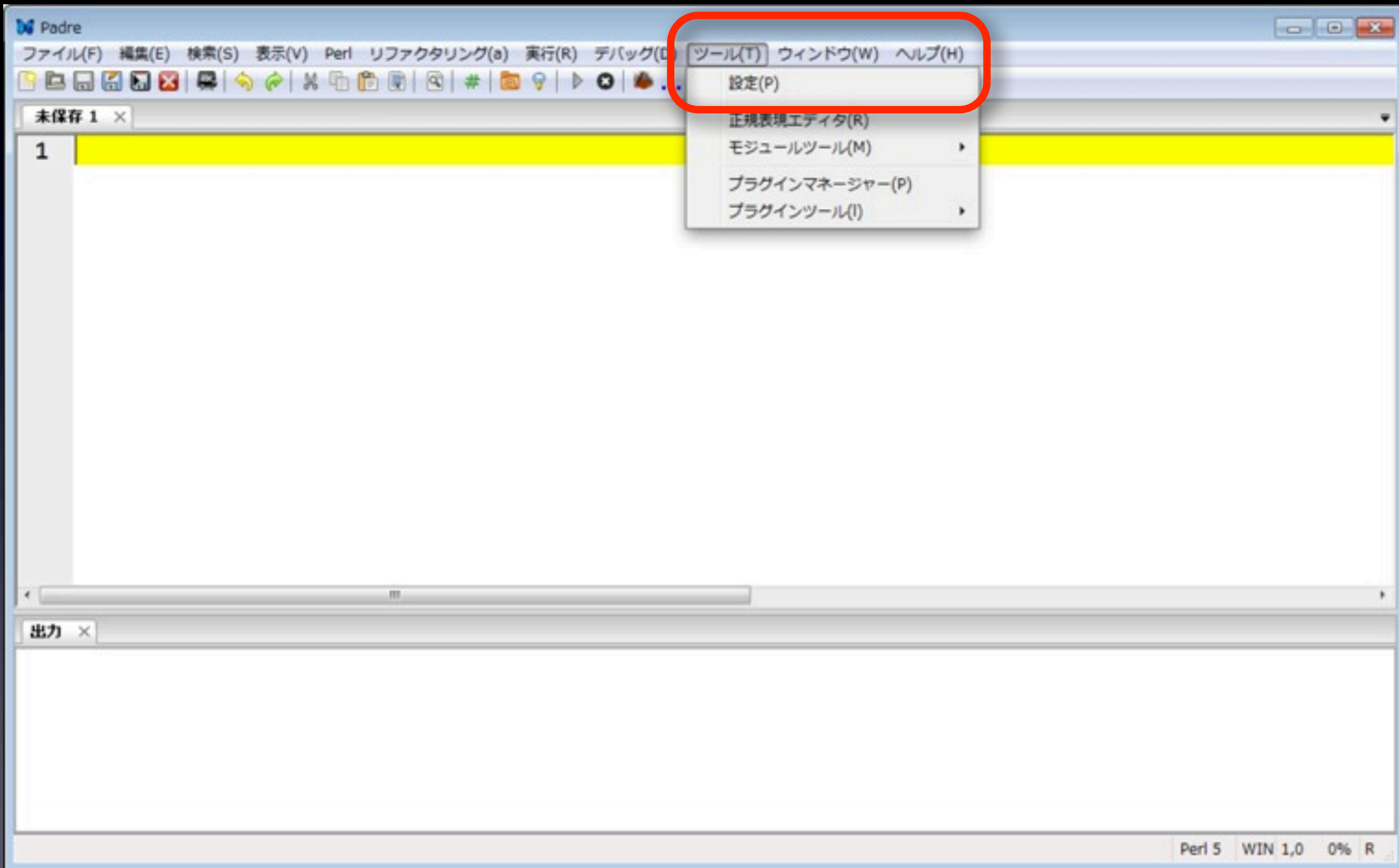


Padreを実行

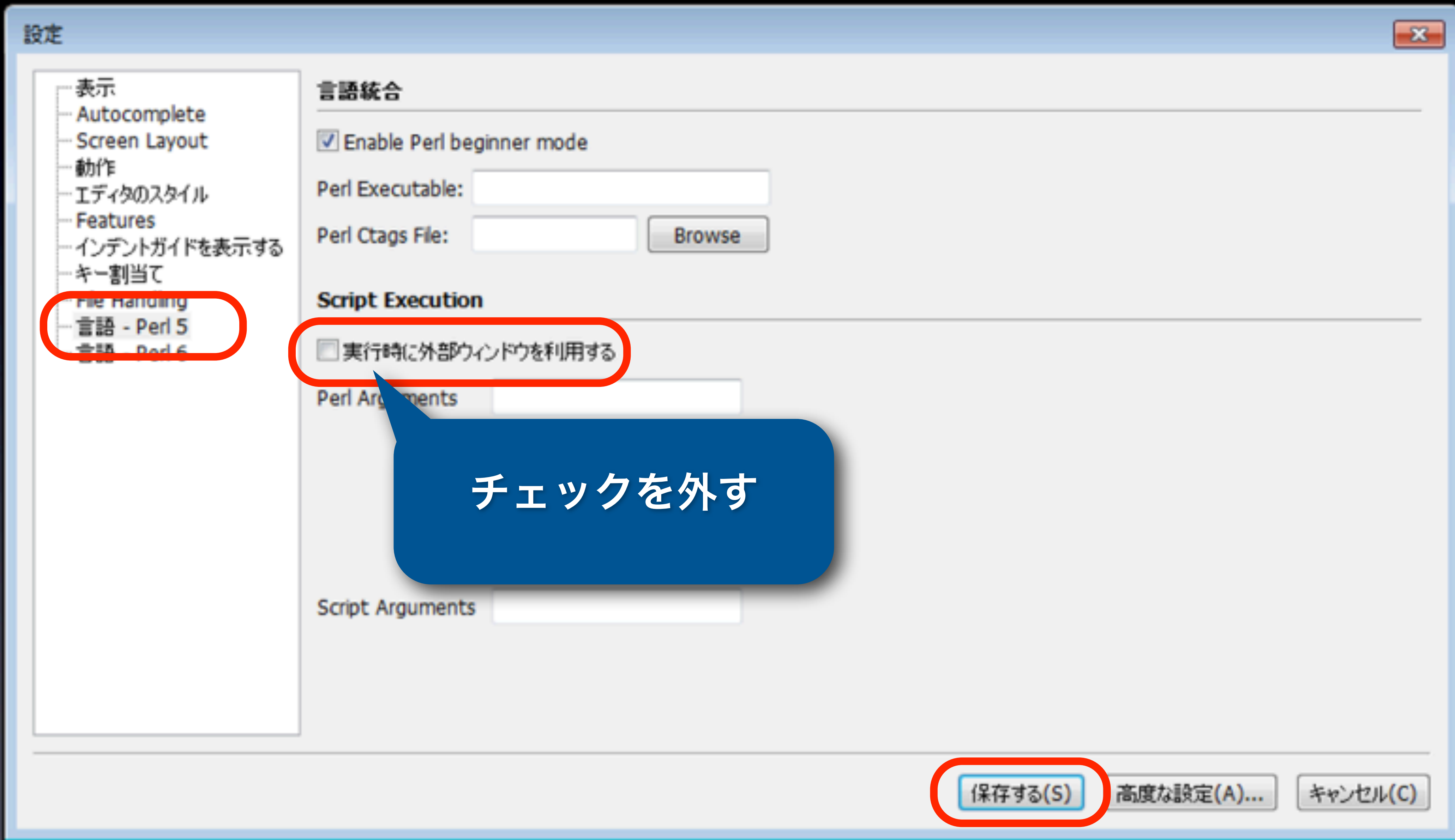
http://padre.perlide.org



まずは設定


Padre
ファイル(F) 編集(E) 検索(S) 表示(V) Perl リファクタリング(a) 実行(R) デバッグ(D) ツール(T) ウィンドウ(W) ヘルプ(H)
設定(P)
正規表現エディタ(R)
モジュールツール(M)
プラグインマネージャー(P)
プラグインツール(I)
未保存 1
1
出力
Perl 5 WIN 1,0 0% R

まずは設定


設定
表示
Autocomplete
Screen Layout
動作
エディタのスタイル
Features
インデントガイドを表示する
キー割当て
言語 - Perl 5
言語統合
Enable Perl beginner mode
Perl Executable:
Perl Ctags File:
Browse
Script Execution
実行時に外部ウィンドウを利用する
チェックを外す
Script Arguments
保存する(S)
高度な設定(A)...
キャンセル(C)

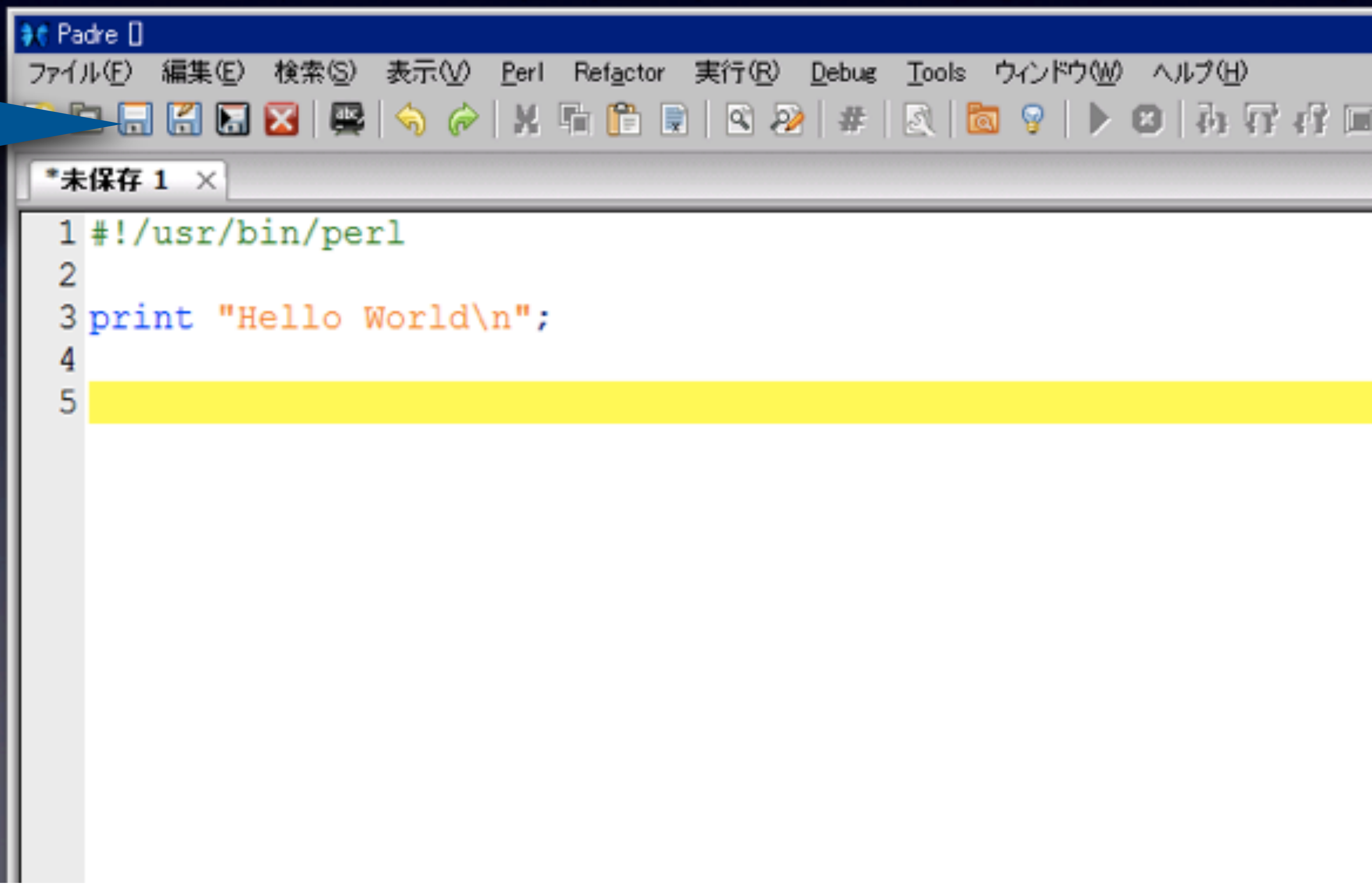
最初のプログラム

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

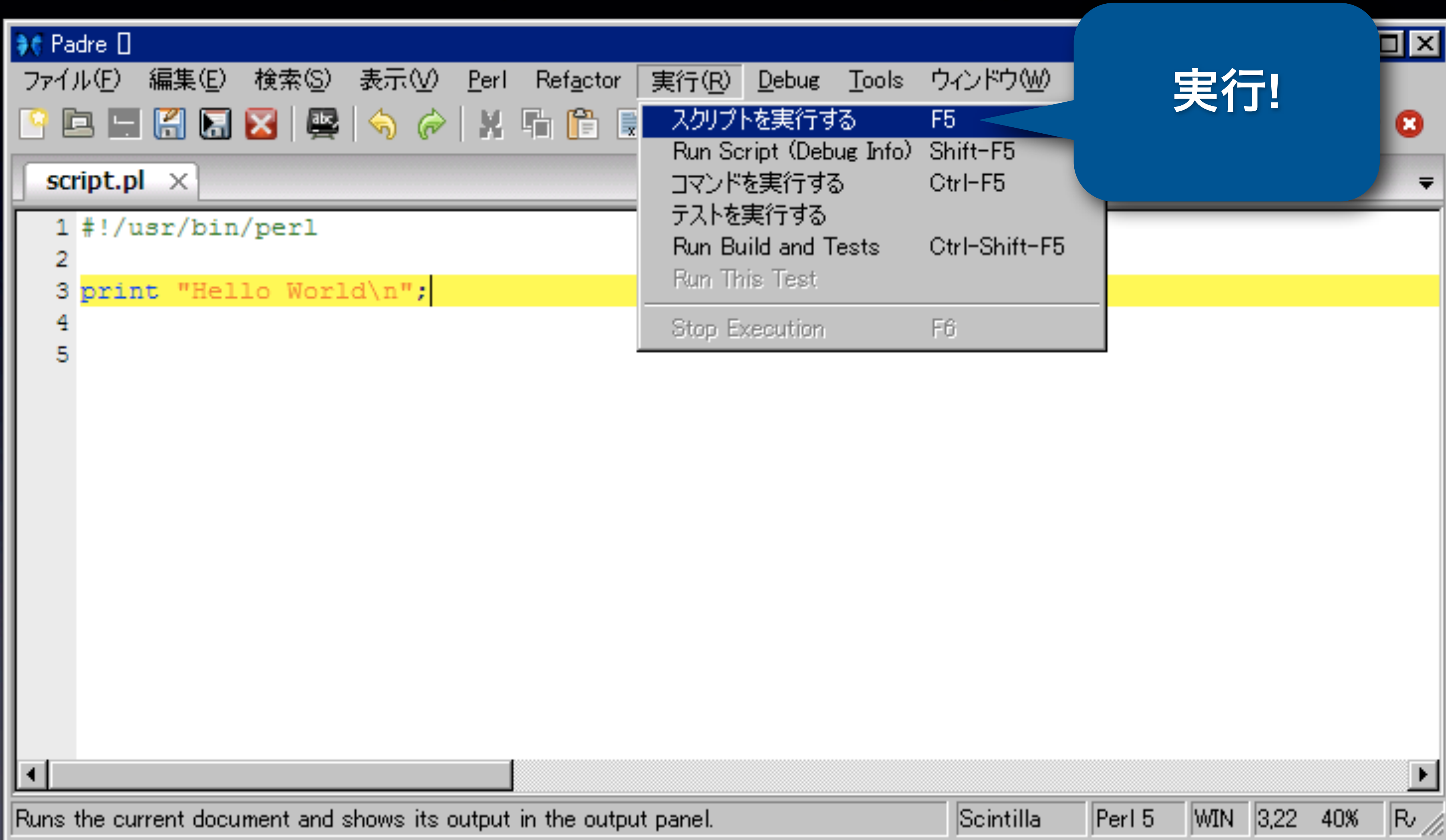
¥

保存

(script.pl)



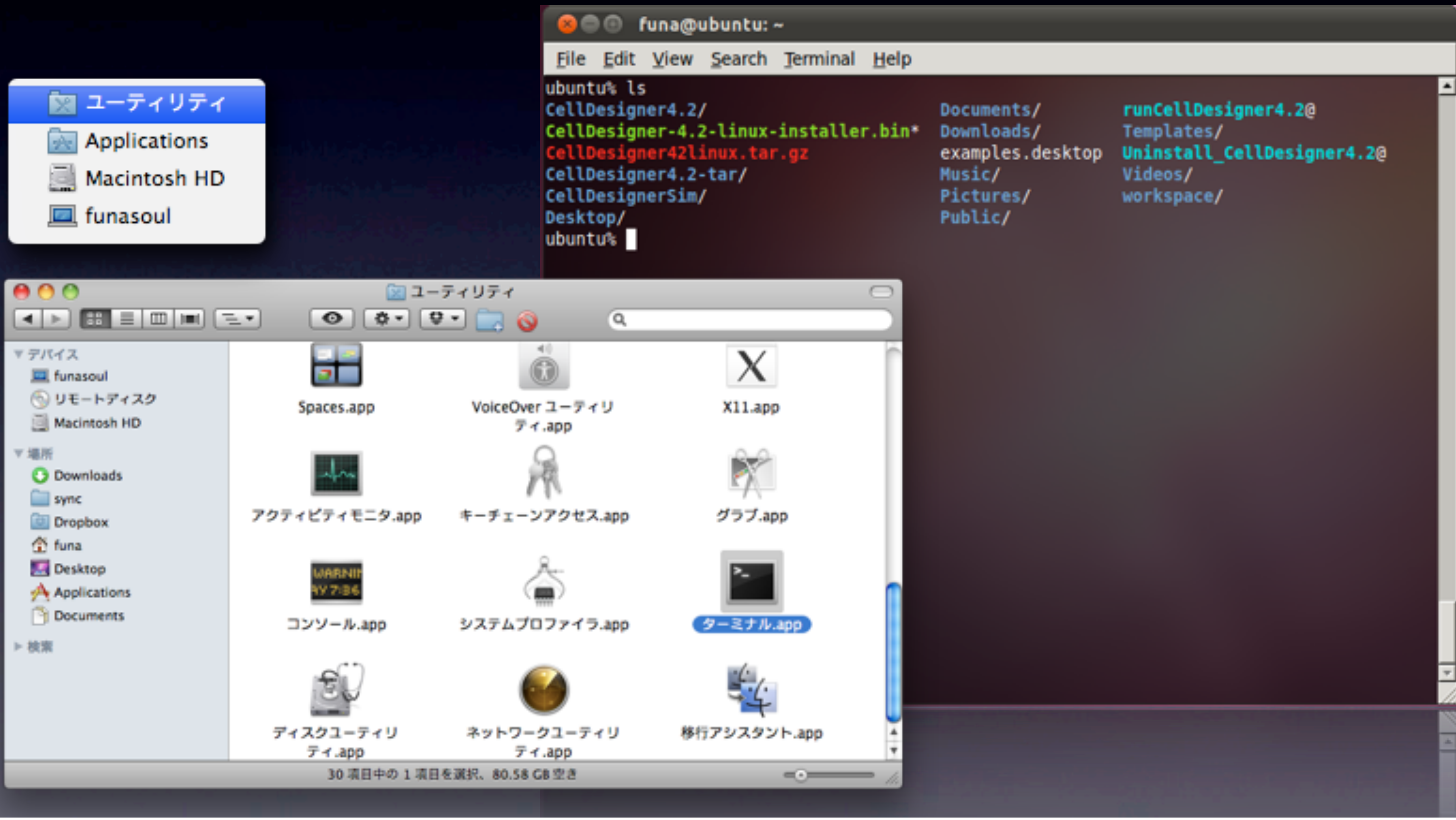
```
#!/usr/bin/perl
print "Hello World\n";
```



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

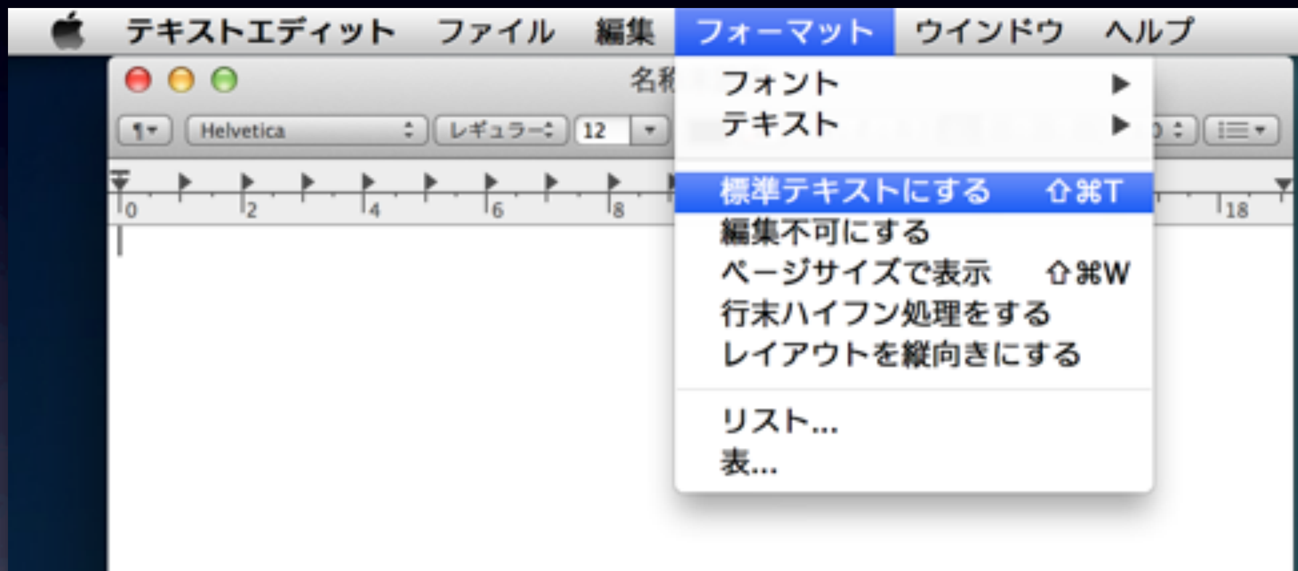
Mac, Linuxの人

Terminalを起動



Emacs, Xcode, テキストエディットなどで作成

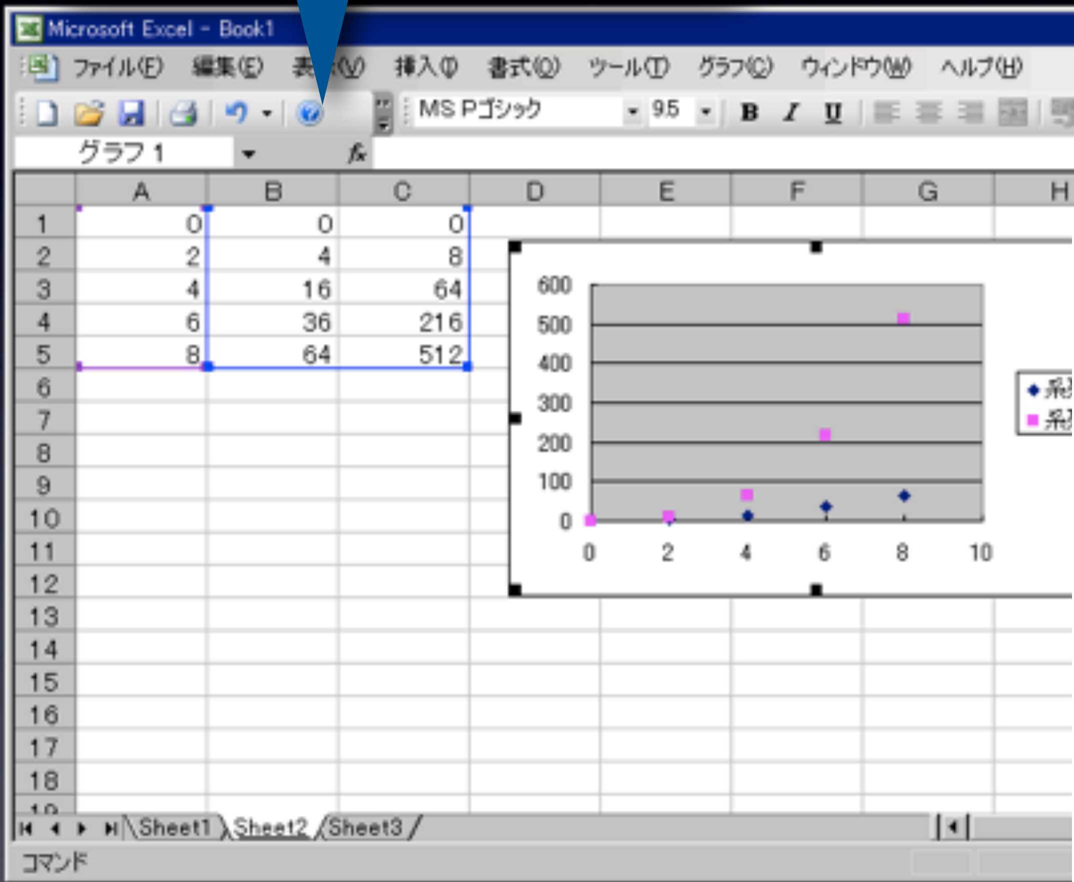
```
#!/usr/bin/perl
print "Hello World\n";
```



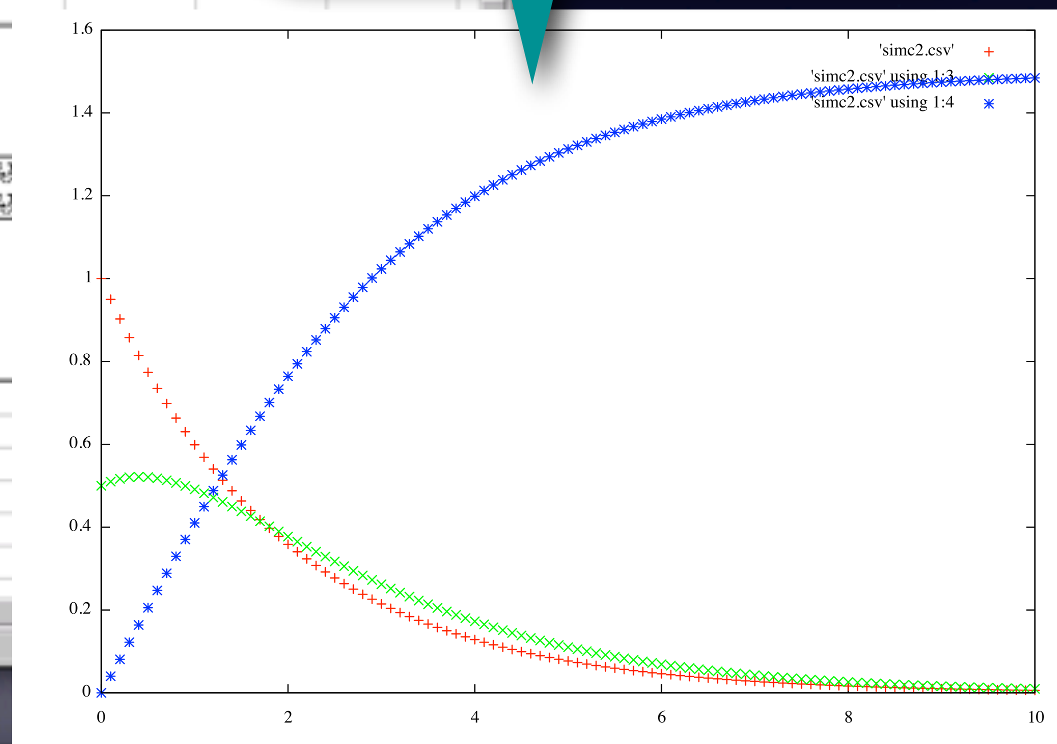

```
emacs script.pl
(Perlのプログラムを書く)

perl script.pl
Hello World
```

プロット

Excel等の表計算ソフト

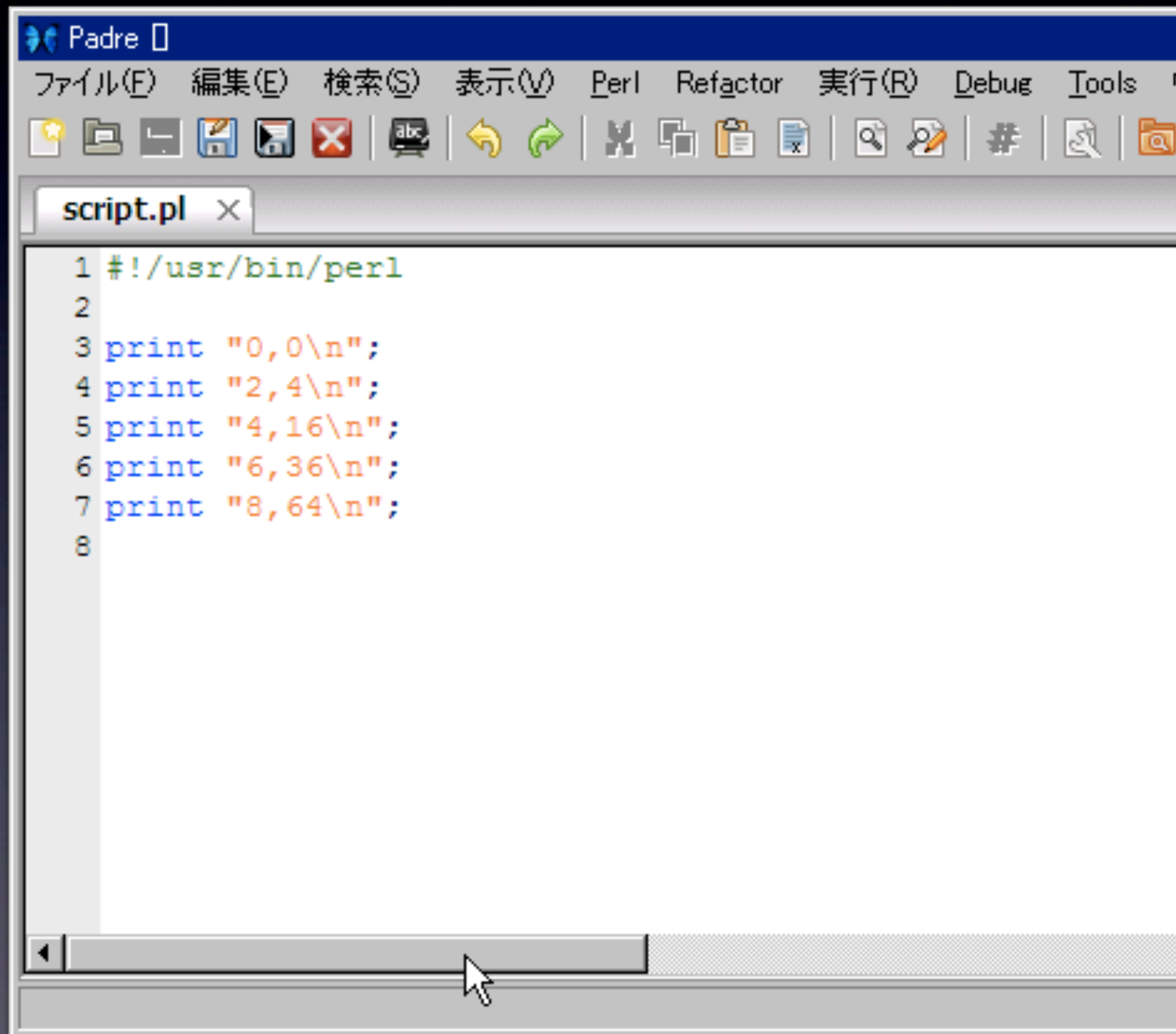


gnuplot等



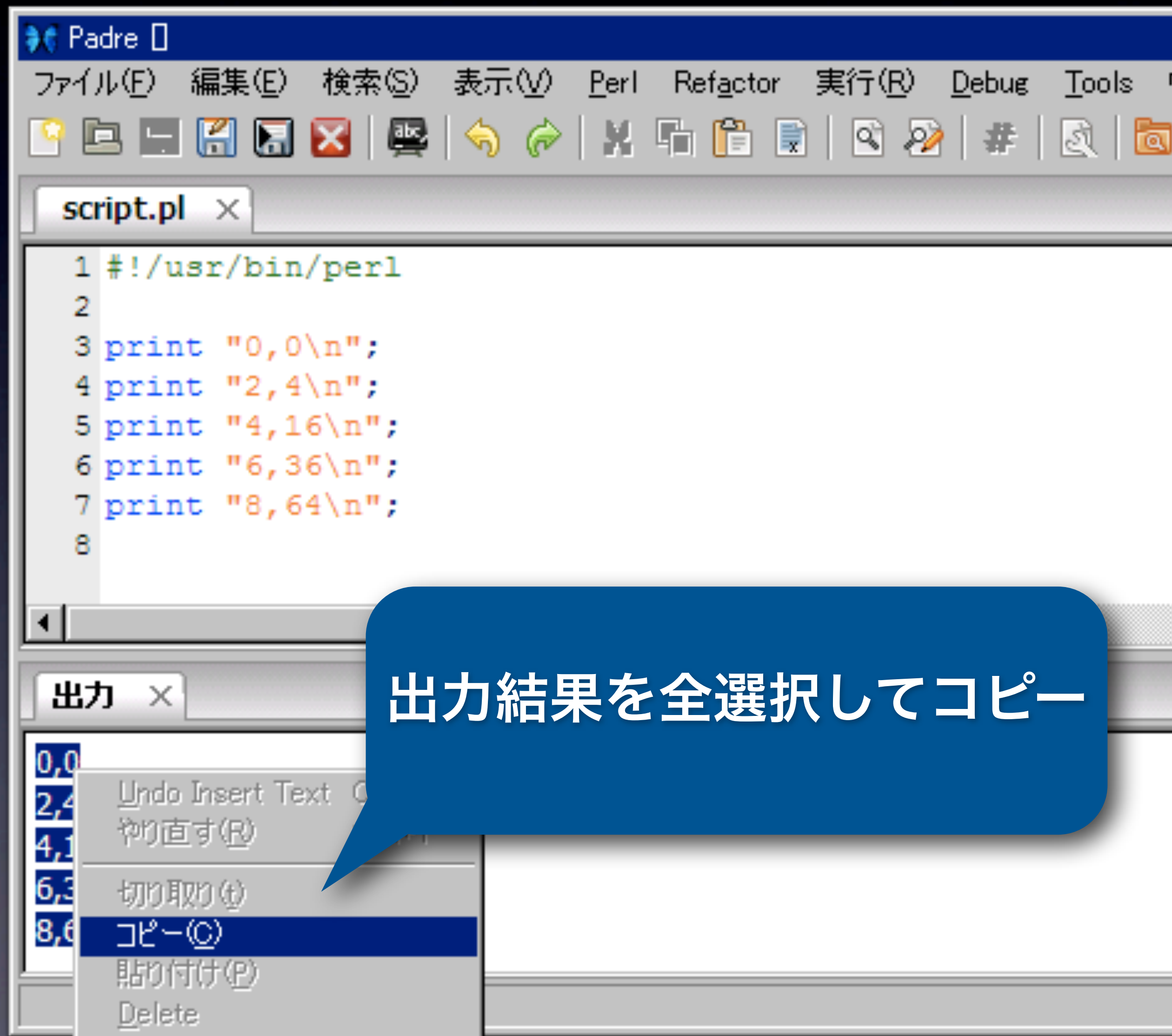
プロット用データを用意

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

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



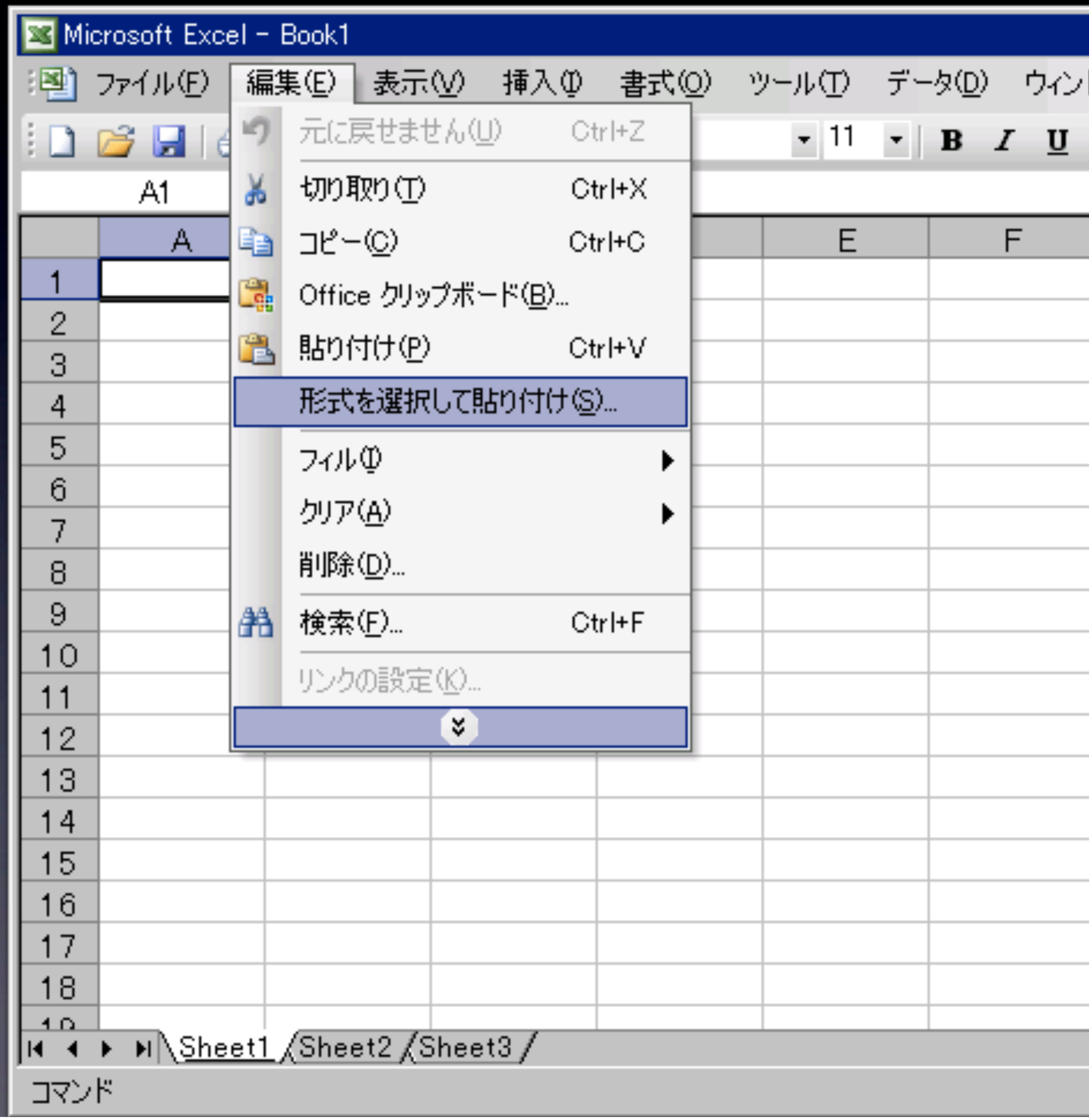
プロット用データを用意

```
#!/usr/bin/perl

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



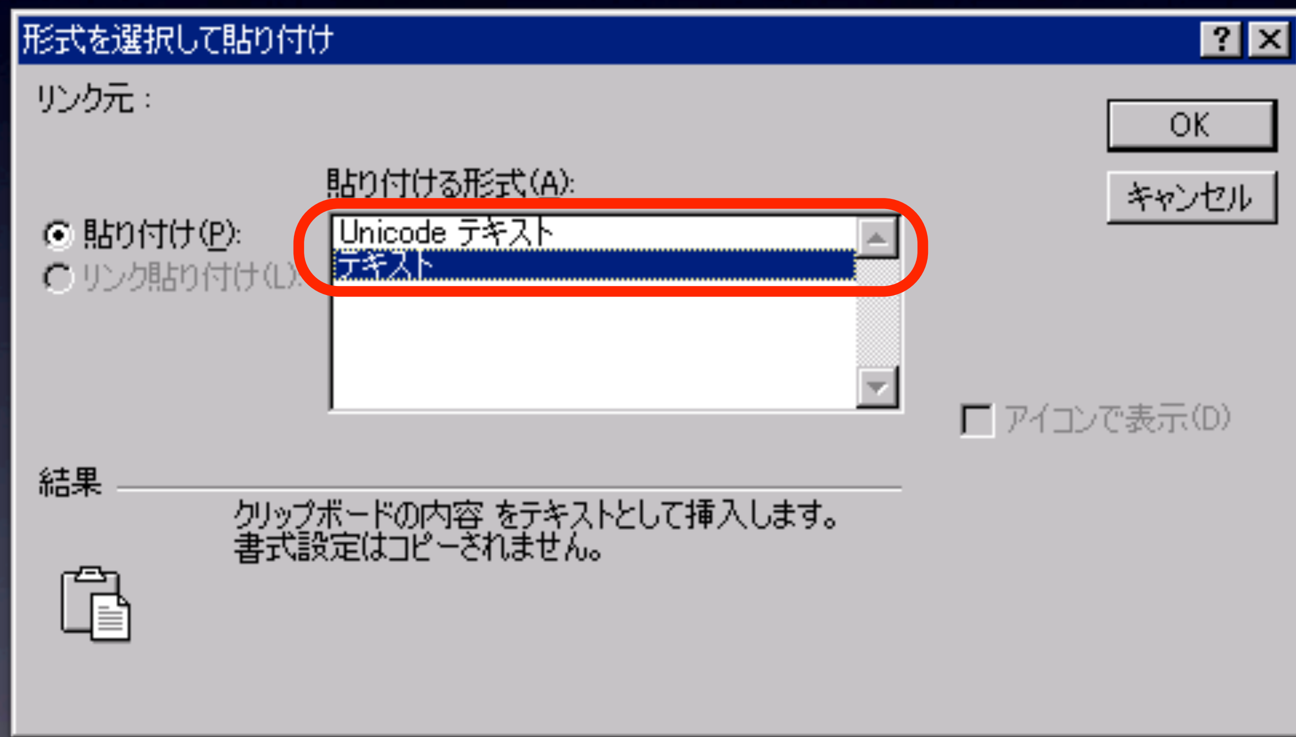
Excelに取り込む

```
#!/usr/bin/perl

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



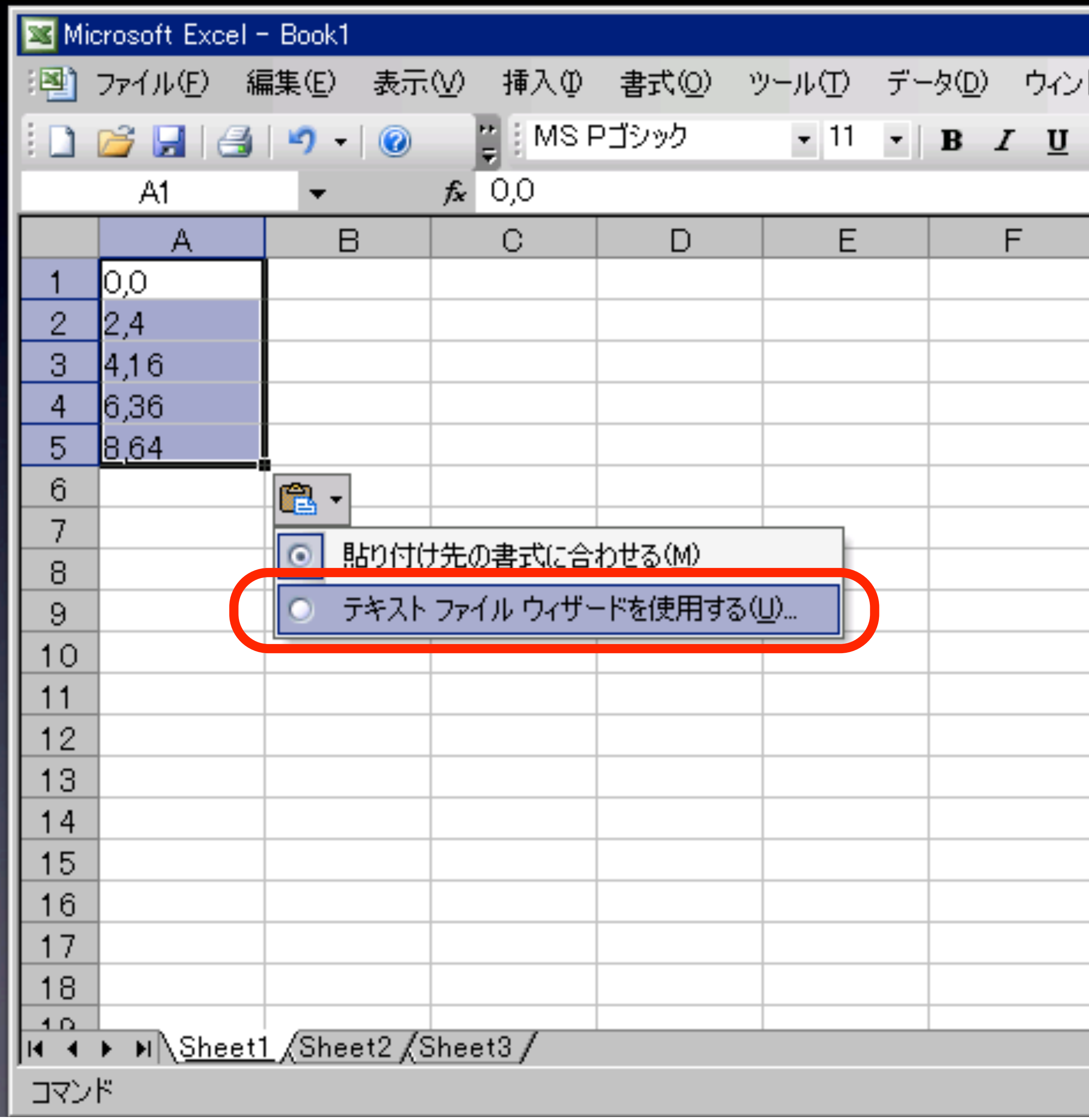
Excelに取り込む

```
#!/usr/bin/perl

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



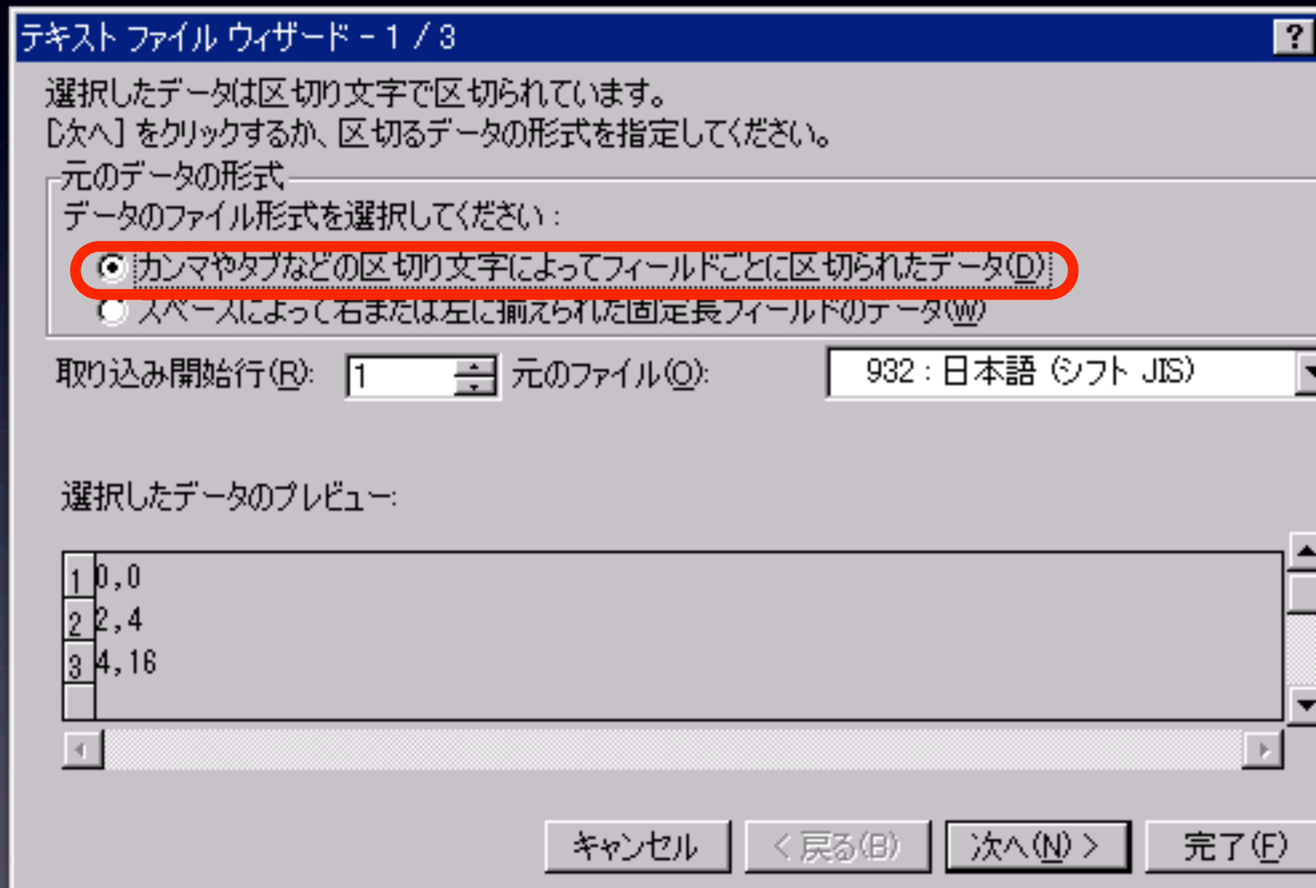
Excelに取り込む

```
#!/usr/bin/perl

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



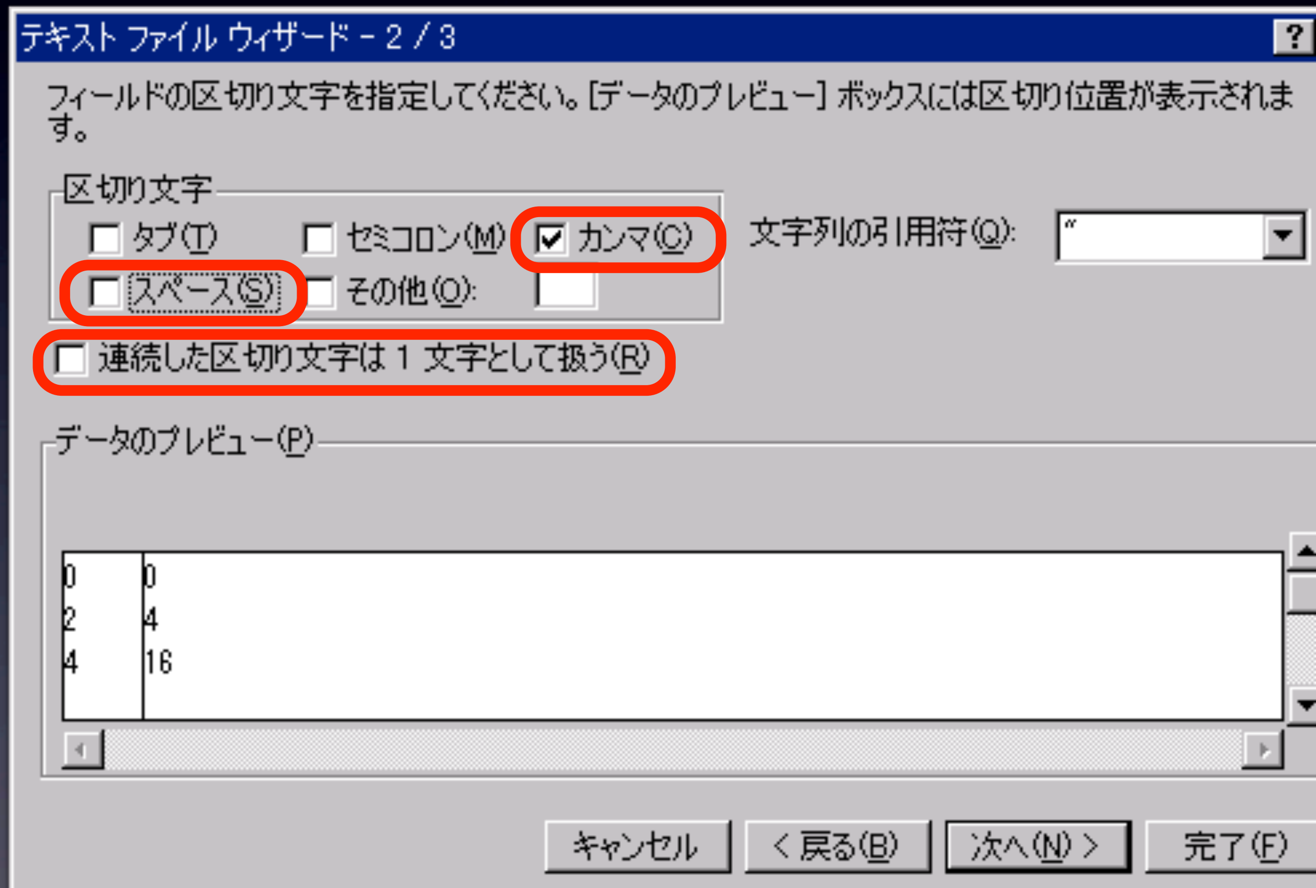
Excelに取り込む

```
#!/usr/bin/perl

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



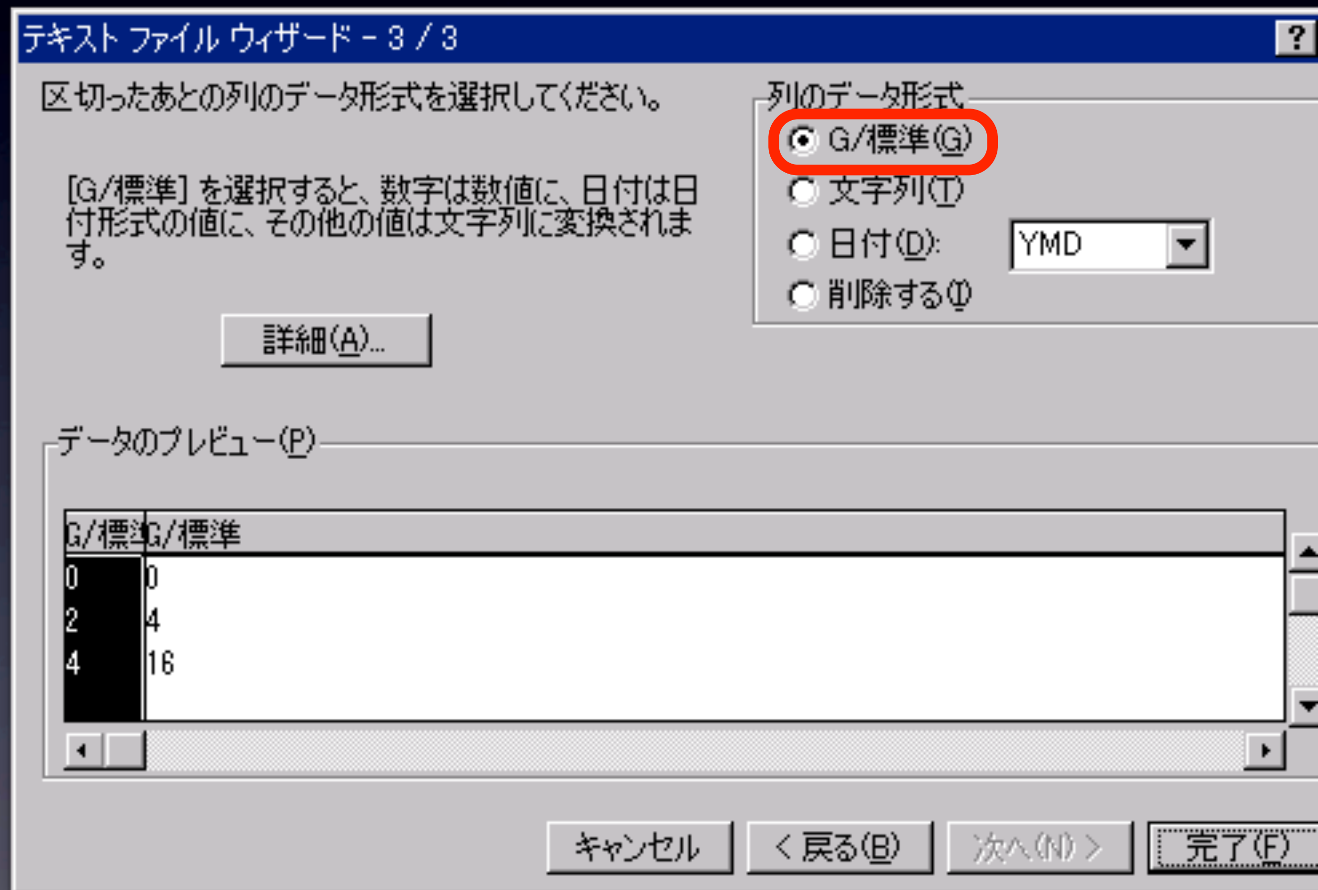
Excelに取り込む

```
#!/usr/bin/perl

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



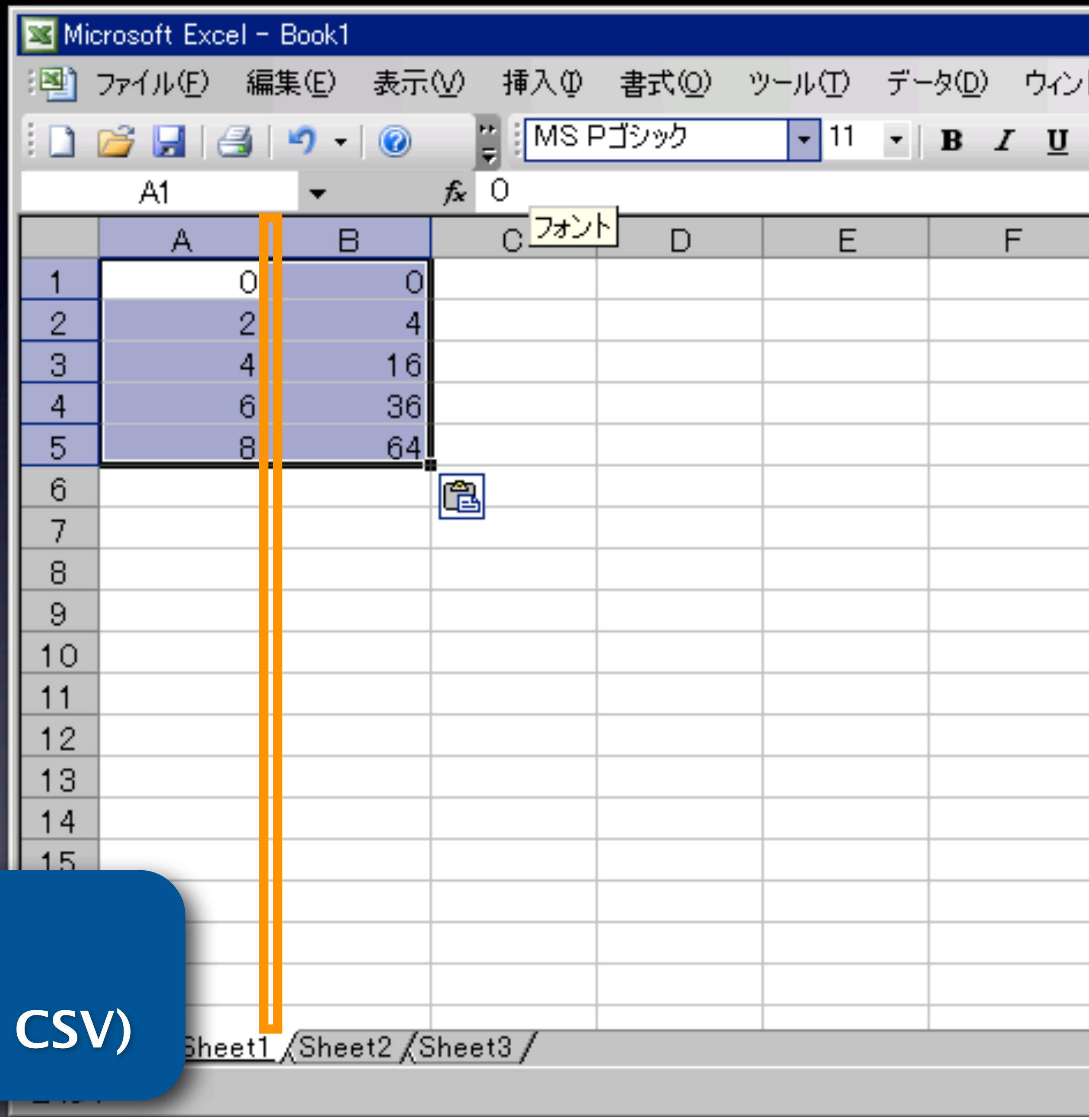
Excelに取り込む

```
#!/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";
```

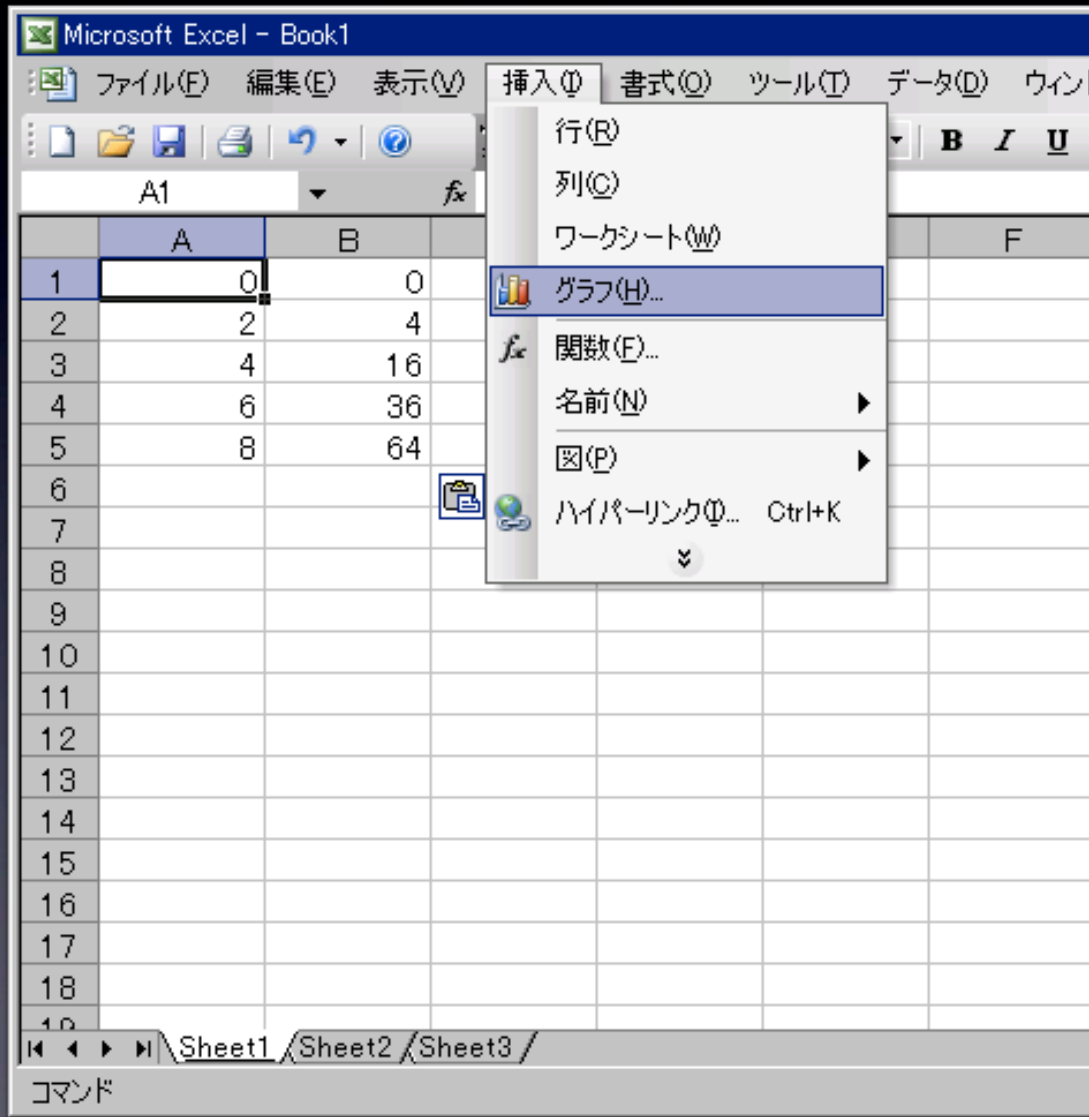


カンマ区切り

(Comma Separated Values: CSV)

ExcelでプロットExcelでプロット

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

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

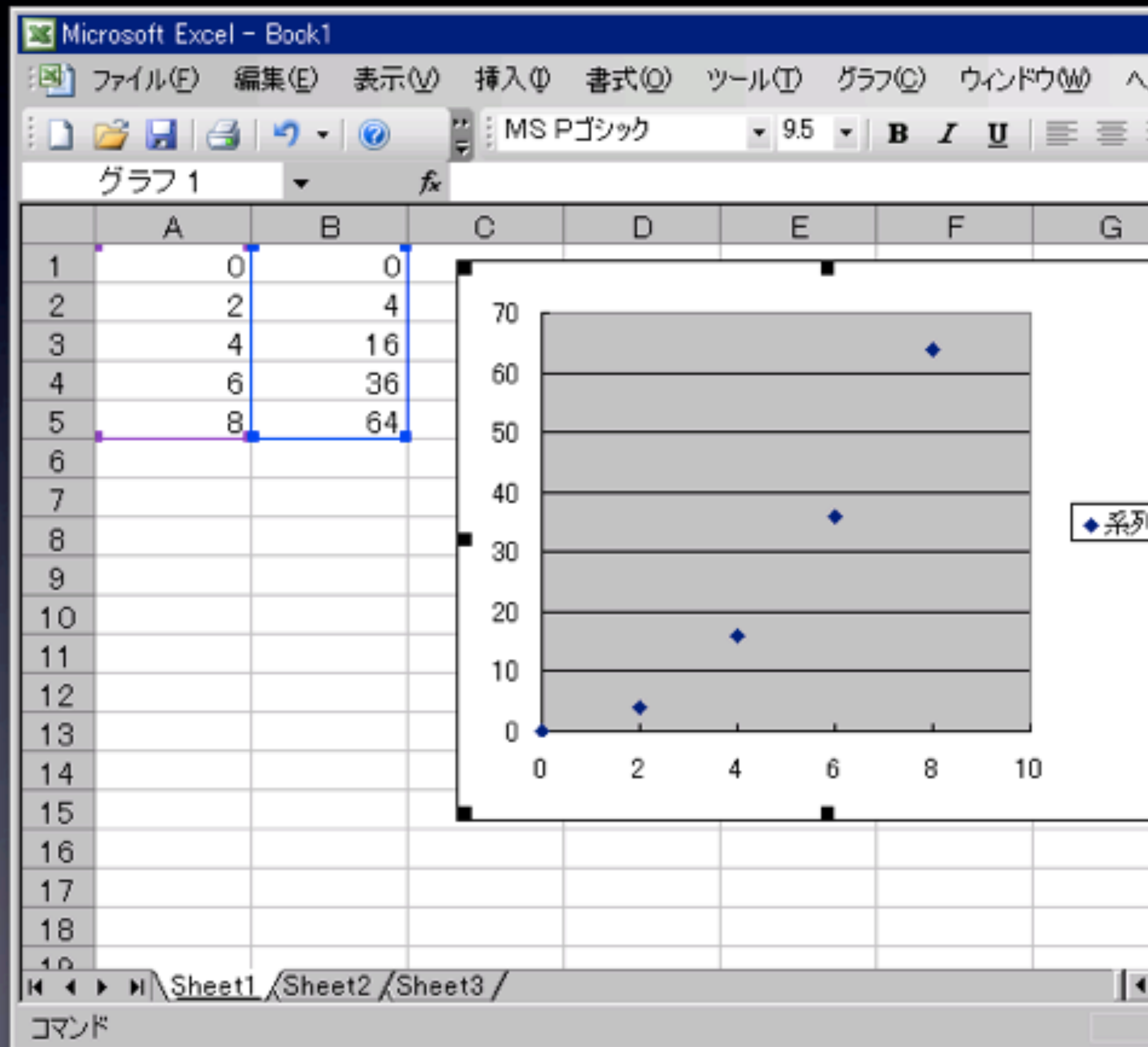


Excelでプロット

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

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

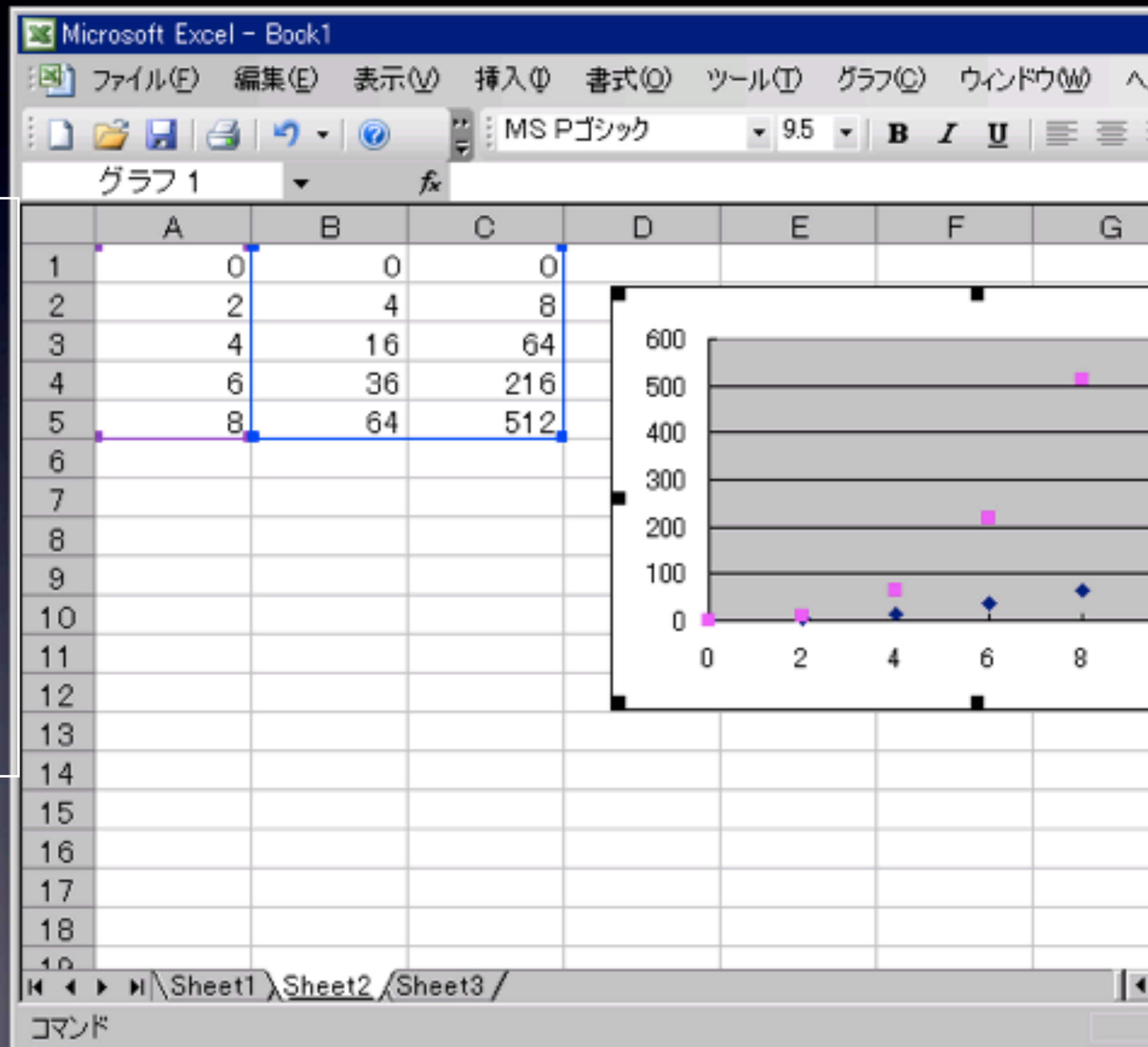
Excelでプロット

```
#!/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

```
#!/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
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

```
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

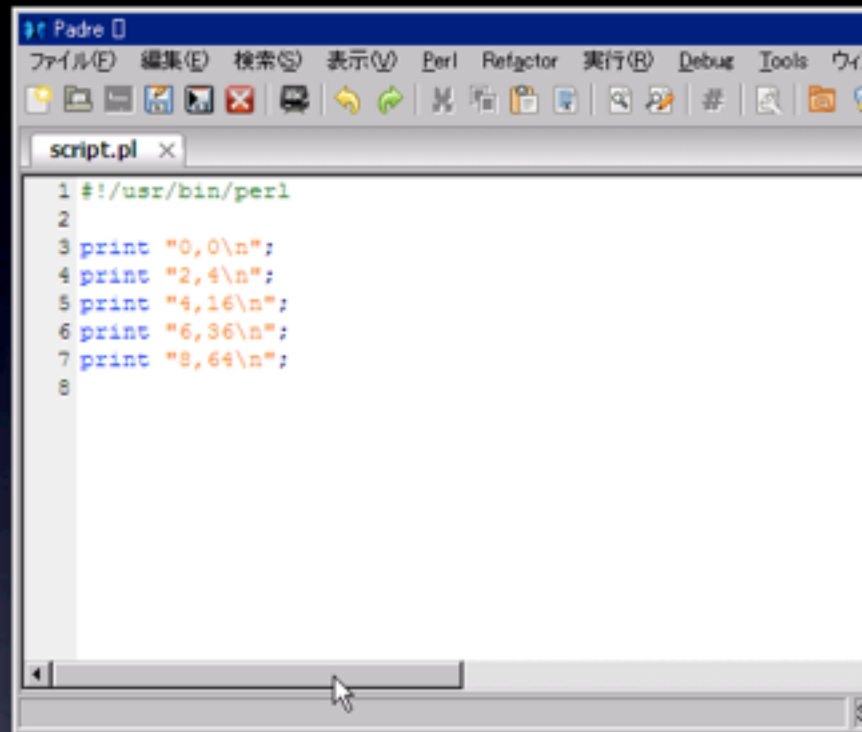
```
perl script.pl >! data.csv
gnuplot
gnuplot> set datafile separator “,”
gnuplot> plot ‘data.csv’
gnuplot> replot ‘data.csv’ using 1:3
```

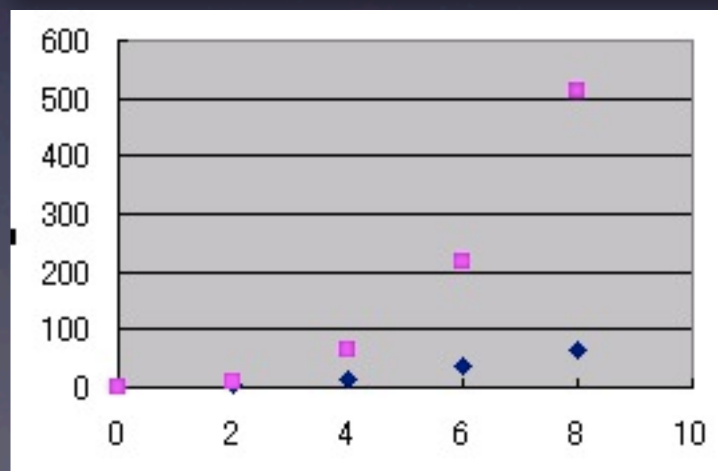
```
#!/usr/bin/perl

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

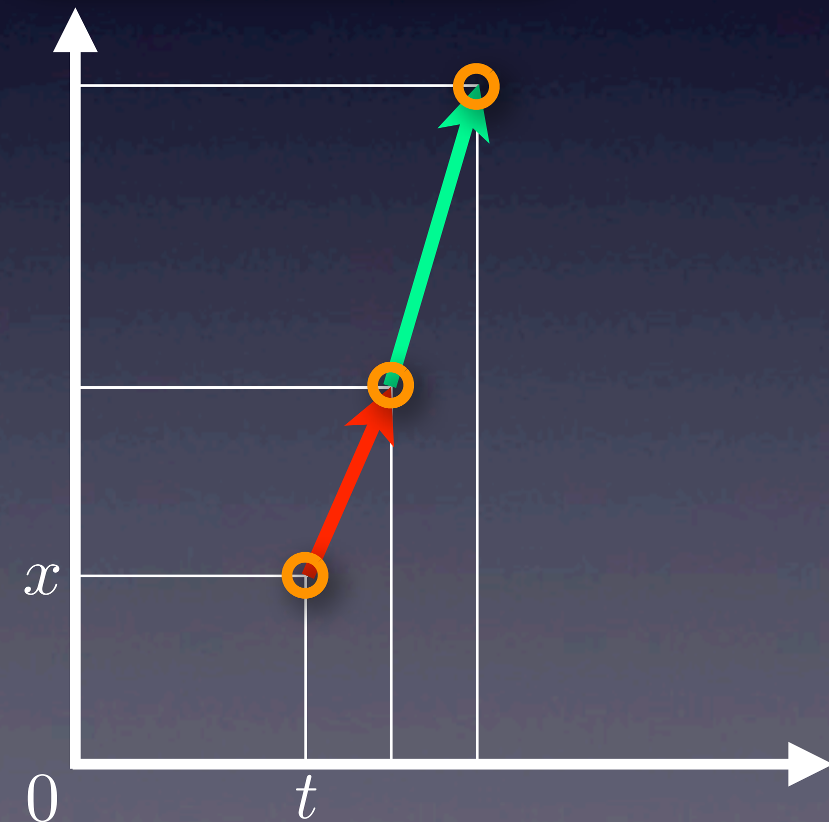




足りないもの

- 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

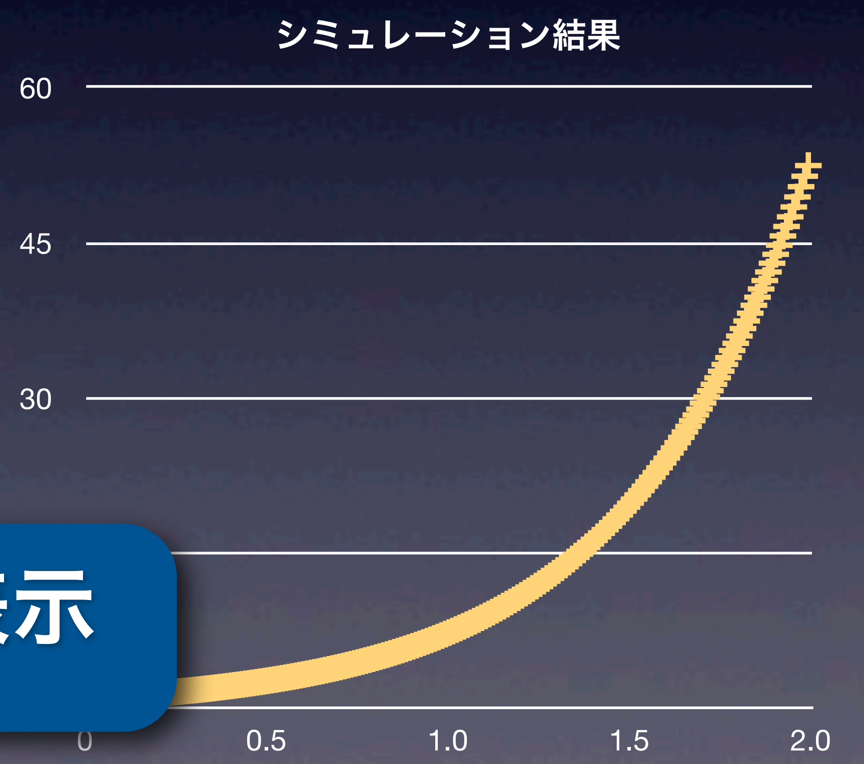


ODE Simulator

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

```
#!/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



ODE Simulator

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

```
#!/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;
}
```

Δt の値

x, tの初期値

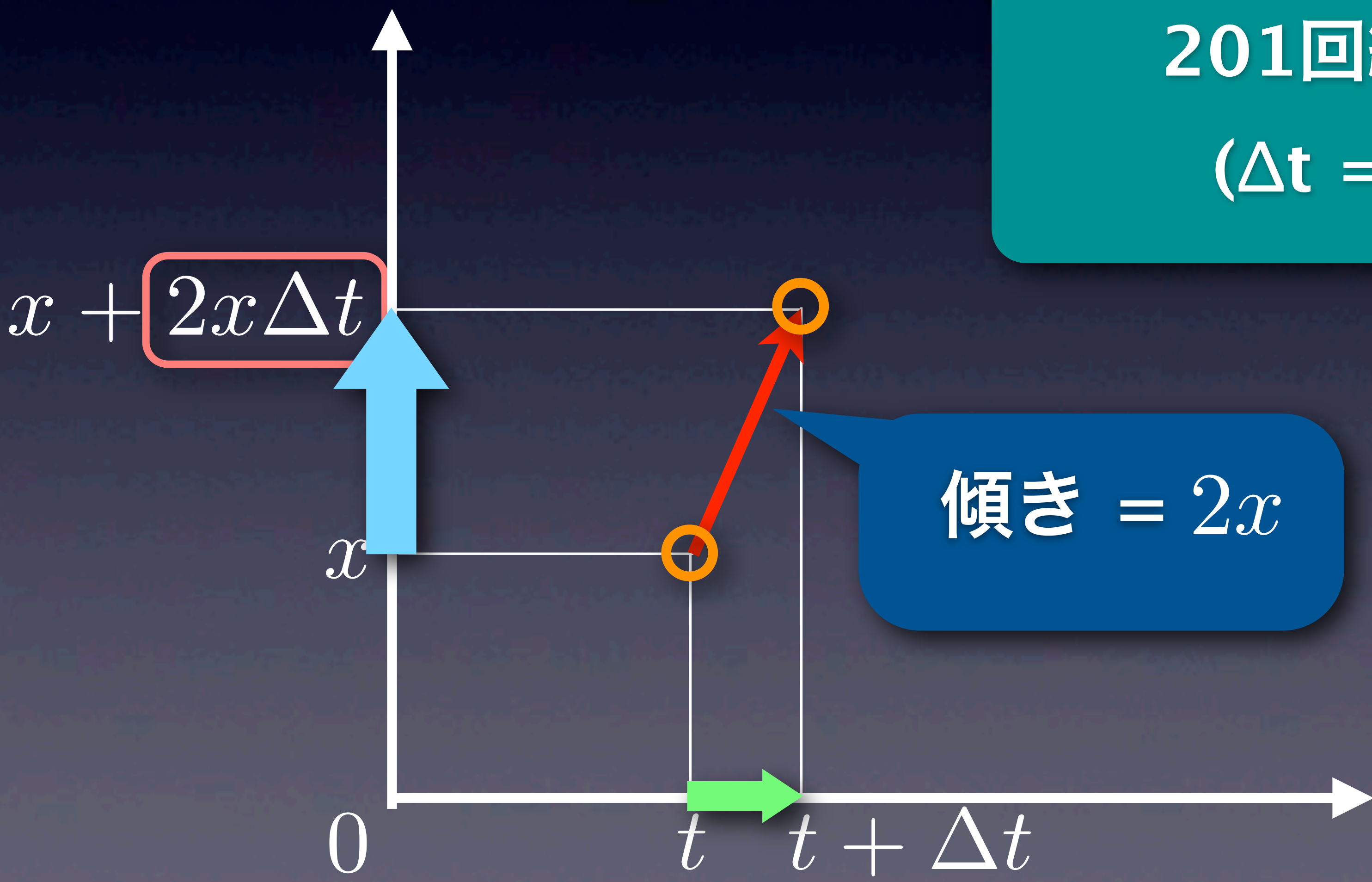
201回繰り返す

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
(Δt = 0.01)



傾き = $2x$

ポイント

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

```
#!/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;
}
```

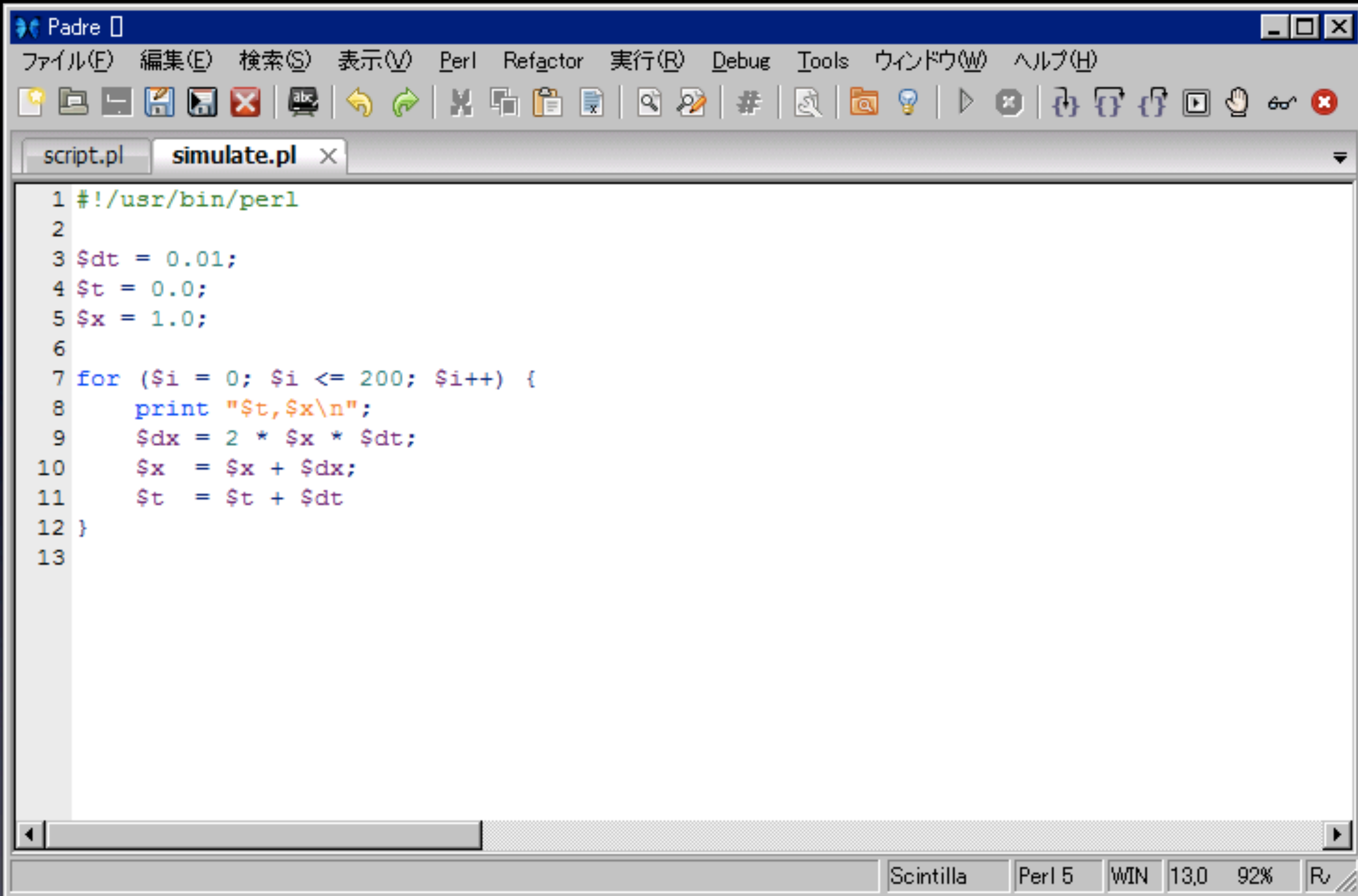
Δt の値

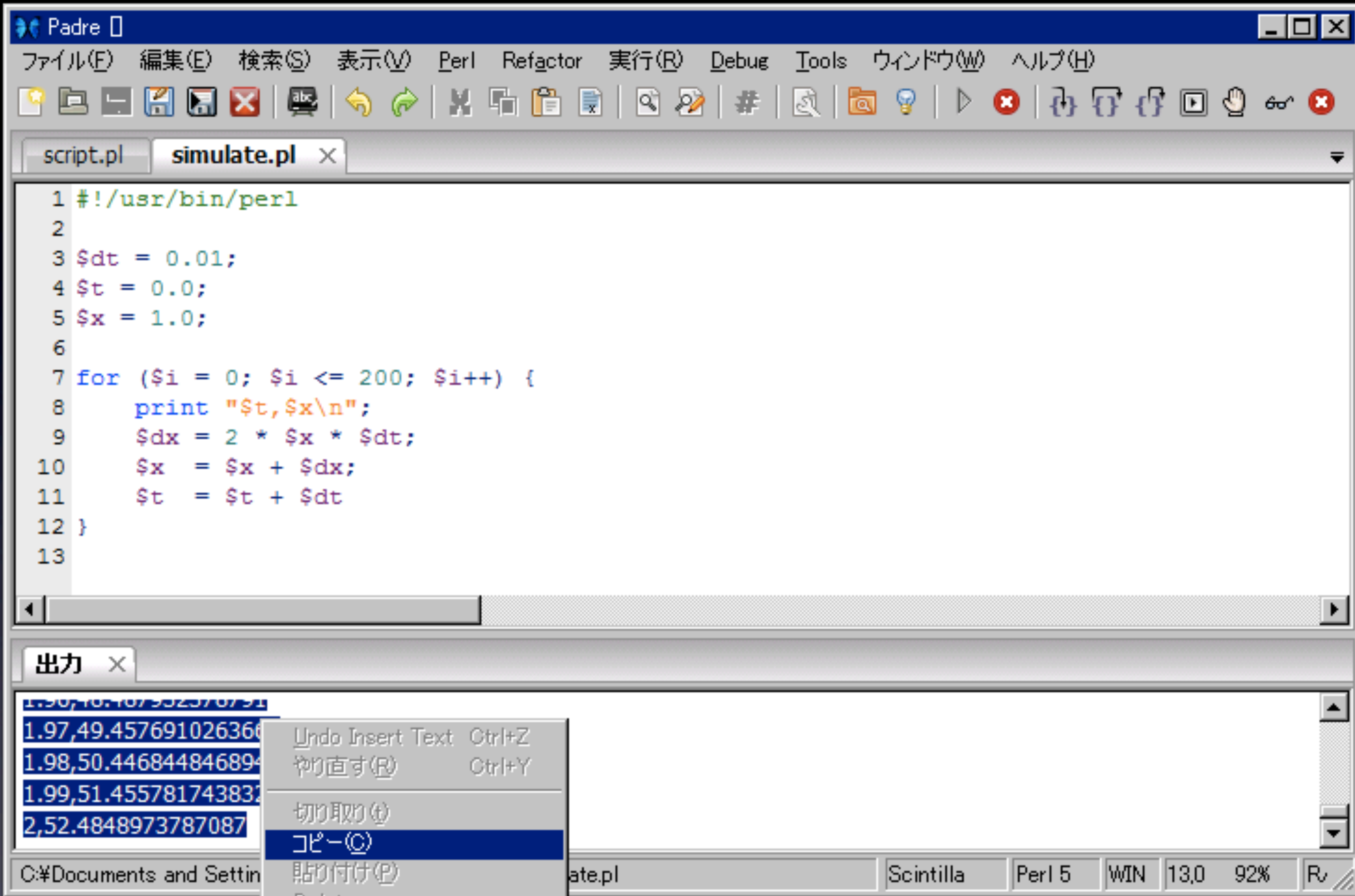
x, tの初期値

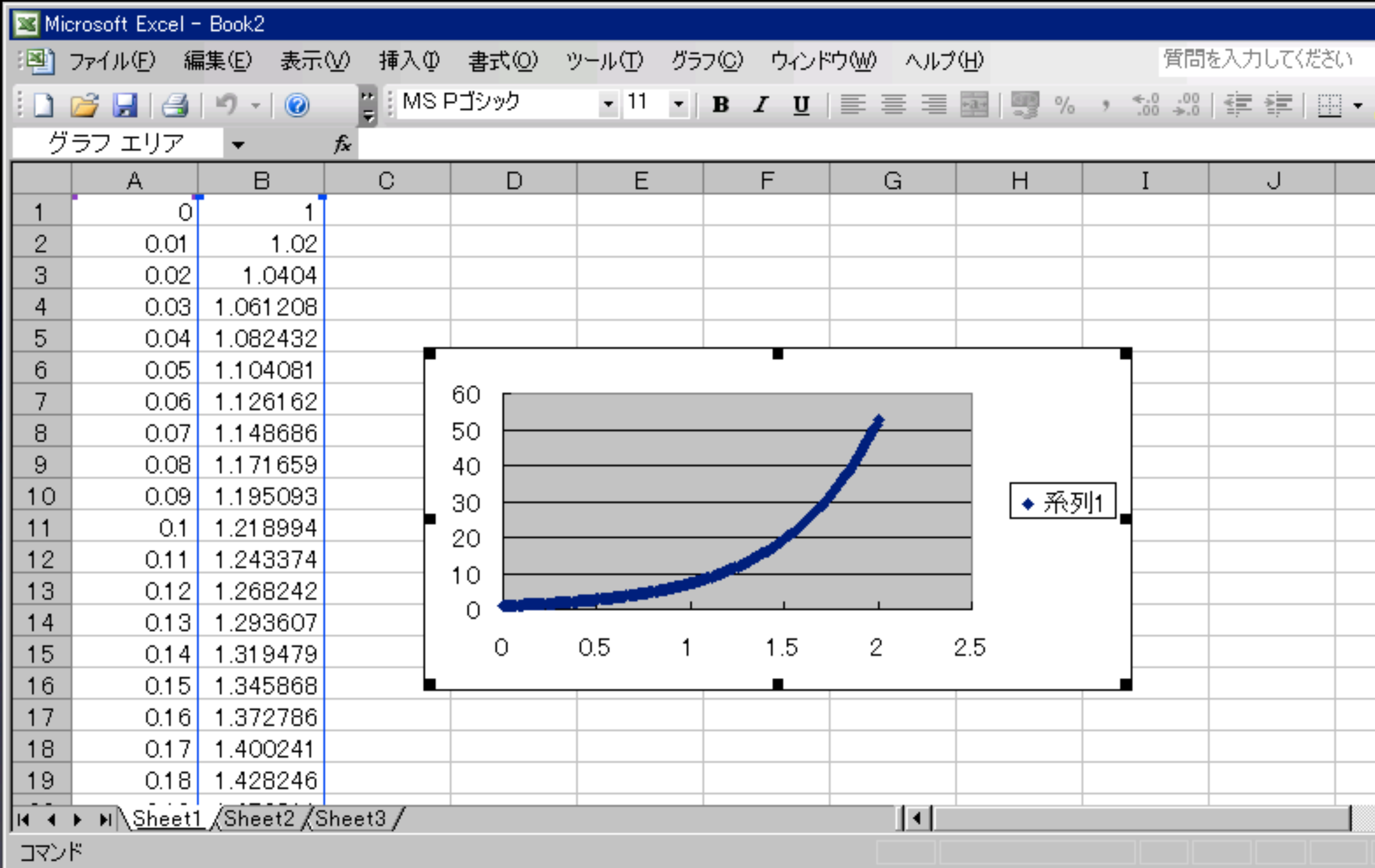
シミュレーション時間とΔtによって繰り返し回数を決める

dx/dt = の右辺を書く

xとtをちょっと増やす(減らす)

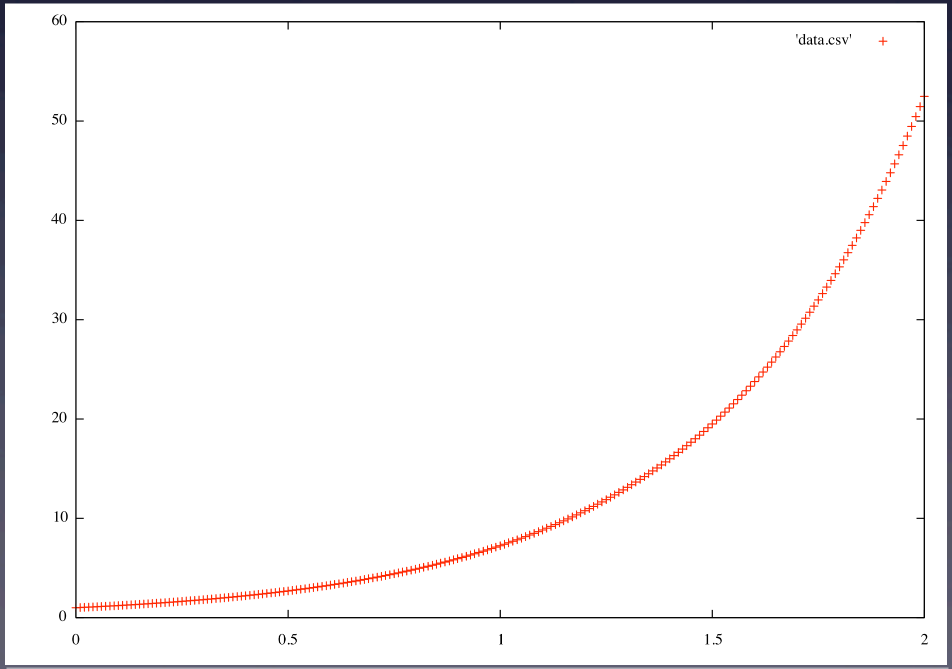

Padre
ファイル(F) 編集(E) 検索(S) 表示(V) Perl Refactor 実行(R) Debug Tools ウィンドウ(W) ヘルプ(H)
script.pl
simulate.pl
#!/usr/bin/perl
$dt = 0.01;
$t = 0.0;
$x = 1.0;
for ($i = 0; $i <= 200; $i++) {
print "$t,$x\n";
$dx = 2 * $x * $dt;
$x = $x + $dx;
$t = $t + $dt
}
Scintilla
Perl 5
WIN
13,0
92%
R/


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
#!/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
}
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(t)
コピー(C)
貼り付け(P)
C:¥Documents and Settin
ate.pl
Scintilla
Perl 5
WIN
13,0
92%


Microsoft Excel - Book2
ファイル(F) 編集(E) 表示(V) 挿入(I) 書式(O) ツール(T) グラフ(C) ウィンドウ(W) ヘルプ(H)
質問を入力してください
MS Pゴシック
11
グラフ エリア
fx
A B C D E F G H I J
1 0 1
2 0.01 1.02
3 0.02 1.0404
4 0.03 1.061208
5 0.04 1.082432
6 0.05 1.104081
7 0.06 1.126162
8 0.07 1.148686
9 0.08 1.171659
10 0.09 1.195093
11 0.1 1.218994
12 0.11 1.243374
13 0.12 1.268242
14 0.13 1.293607
15 0.14 1.319479
16 0.15 1.345868
17 0.16 1.372786
18 0.17 1.400241
19 0.18 1.428246
60
50
40
30
20
10
0
0 0.5 1 1.5 2 2.5
系列1
Sheet1 Sheet2 Sheet3
コマンド

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 $t = 0, \quad x = 1.0$

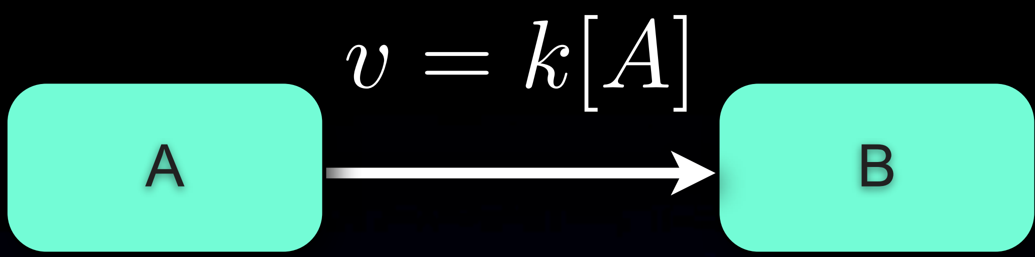
```
#include<stdio.h>

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



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

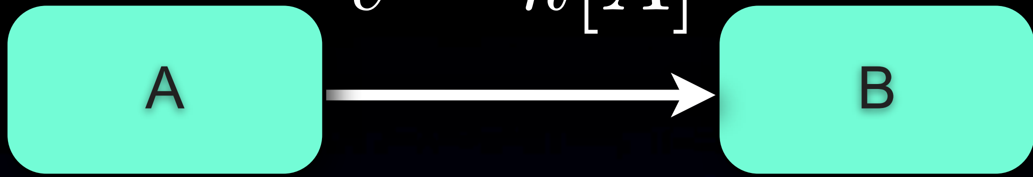
$$[A]_0 = 1.0$$
$$[B]_0 = 0.0$$
$$k = 1.5$$

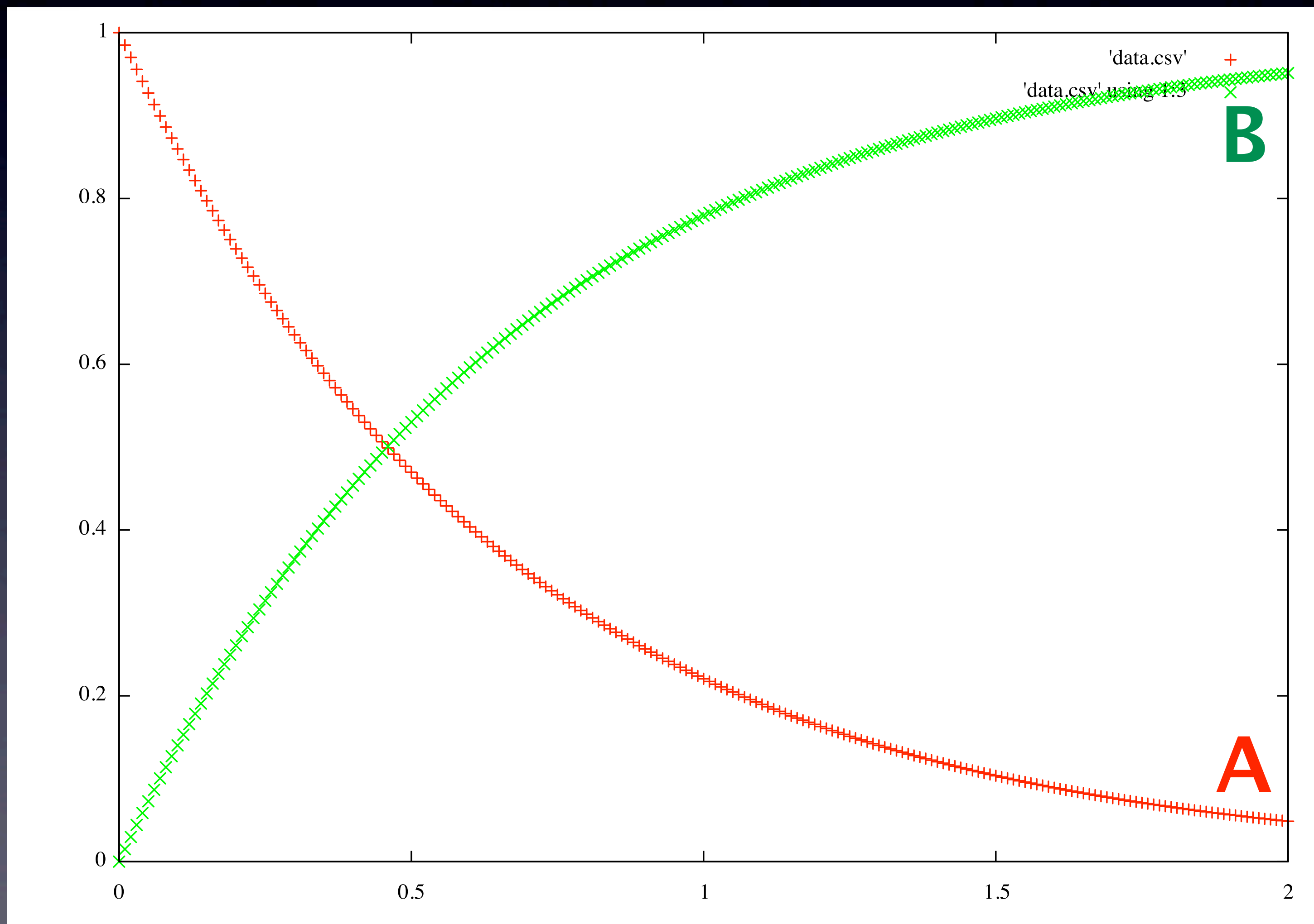
```
#!/usr/bin/perl

$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
}
```

A, B, t をちょっと増やす(減らす)

2変数モデル

$v = k[A]$





C言語版

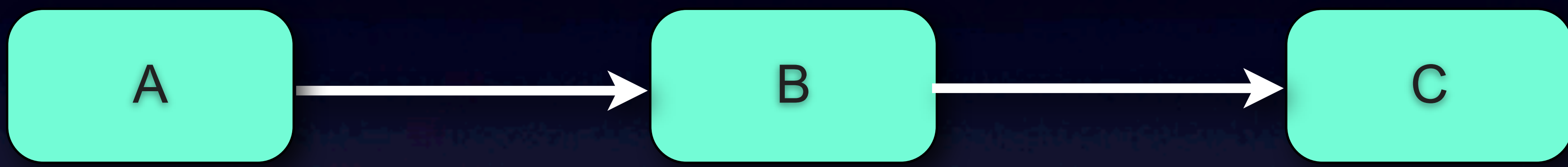
```c
#include<stdio.h>

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

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



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

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

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

Perl版

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

$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
}
```

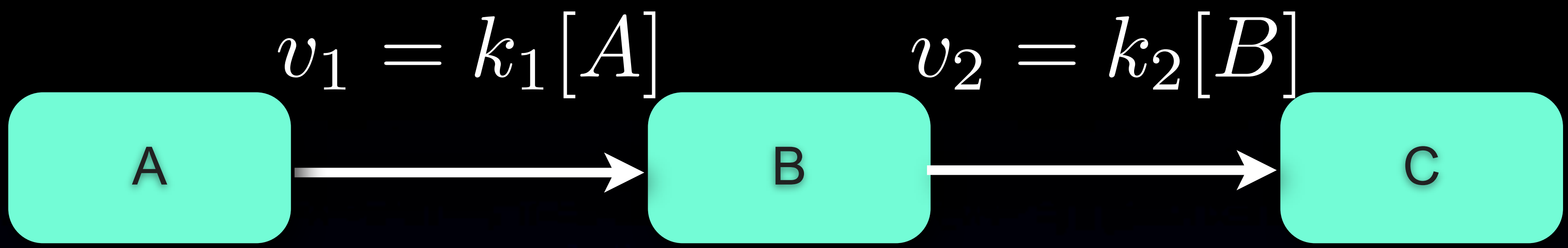
$$v_1 = k_1[A]$$
$$v_2 = k_2[B]$$

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

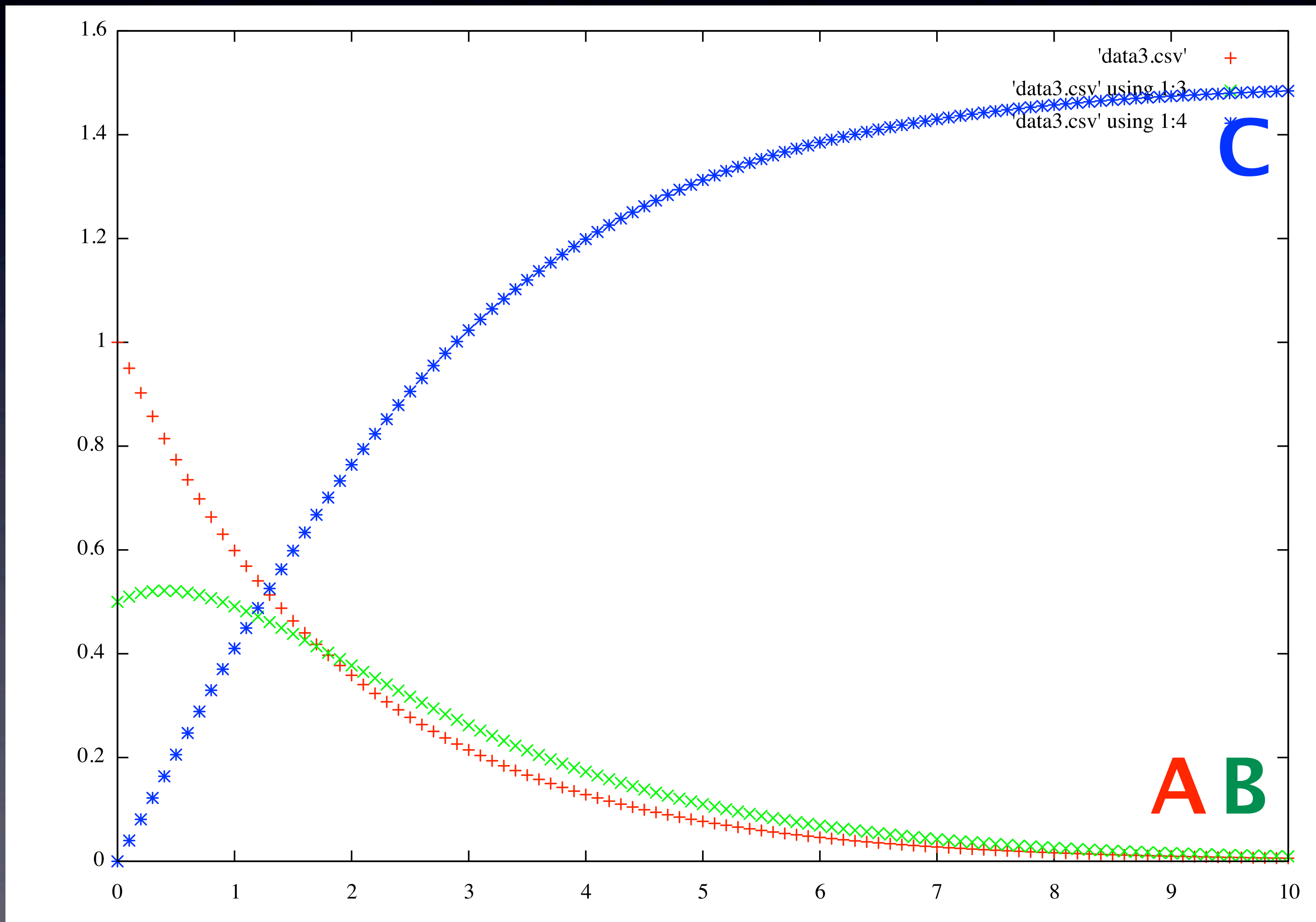
$$[A]_0 = 1.0, \quad [B]_0 = 0.5, \quad [C]_0 = 0.0$$

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

3変数モデル



$v_1 = k_1[A]$
$v_2 = k_2[B]$
A
B
C



1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
1
2
3
4
5
6
7
8
9
10
'data3.csv'
'data3.csv' using 1:3
'data3.csv' using 1:4
C
A
B

C言語版

```c
#include<stdio.h>

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;
}
```

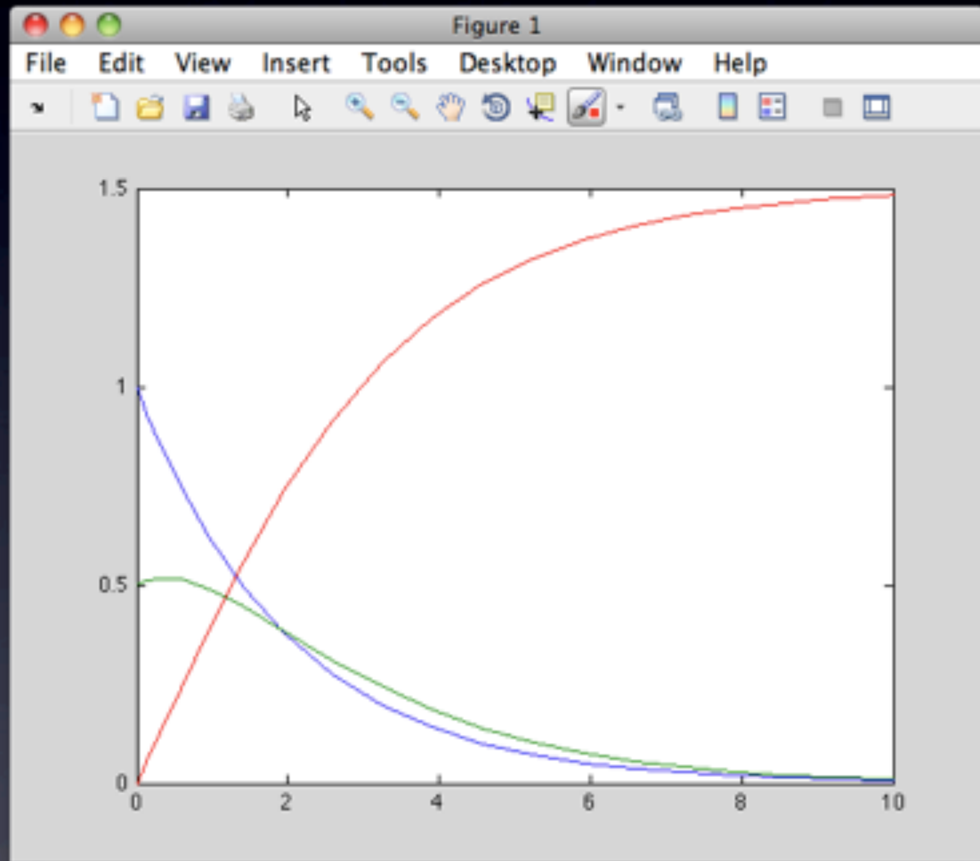
MATLAB版

sim3.m

```
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




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

Octave-Forge odepkg でも可

本格的なモデル

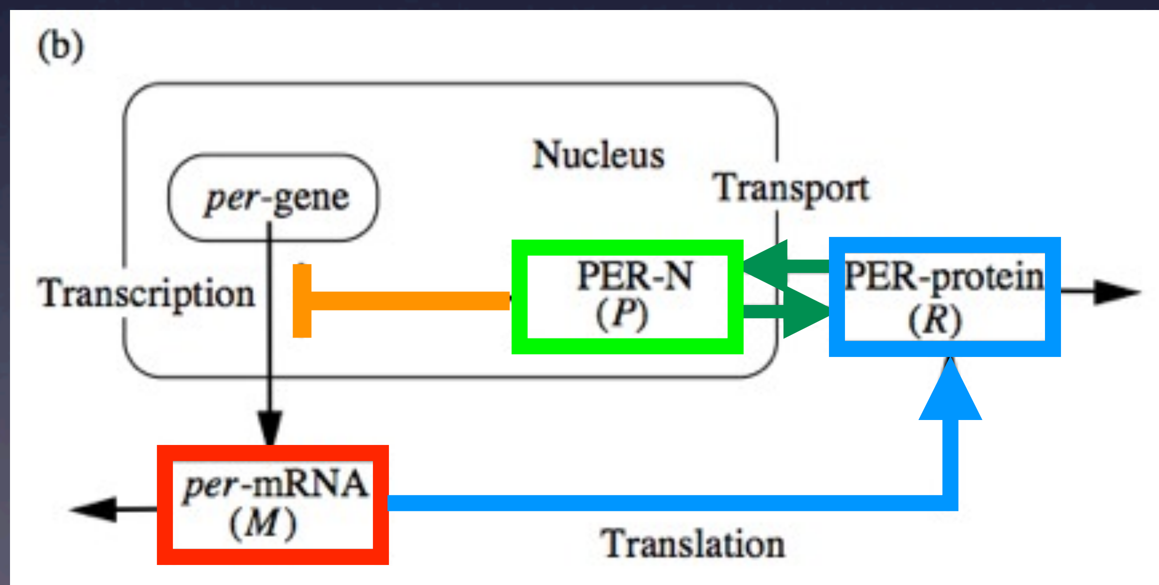
Circadian clock model

- 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

$$\frac{\mathrm{d}M}{\mathrm{d}t} = \frac{1}{1+(P/h)^n} - aM,$$

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

$$\frac{\mathrm{d}P}{\mathrm{d}t} = uR - vP.$$



Kurosawa et al. J. Theor. Biol. (2002) 216, 193-208

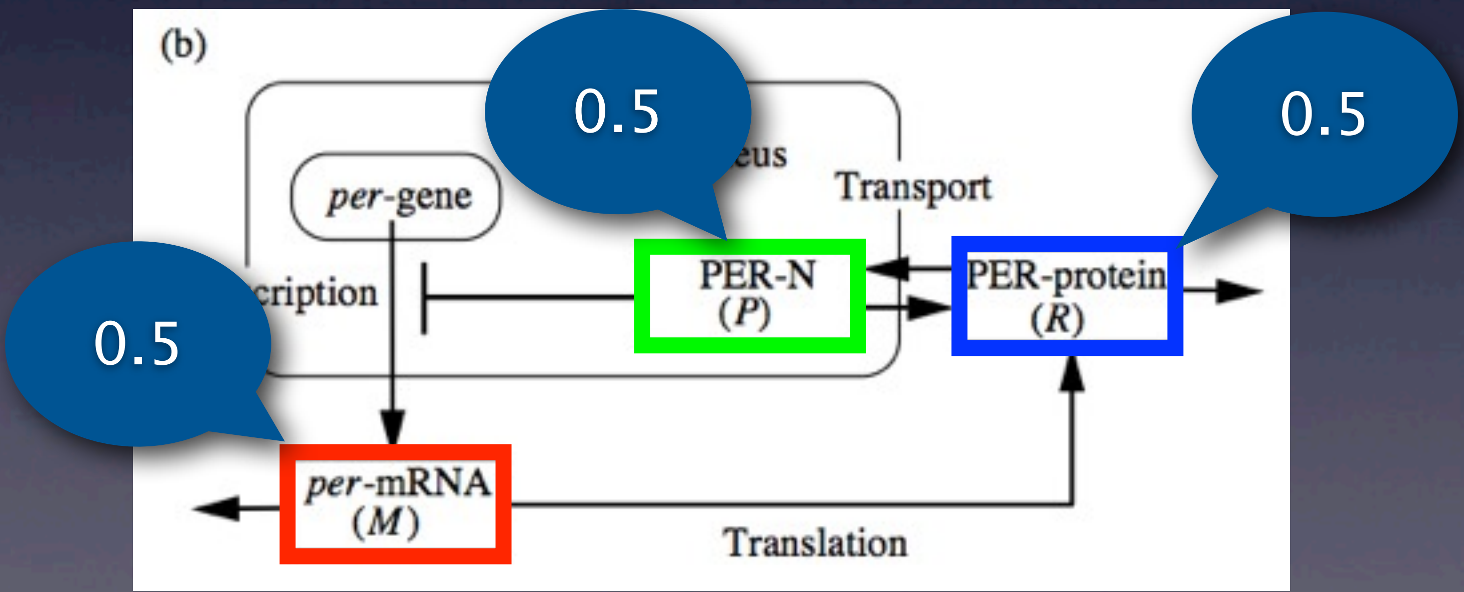
Circadian clock model

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

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

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

$$a = s = d = v = 1.0$$
$$u = 0.1$$
$$h = 0.01$$
$$n = 40$$



Kurosawa et al. J. Theor. Biol. (2002) 216, 193-208

Circadian clock model

$$\frac{\mathrm{d}M}{\mathrm{d}t} = \frac{1}{1+(P/h)^n} - aM,$$

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

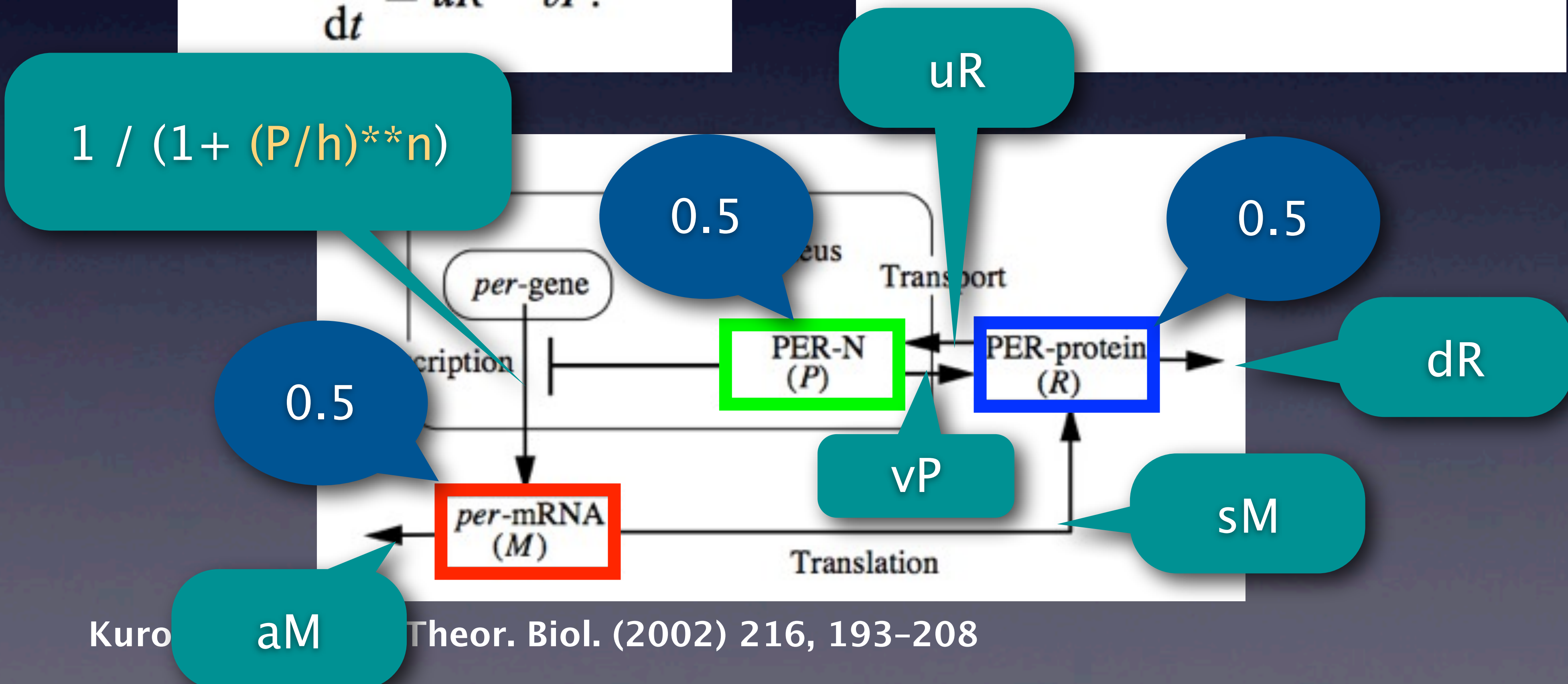
$$\frac{\mathrm{d}P}{\mathrm{d}t} = uR - vP.$$

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

$$u = 0.1$$

$$h = 0.01$$

$$n = 40$$

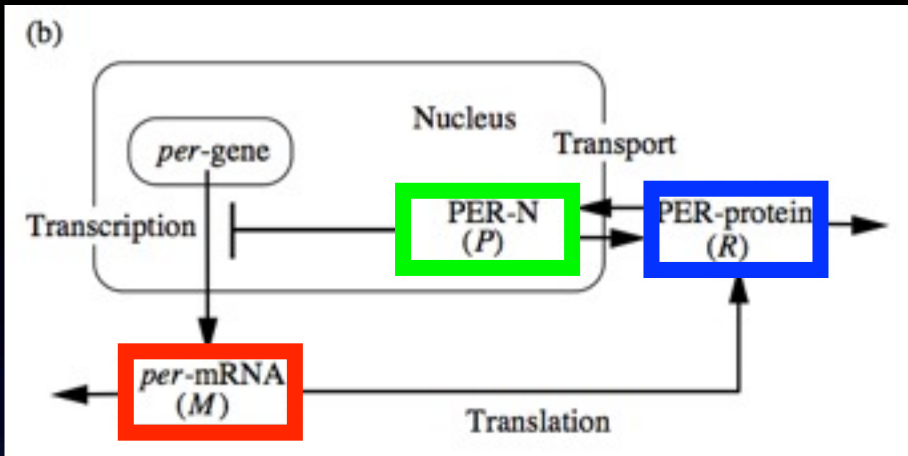


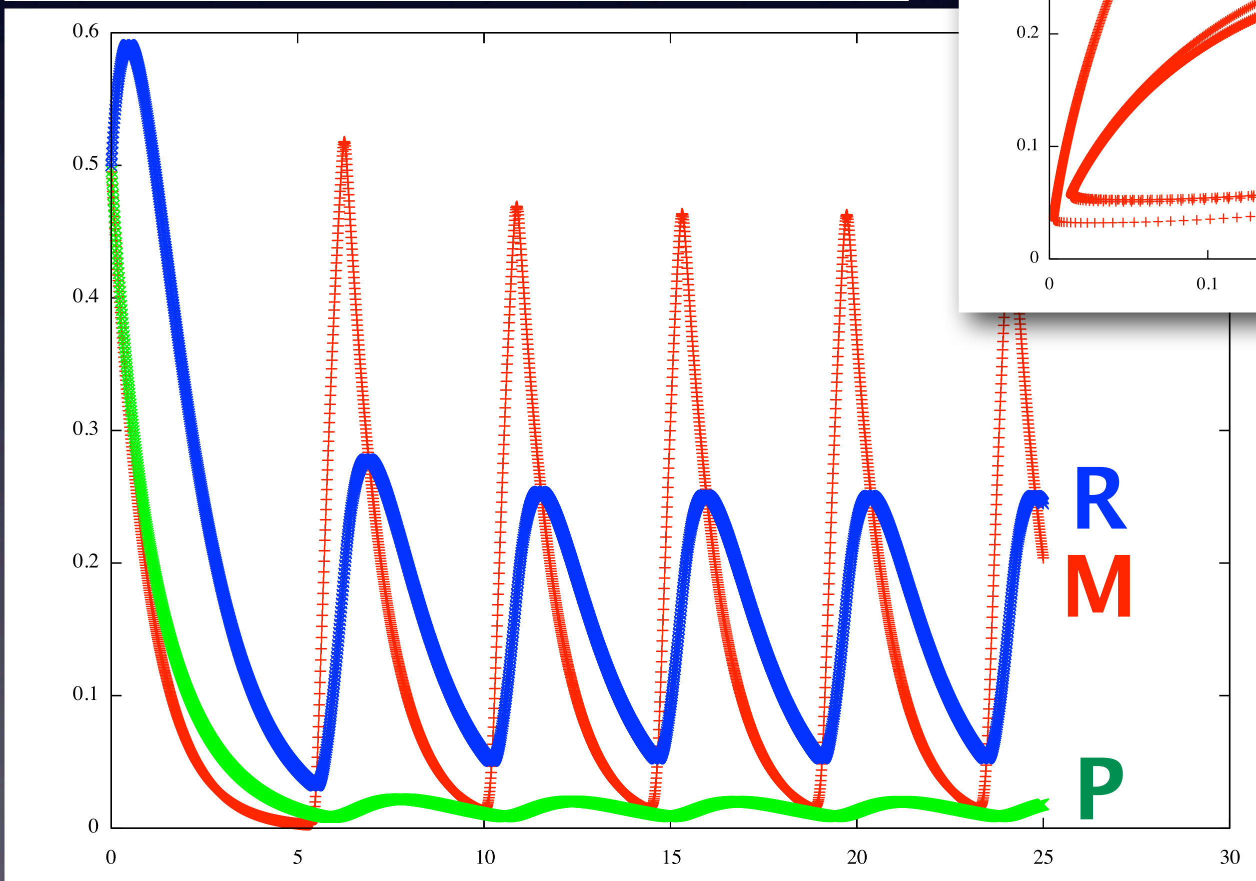
Kuro Theor. Biol. (2002) 216, 193–208

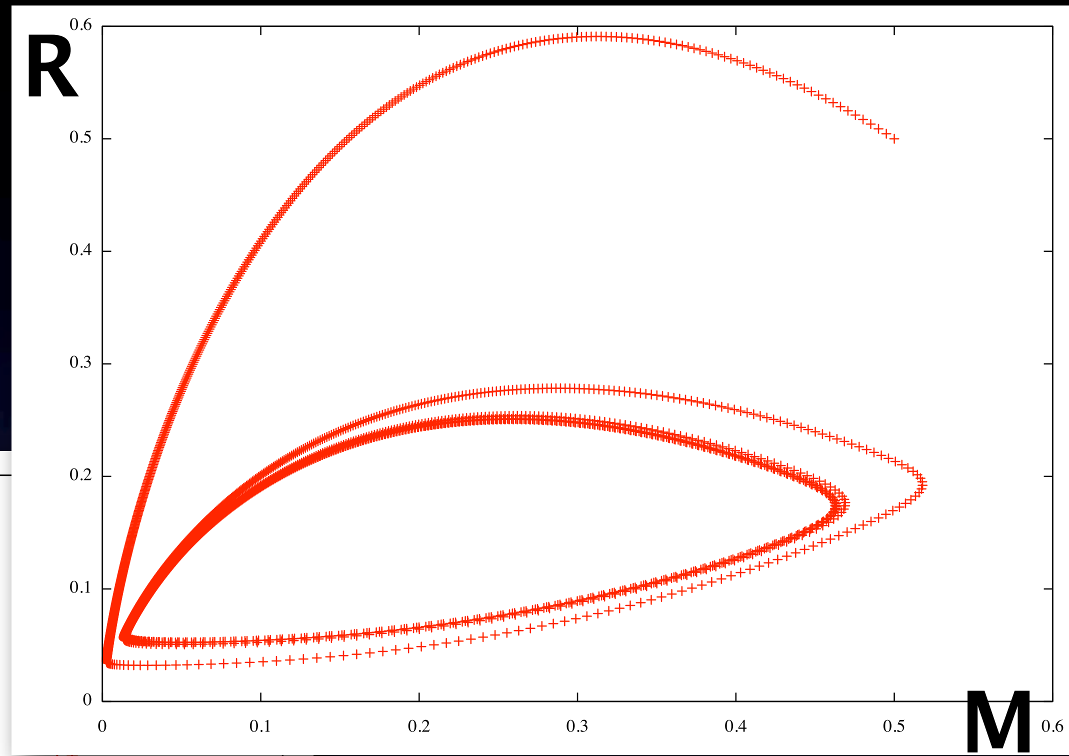
```perl
#!/usr/bin/perl

# 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;
}
```

Perl版

Circadian clock model

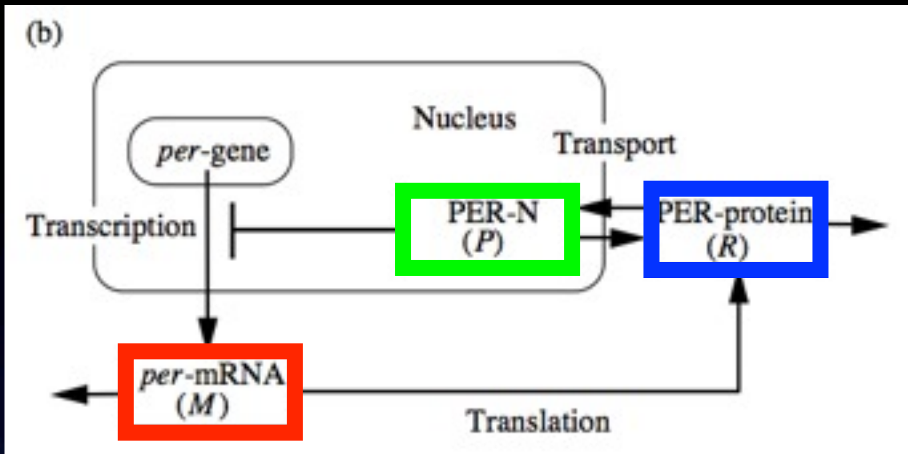


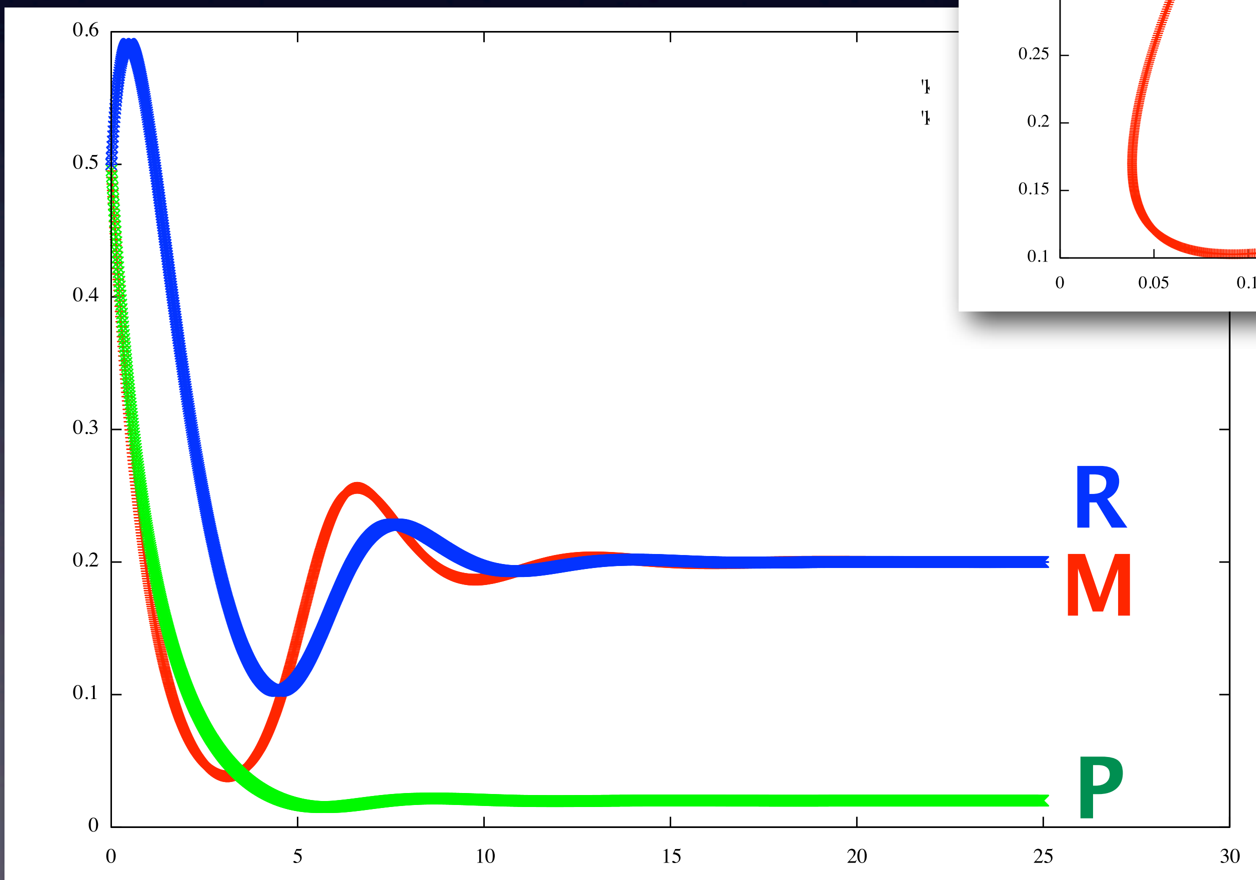


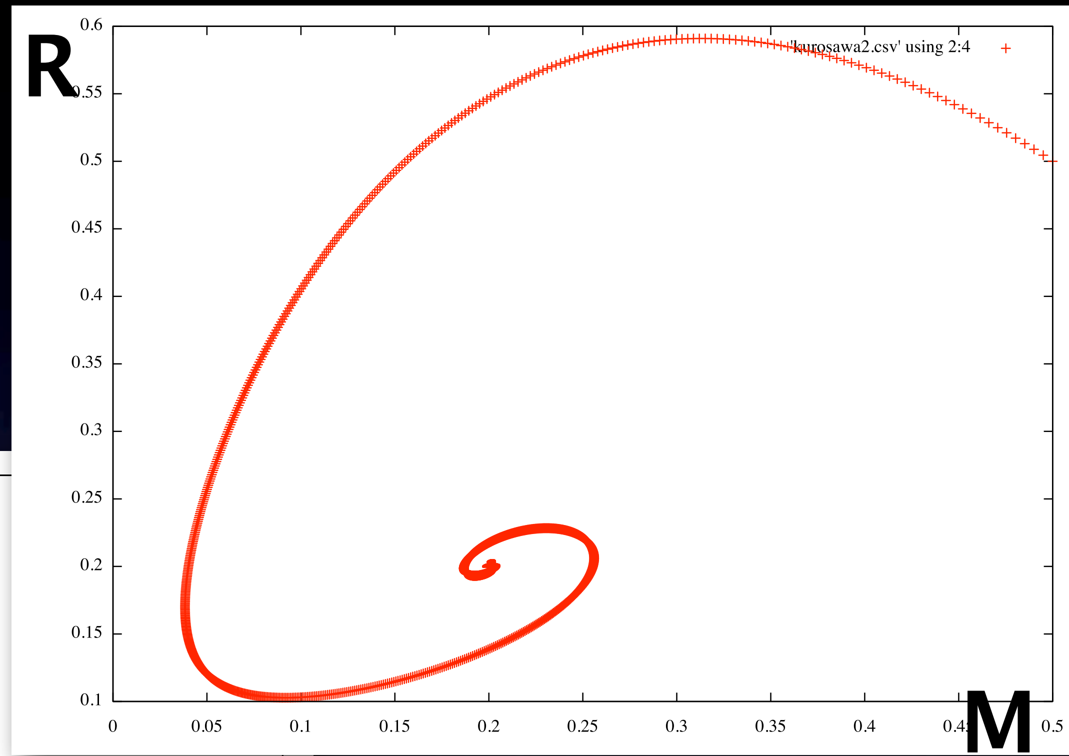


$n = 40$

Circadian clock model







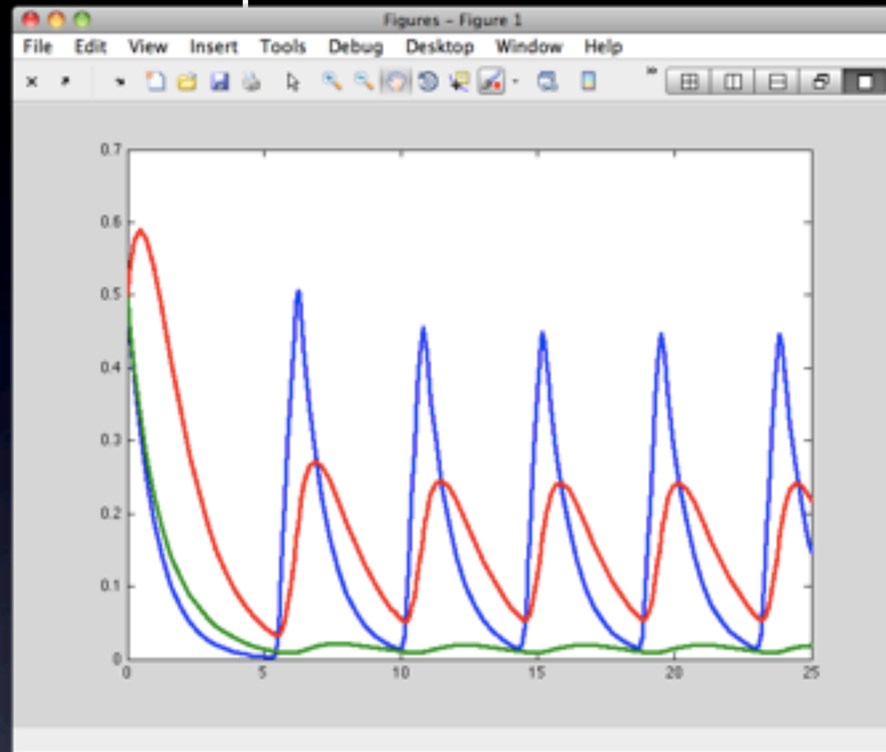
$$n = 2$$

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;
}
```

```
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;
```

MATLAB版



kurosawa.m

Octave-Forge
odepkg でも可

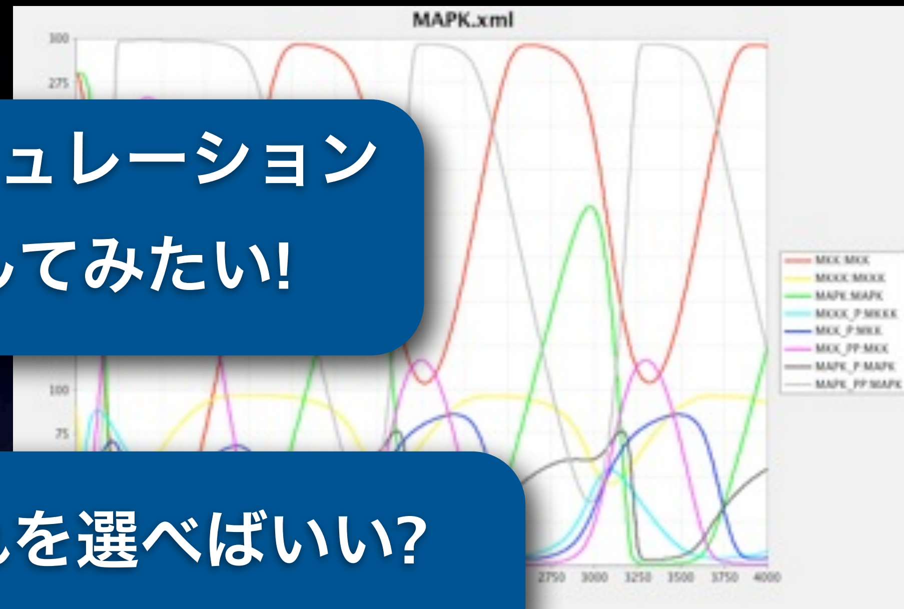
```
>> [t,x] = ode23(@kurosawa,[0 10],kurosawa);
>> plot(t,x)
```

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!

どれを選べばいい?



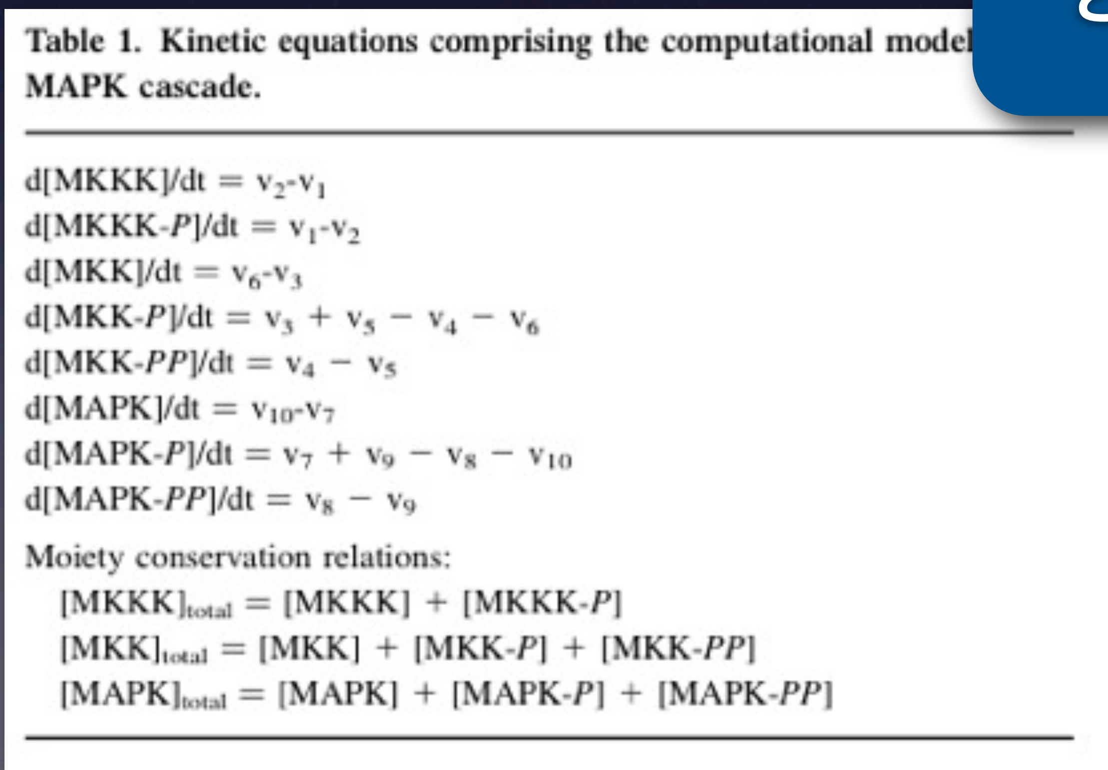
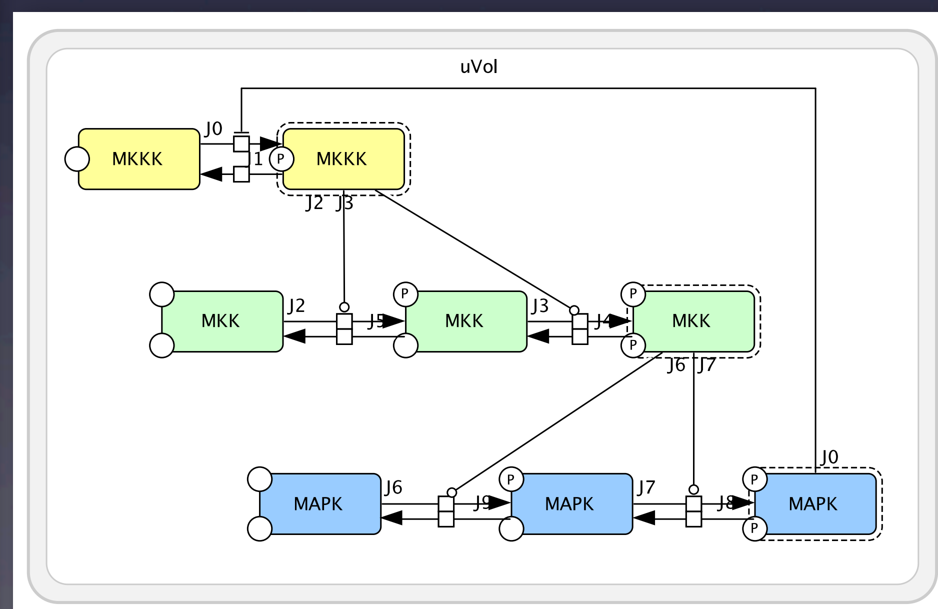


Table 1. Kinetic equations comprising the computational model MAPK cascade.

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

Moiety conservation relations:

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



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)
  - 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

丸め誤差

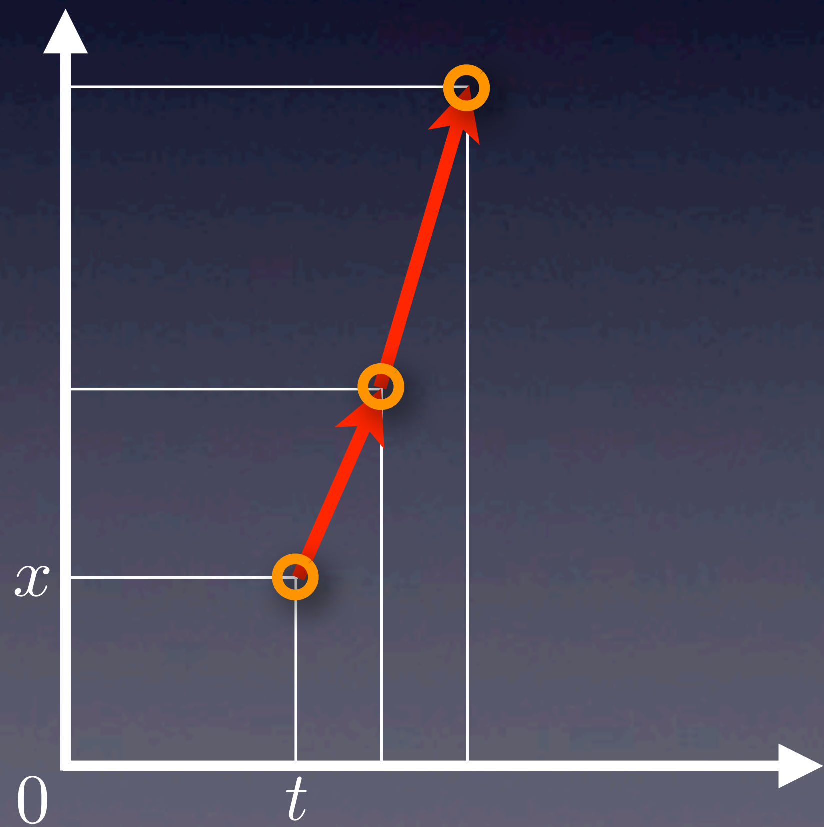
打ち切り誤差

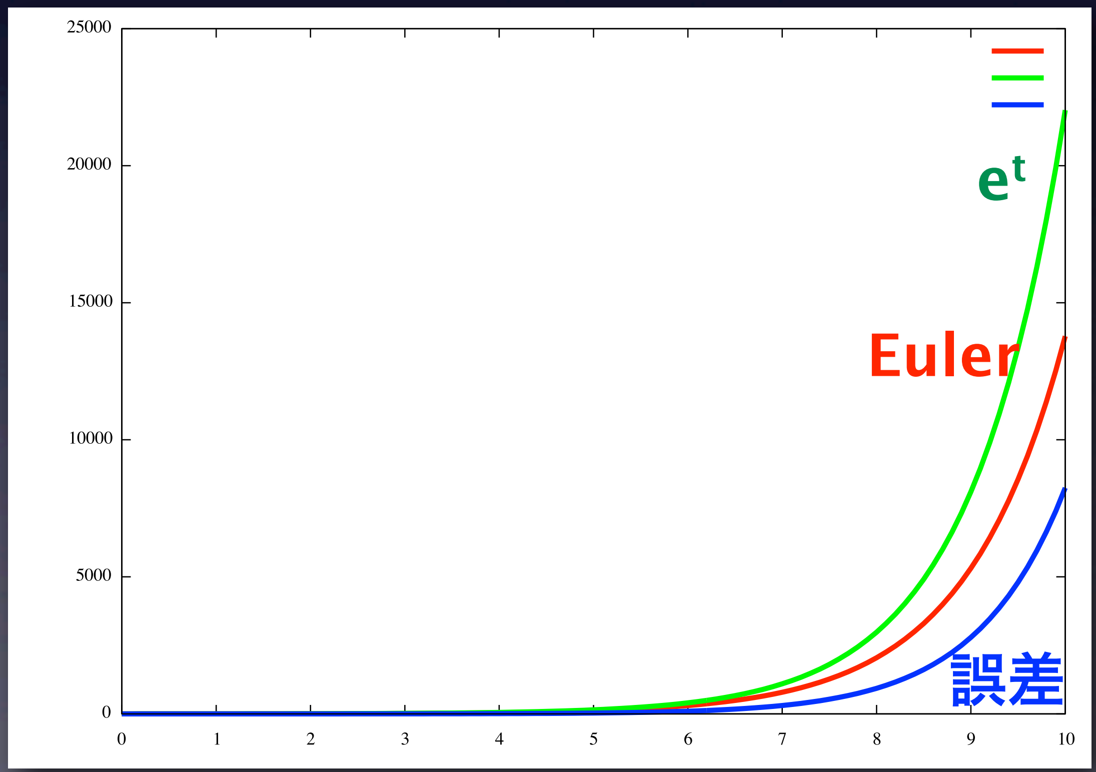
- 速度
  - 刻み幅(精度と計算時間のトレードオフ)
  - 高速化(アルゴリズム、言語、ハードウェア)

Euler法と誤差

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

$$x = e^t$$





Taylor展開とEuler法

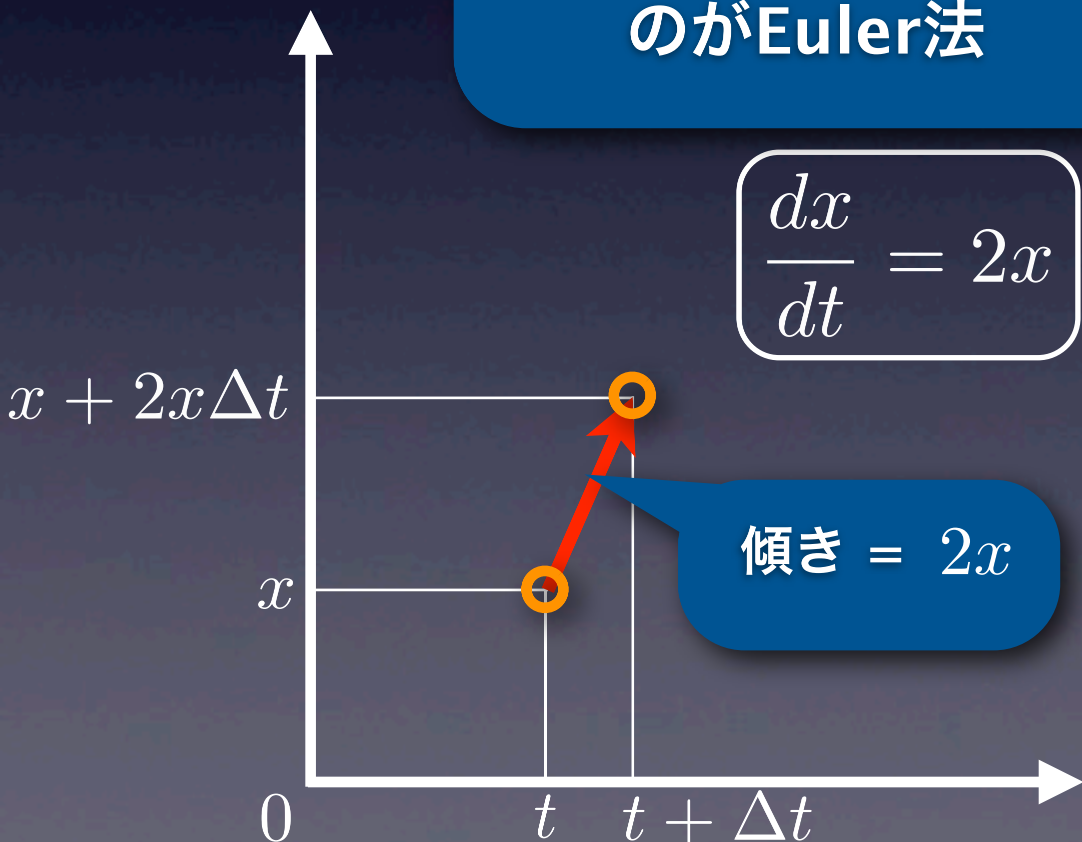
$$x(t+\Delta t) = x(t) + \Delta t x'(t) + \frac{1}{2}\Delta t^2 x''(t) + \frac{1}{3!}\Delta t^3 x^{(3)}(t) + \cdots$$

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

ここまでで打ち切ったのがEuler法

打ち切り誤差

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



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

Runge-Kutta法(4次)

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

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

打ち切り誤差

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

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

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

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

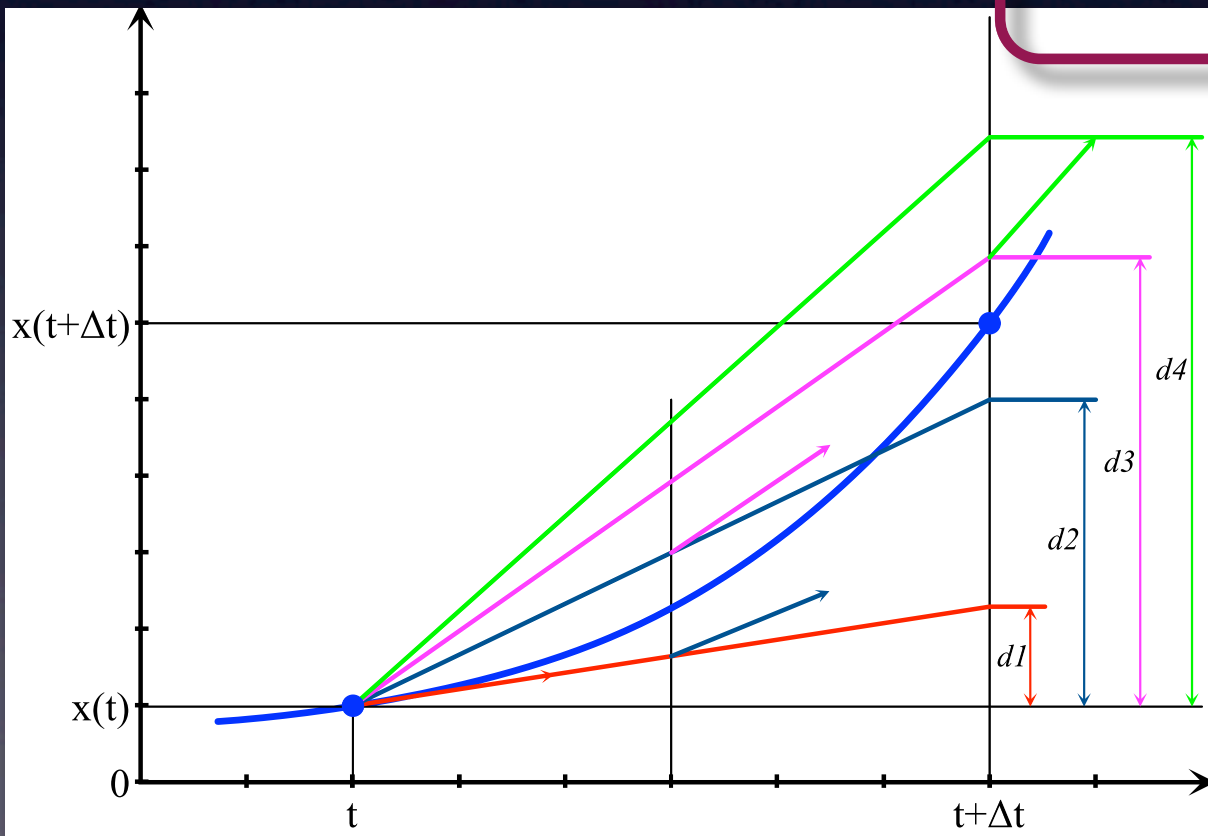
$$x(t+\Delta t) = x(t) + \frac{d_1 + 2d_2 + 2d_3 + d_4}{6}$$

Runge-Kutta法(4次)

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

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

打ち切り誤差



打ち切り誤差 = $O(\Delta t^5)$

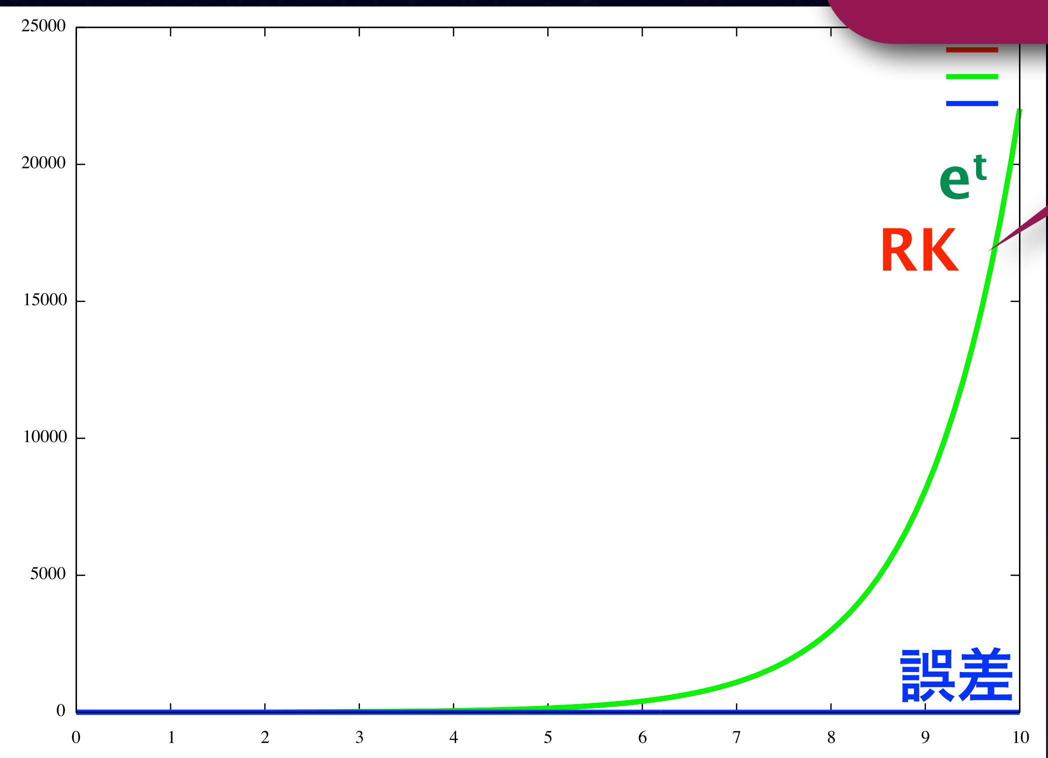
刻み幅を1/2にすれば
誤差は1/32

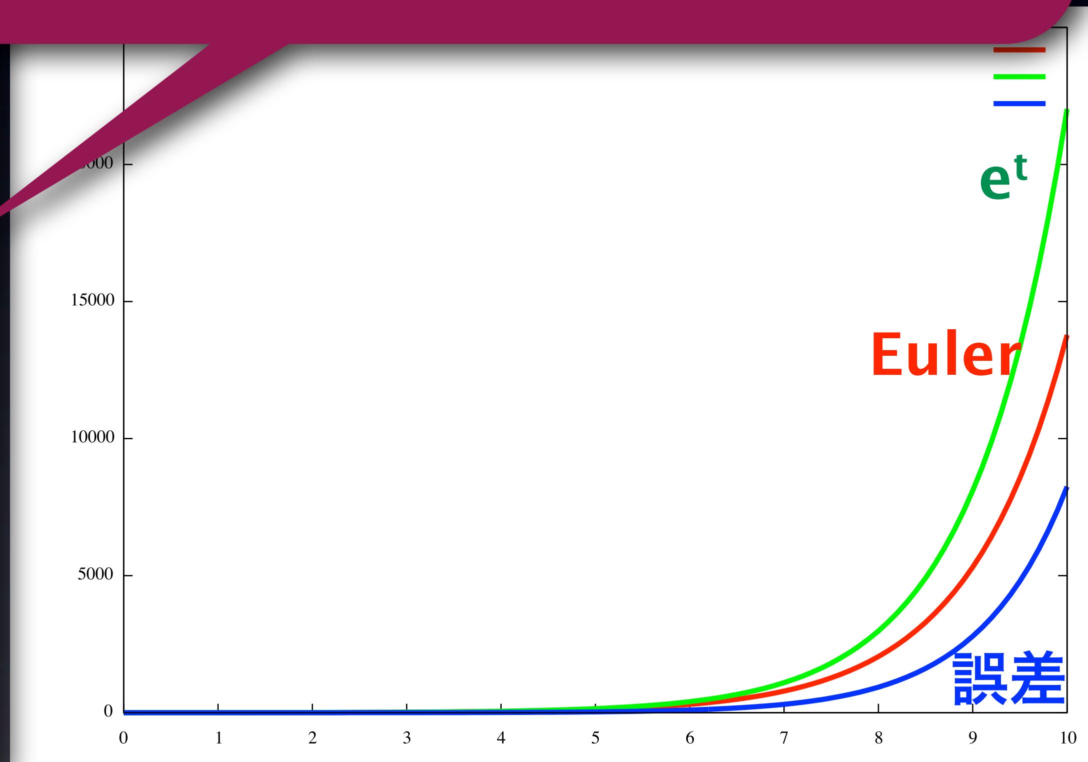
Runge-Kutta法と誤差

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

$$x = e^t$$

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

```perl
#!/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

```c
#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

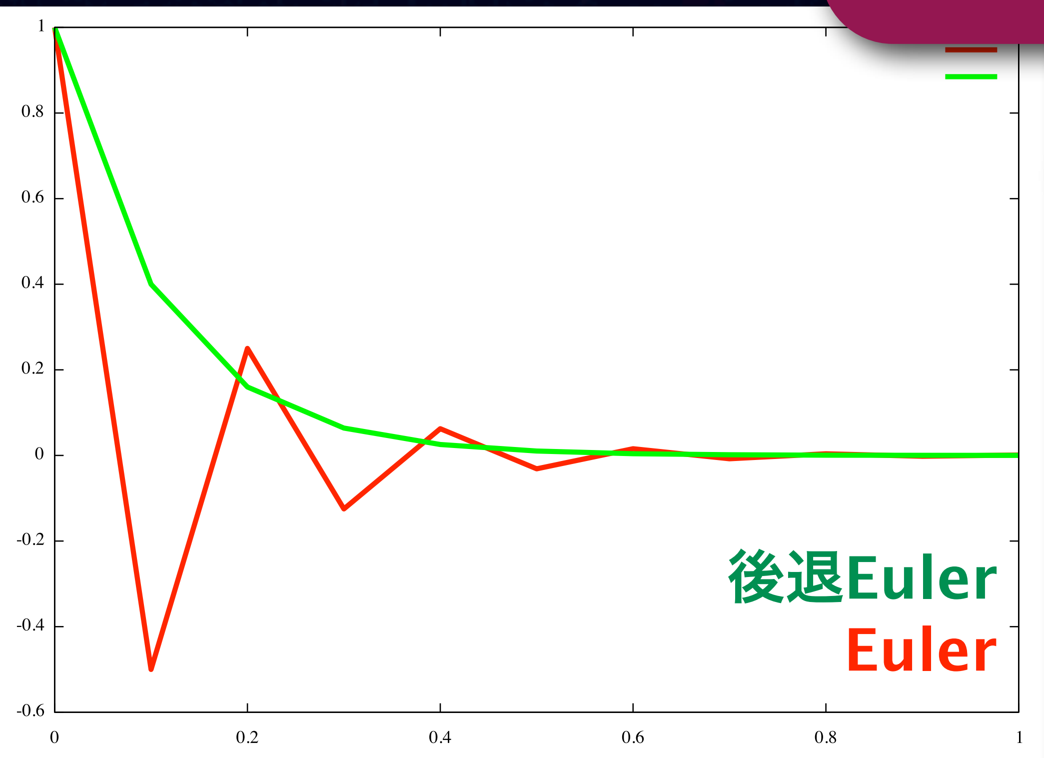
- 陰解法 ($x(t+\Delta t)$ を求めるのに $x(t+\Delta t)$ が必要)
  - 後退Euler法, Adams-Moulton法, Gear法
  - 硬い微分方程式に対応
  - 実装が面倒 (例: 数値微分, LU分解, ニュートン法, etc.)

Euler法と後退Euler法

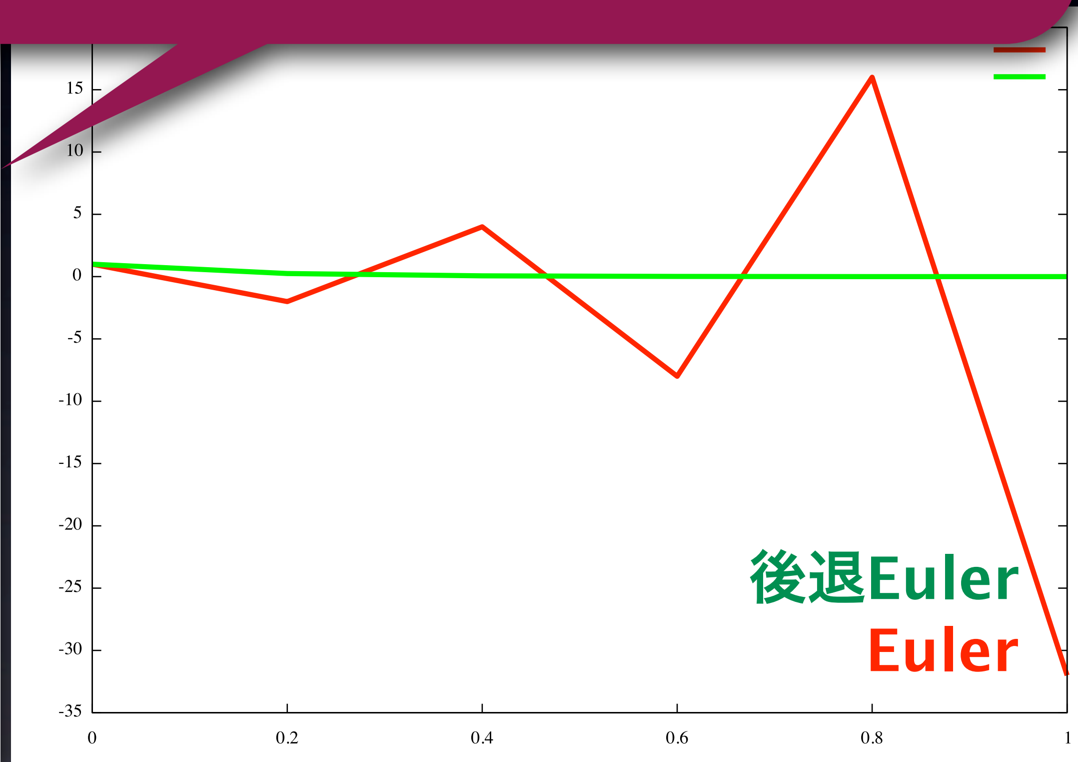
$$\frac{dx}{dt} = -15x \quad (t = 0,\ x = 1)$$

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

刻み幅によっては不安定になる、発散する

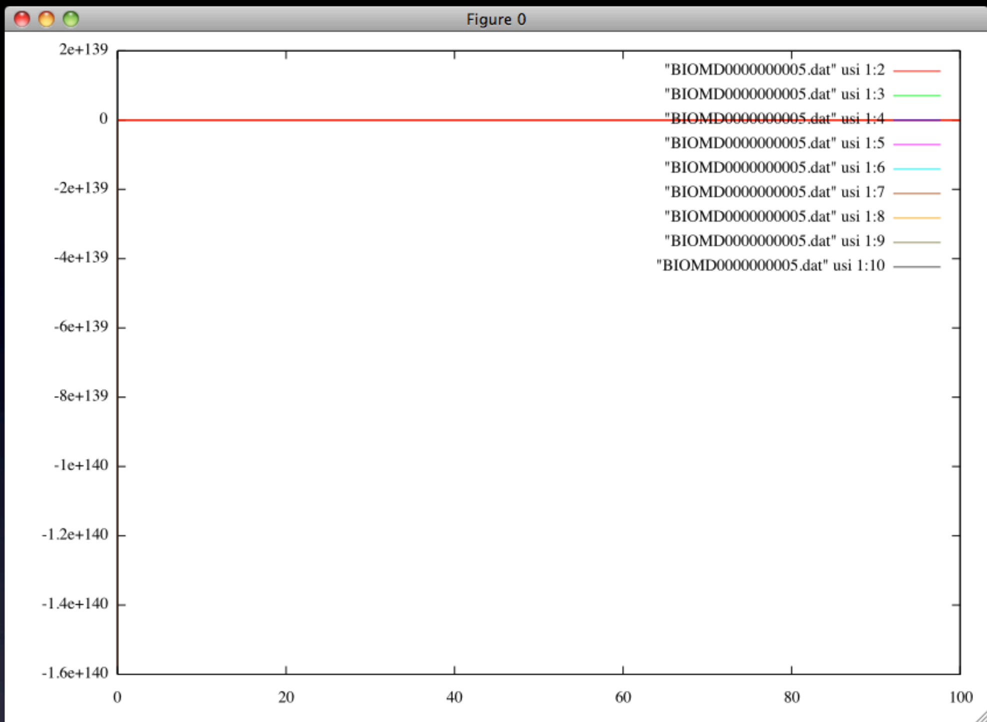


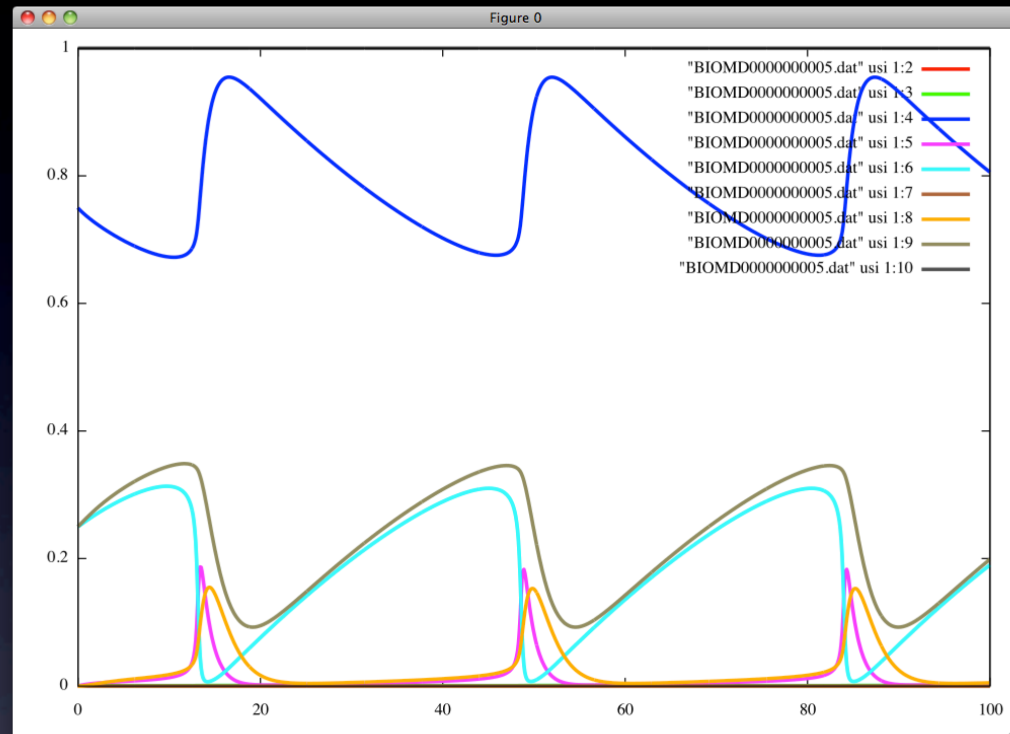
$\Delta t = 0.1$



$\Delta t = 0.2$

陽解法と陰解法





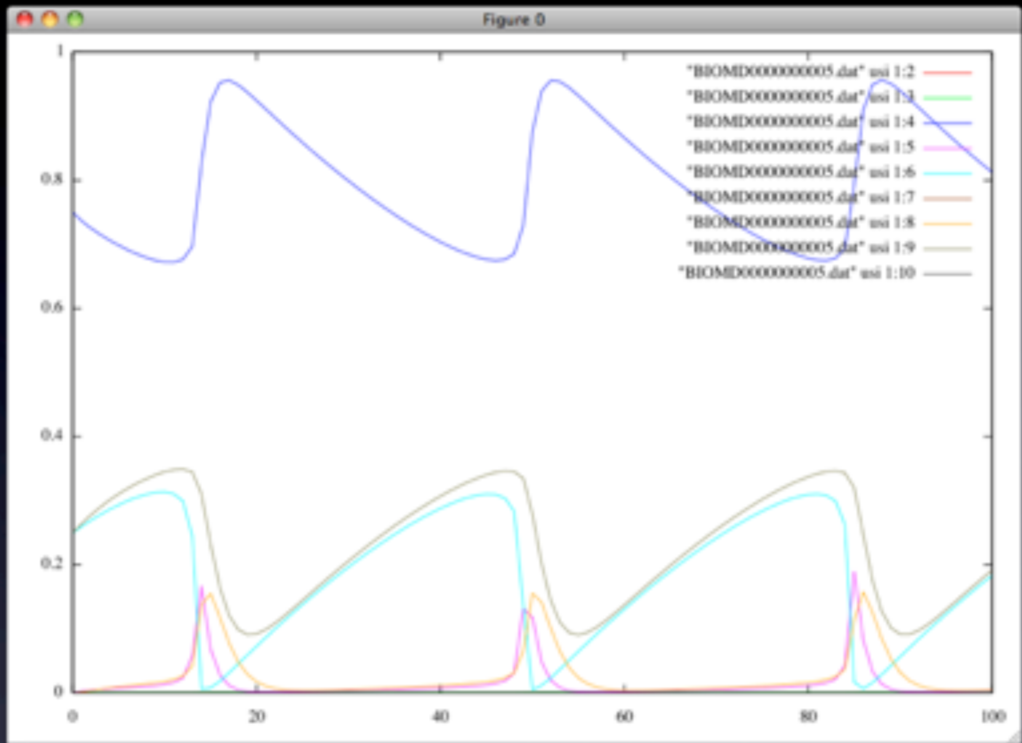
Runge-Kutta dt = 0.0001

Backward Euler dt= 0.1(RKの1,000倍)

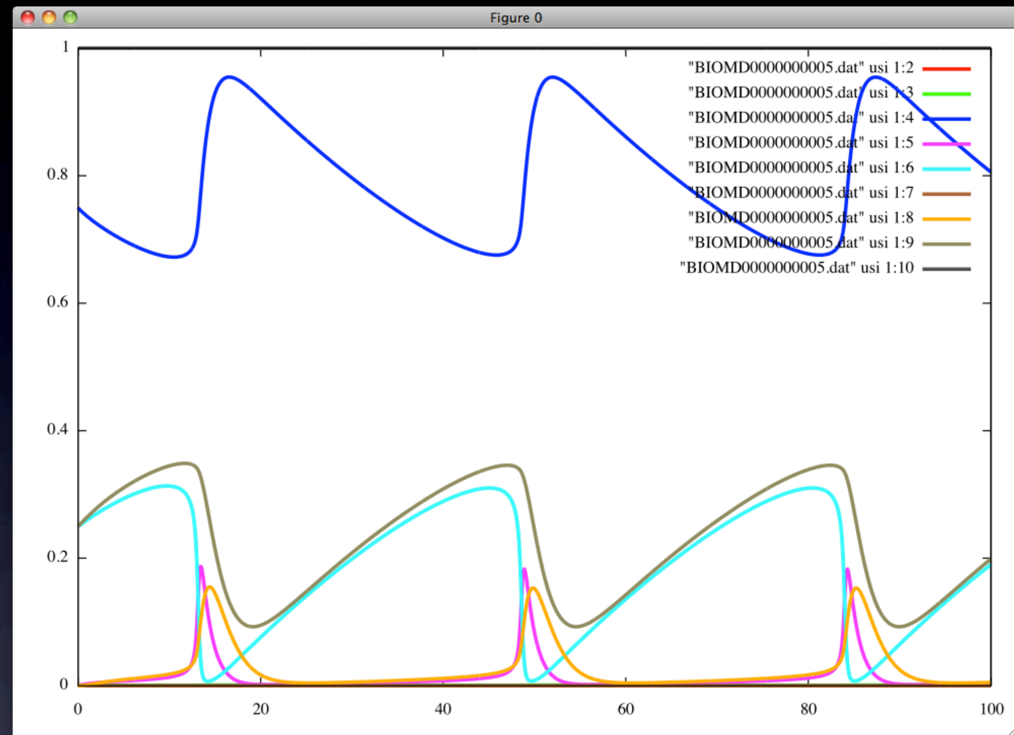
Tyson JJ.*PNAS* 1991 15;88(16):7328-32.
Modeling the cell division cycle: cdc2 and cyclin interactions.

陽解法と比べ1,000倍のステップ幅で正しい解を得ることが可能

陽解法(可変ステップ幅)と陰解法



RK435-2R 30M steps 184 sec.



Backward Euler dt= 0.1, 0.71 sec.

Tyson JJ.*PNAS* 1991 15;88(16):7328-32.
Modeling the cell division cycle: cdc2 and cyclin interactions.

方程式が急激な反応と緩やかな反応とを多数含む系の
挙動を表すとき、時間前進積分は難しくなる
~H.H.Robertson~

数値計算での注意点

- 精度
  - 浮動小数点数 (単精度と倍精度)
  - 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

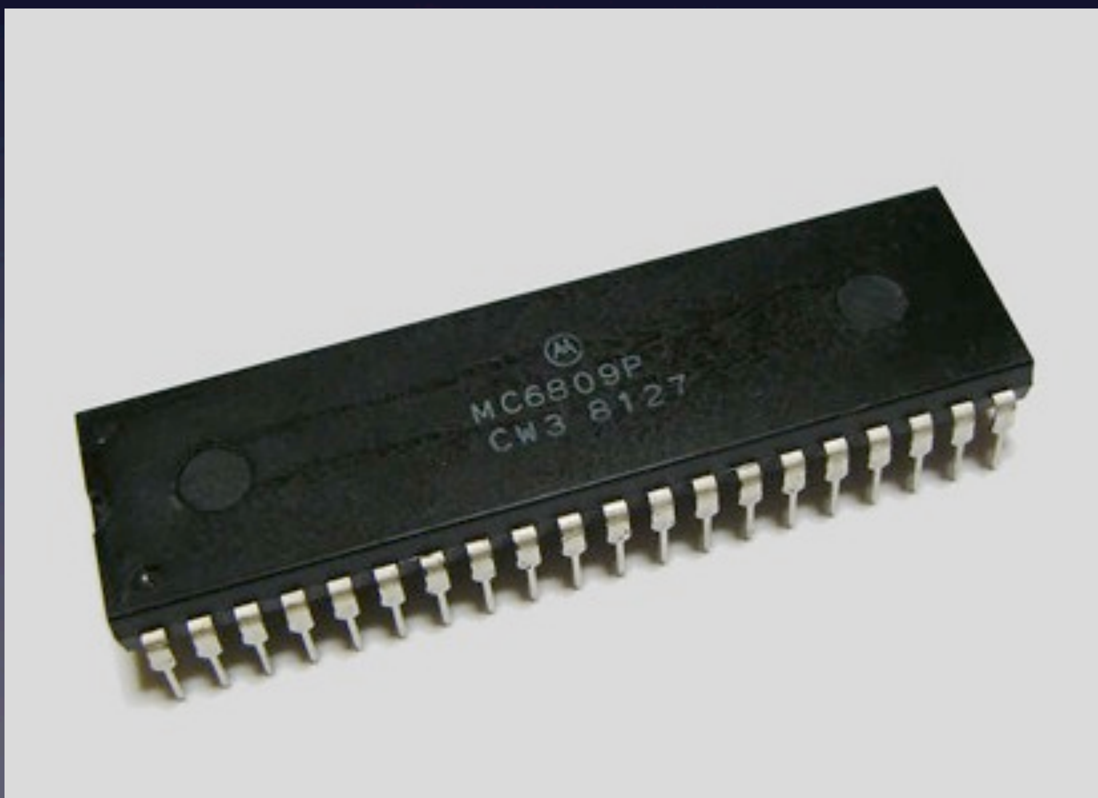
丸め誤差

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)

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 x 8 + 0 x 4 + 1 x 2 + 1 x 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

```c
#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**
  - $123400000 = 1.234 \times 10^{8}$
  - $0.001234 = 1.234 \times 10^{-3}$
- **2進数**
  - $11011000000 = 1.1011 \times 2^{10}$
  - $0.0000010101 = 1.0101 \times 2^{-6}$

仮数

指数

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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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}	2^{-2}	2^{-3}	2^{-4}	2^{-5}		2^{-22}	2^{-23}
		0	1	0	1	0	...	0	0

$$2^{-2} + 2^{-4} = 0.25 + 0.0625$$
$$= 0.3125$$

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

$$0.1_{(10)} = 0.0\,0011\,0011\,0011\,0011\ldots_{(2)}$$

```c
#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

$$0.1_{(10)} = 0.0\,0011\,0011\,0011\,0011\ldots_{(2)}$$

```c
#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

$$0.1_{(10)} = 0.0\,0011\,0011\,0011\,0011\ldots_{(2)}$$

```
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

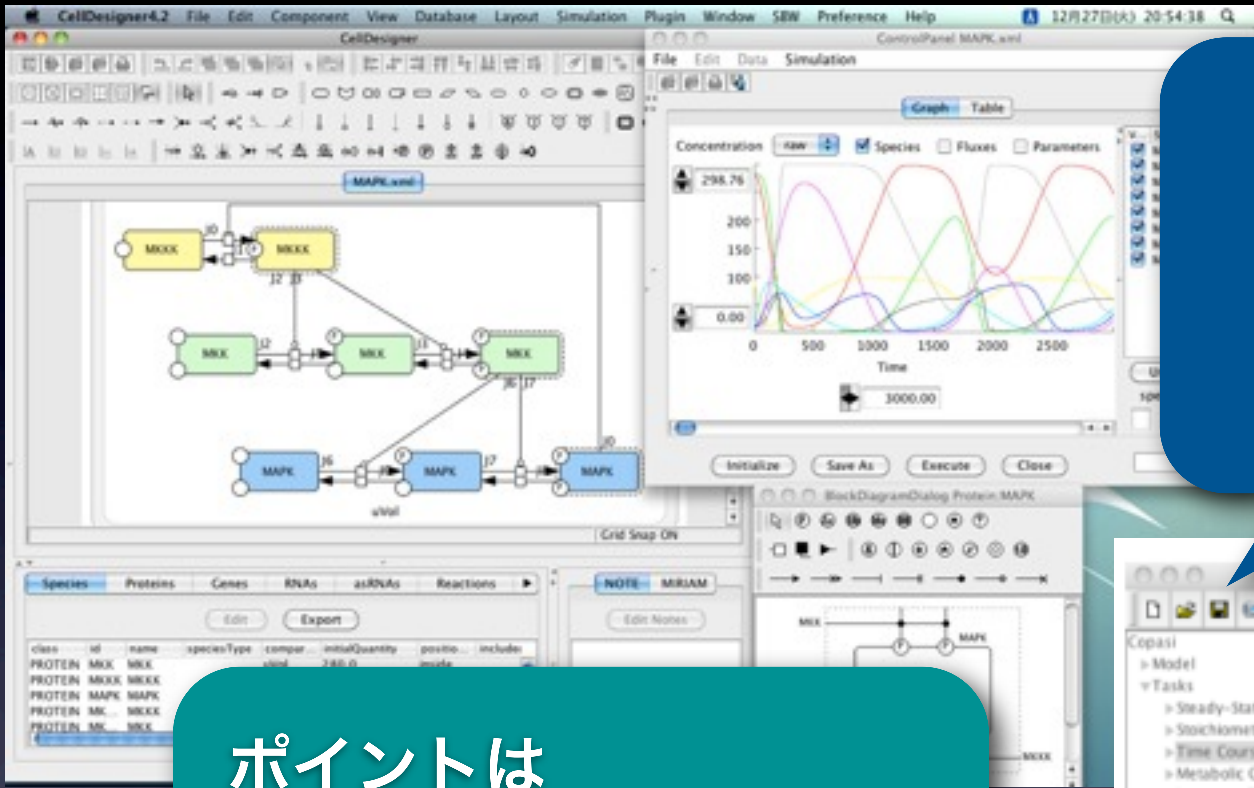
```
z = 9.007199254740996E15
d = 2.0
```

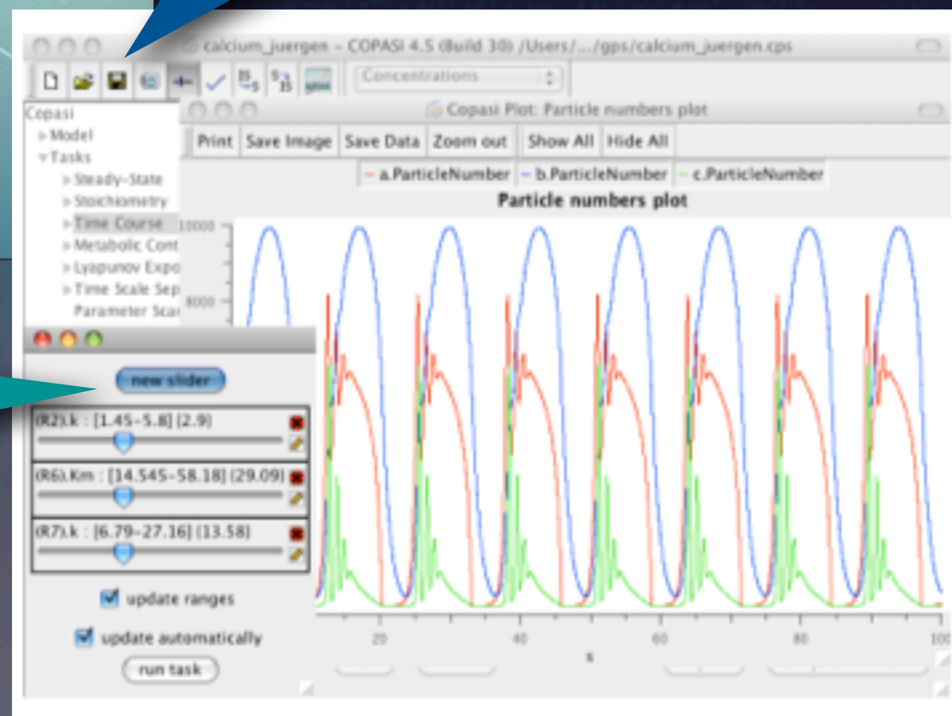
NG

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

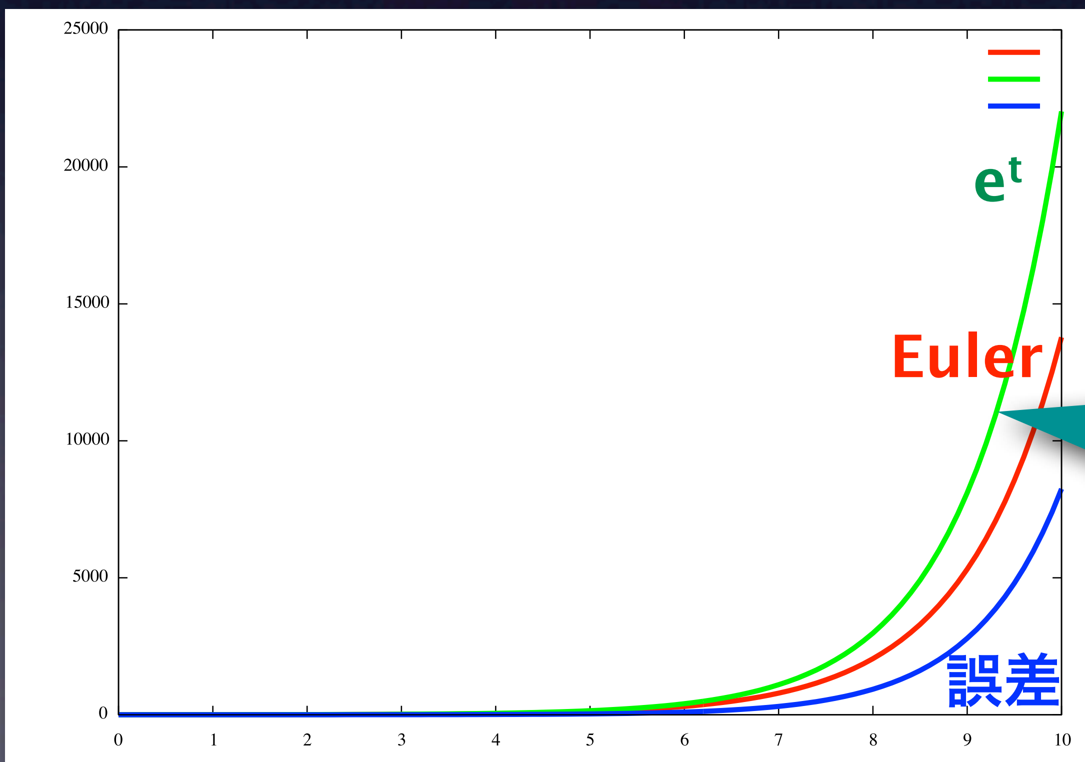
1. 精度
2. 使いやすさ
3. モデルを再利用可能か

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

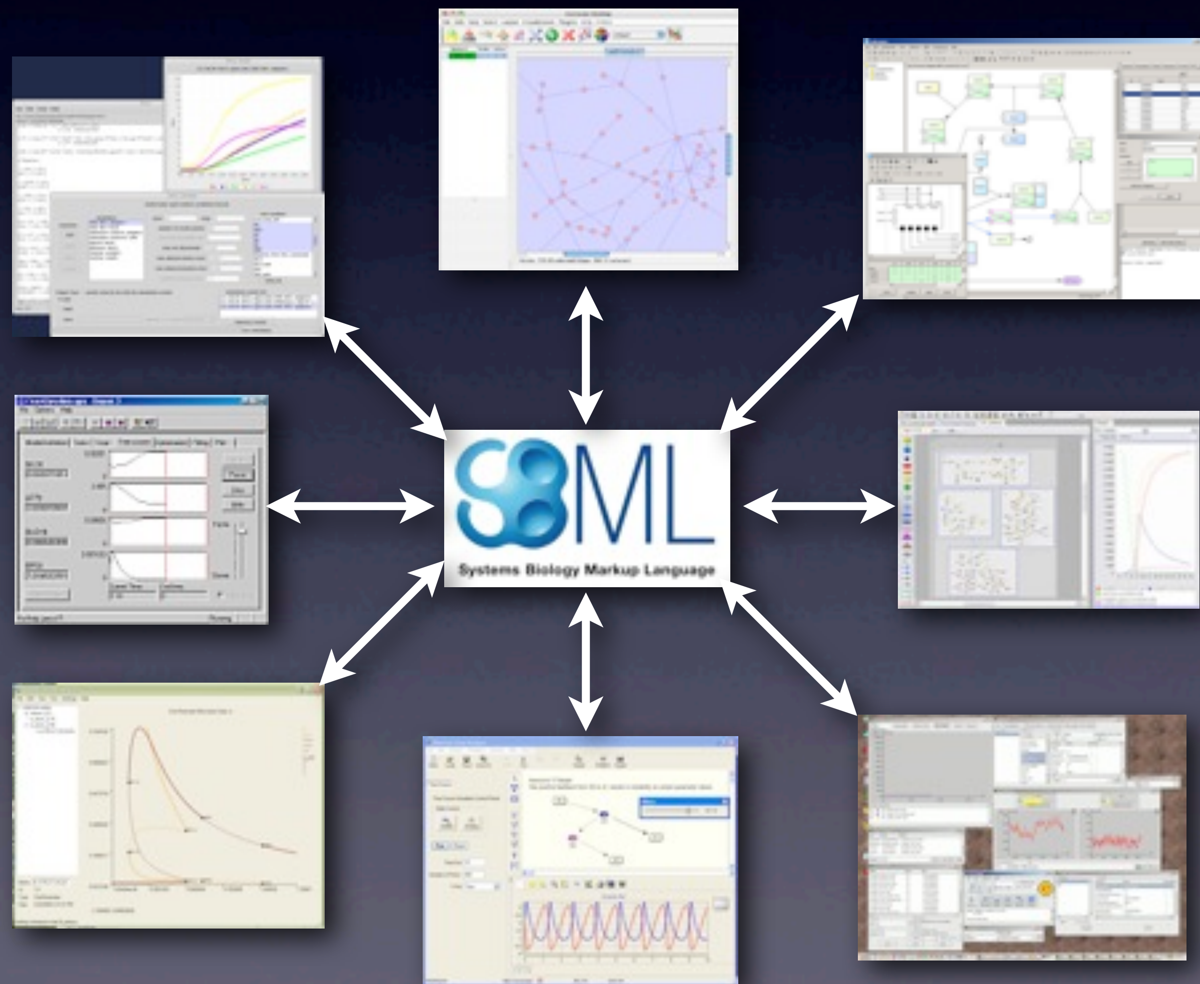


* SUNDIALS (CVODE)
* LSODA
* GSL 等

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

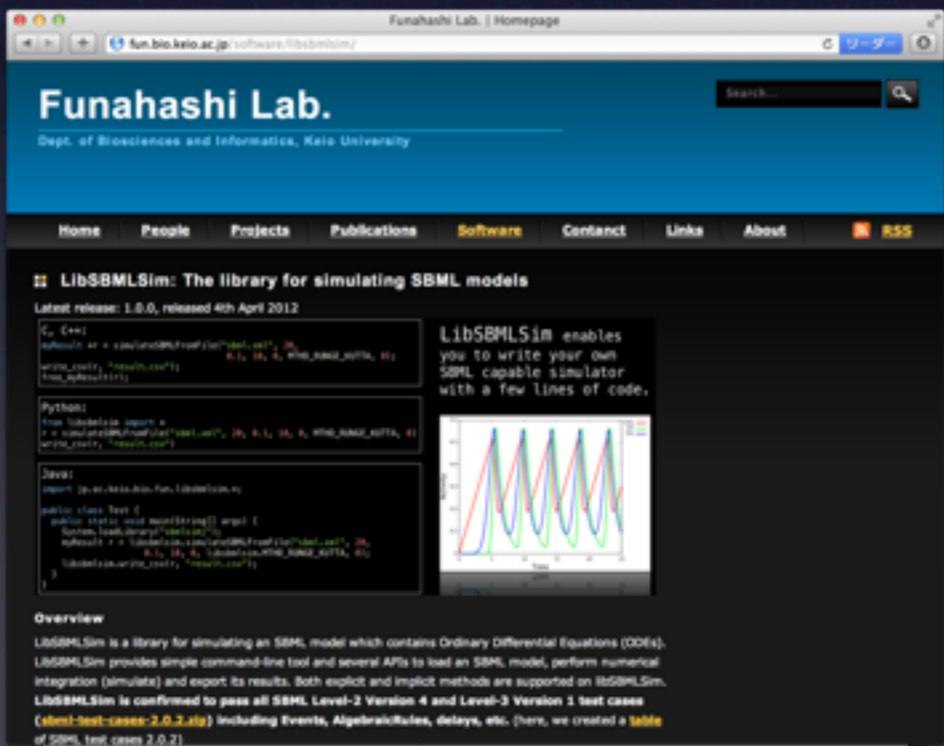
モデルの再利用

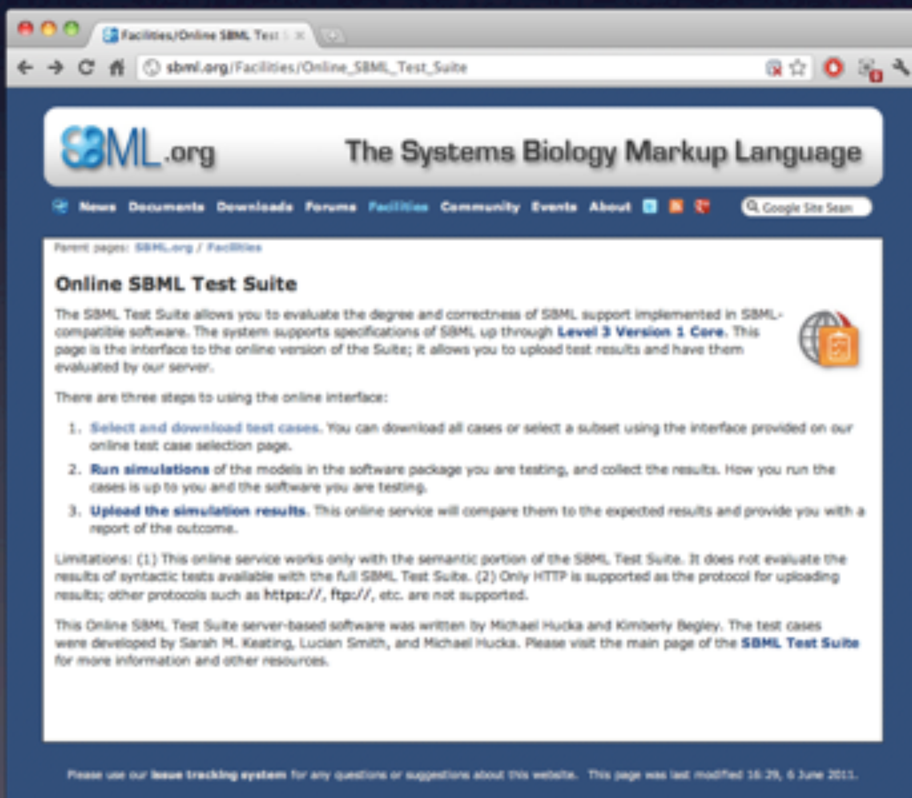
- 230以上のSBML対応ソフトウェア
- http://sbml.org



機能

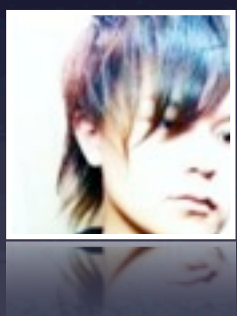
- 常微分方程式の初期値問題を解くだけじゃない
  - 代数微分方程式、イベント、遅延微分方程式、etc.
  - SBMLのすべての仕様を満たす(テストをパスした)
    シミュレータは世界に2つだけ



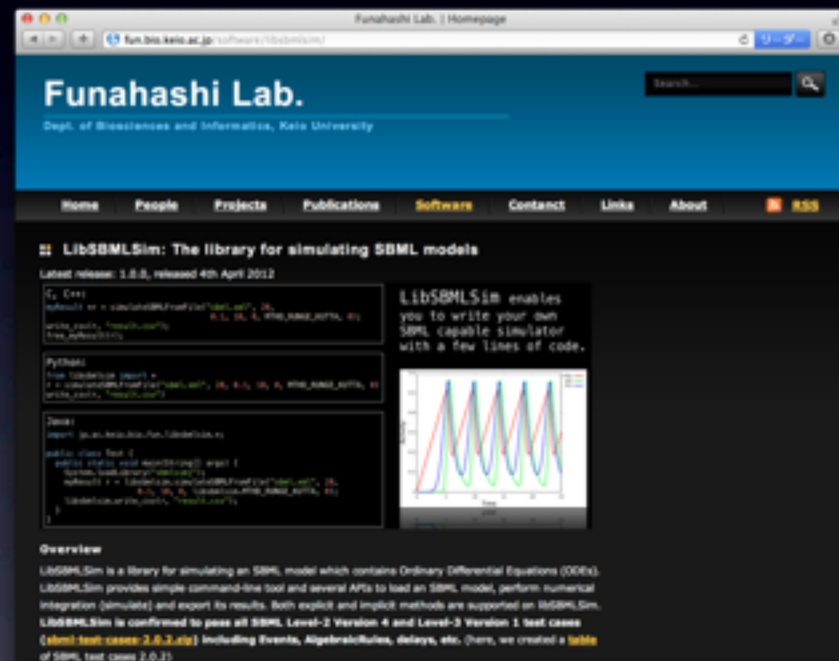


LibSBMLSim

- C, C++, Perl, Python, Ruby, Javaから利用可能
- 陽解法 + 陰解法
- 可変ステップ幅は絶賛実装中



by 近原鷹一 (P-117)

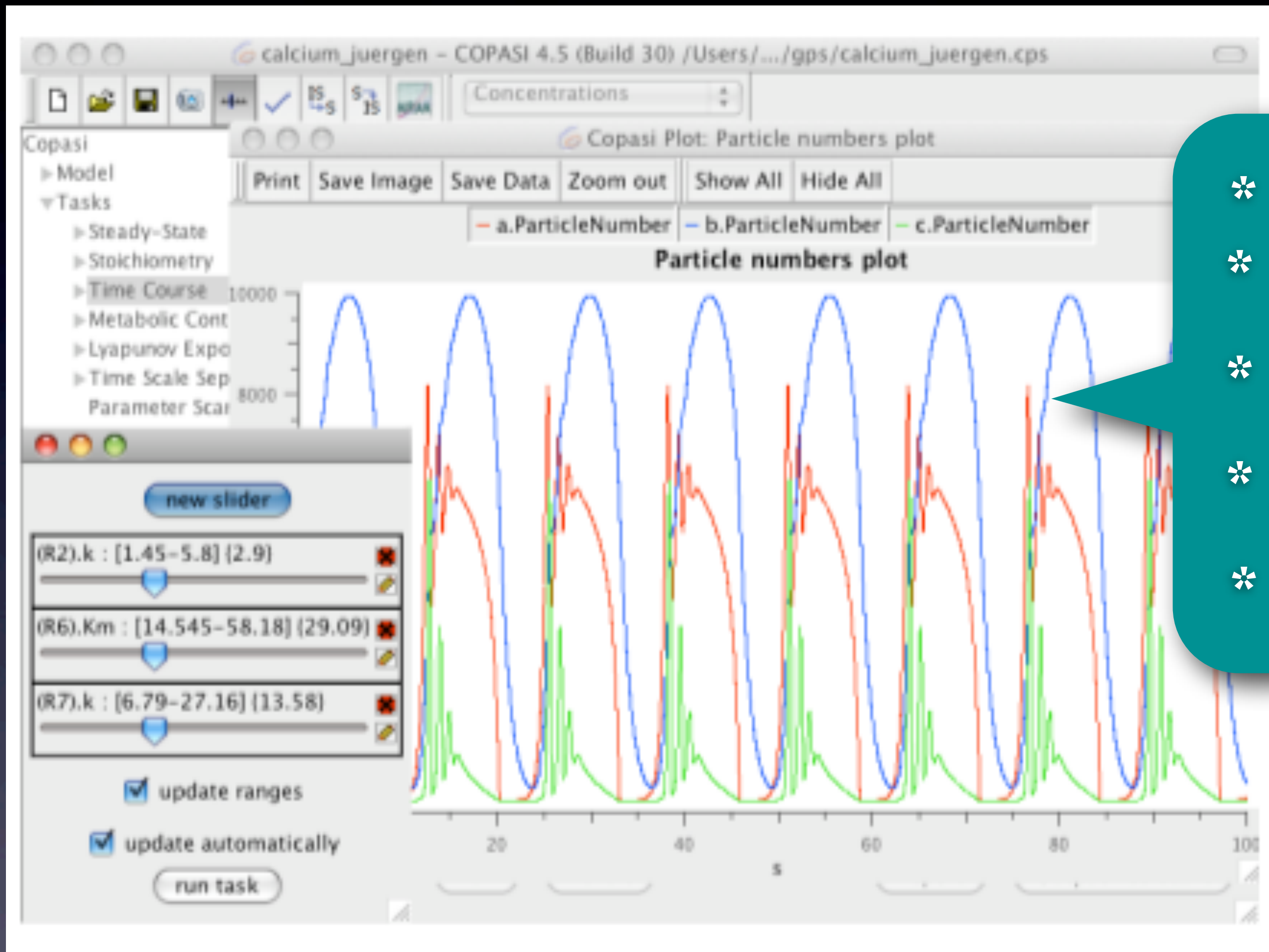



```
# Example Python code
from libsbmlsim import *
r = simulateSBMLFromFile('sbml.xml', 20, 0.1, 10, 0, MTHD_RUNGE_KUTTA, 0)
write_csv(r, 'result.csv')
```

http://fun.bio.keio.ac.jp/software/libsbmlsim

シミュレータの紹介

COPASI

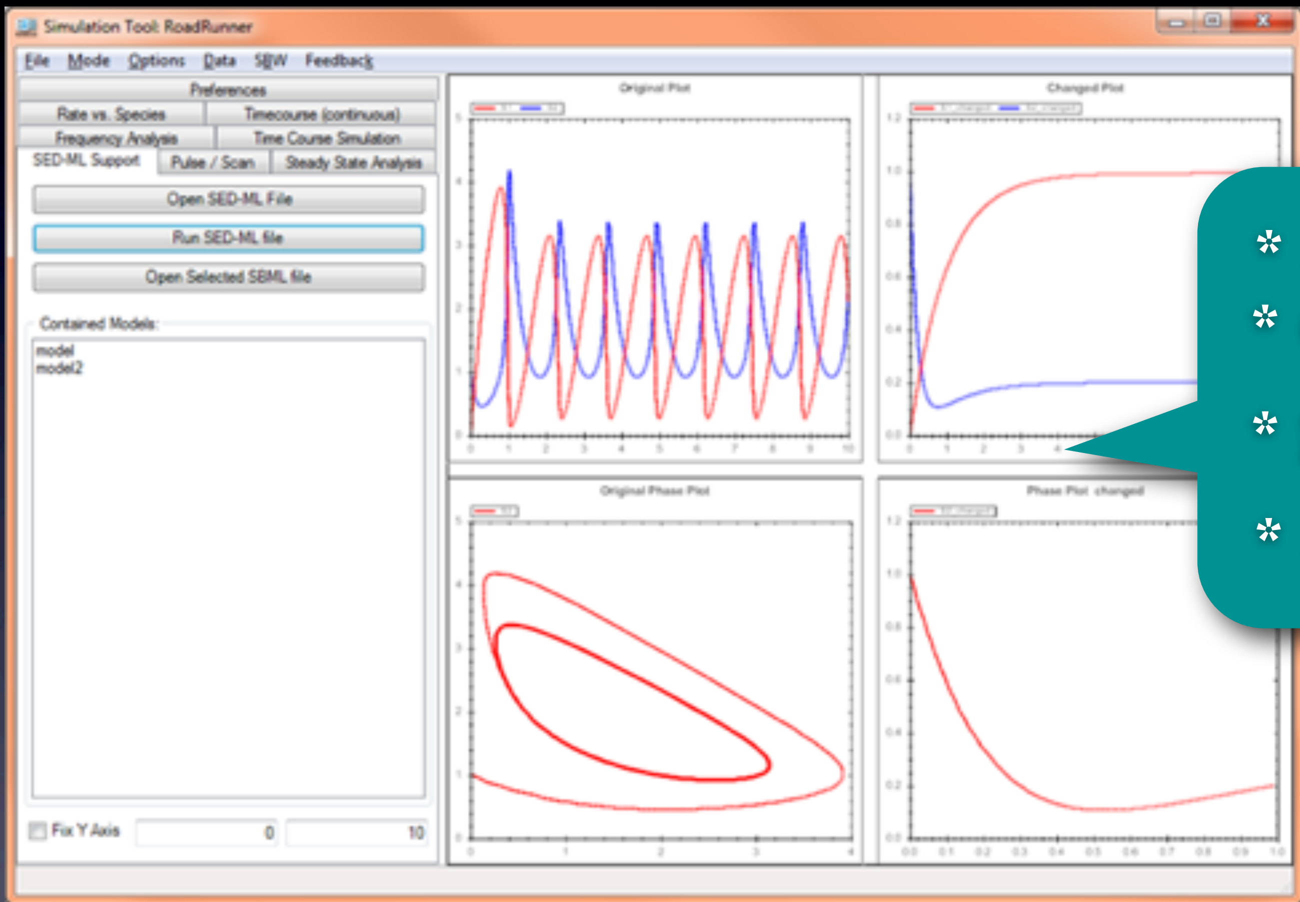


* ODE, SSA
* LSODA (陰解法もOK)
* パラメータフィッティング
* 解析
* C++

Univ. of Manchester, Univ. of Heidelberg, VBI

http://copasi.org/

RoadRunner

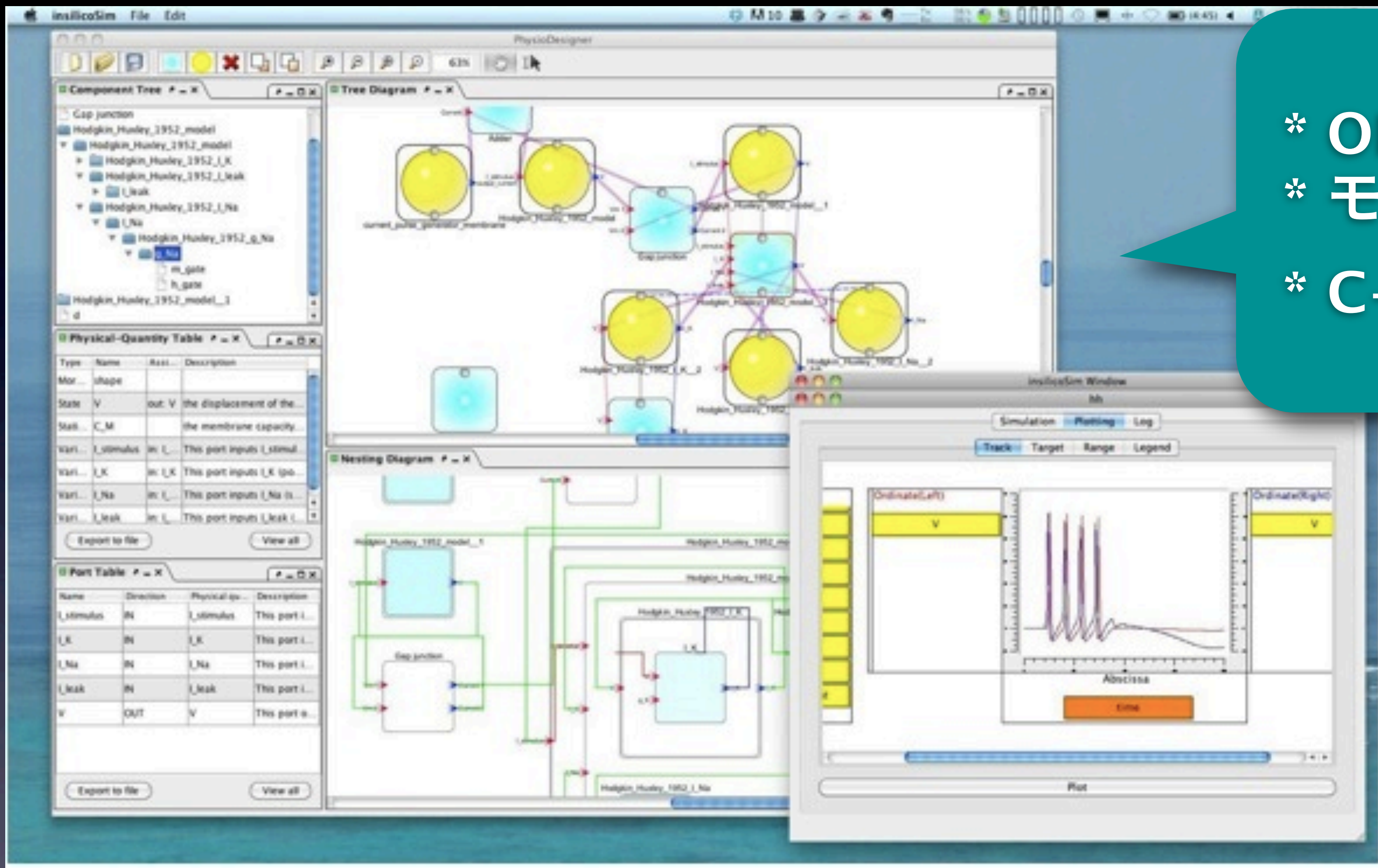


* ODE
* CVODE (陰解法もOK)
* 解析ツールと連動
* C# (.NET, Mono)

Univ. of Washington, Caltech

http://sf.net/projects/roadrunner

PhysioDesigner

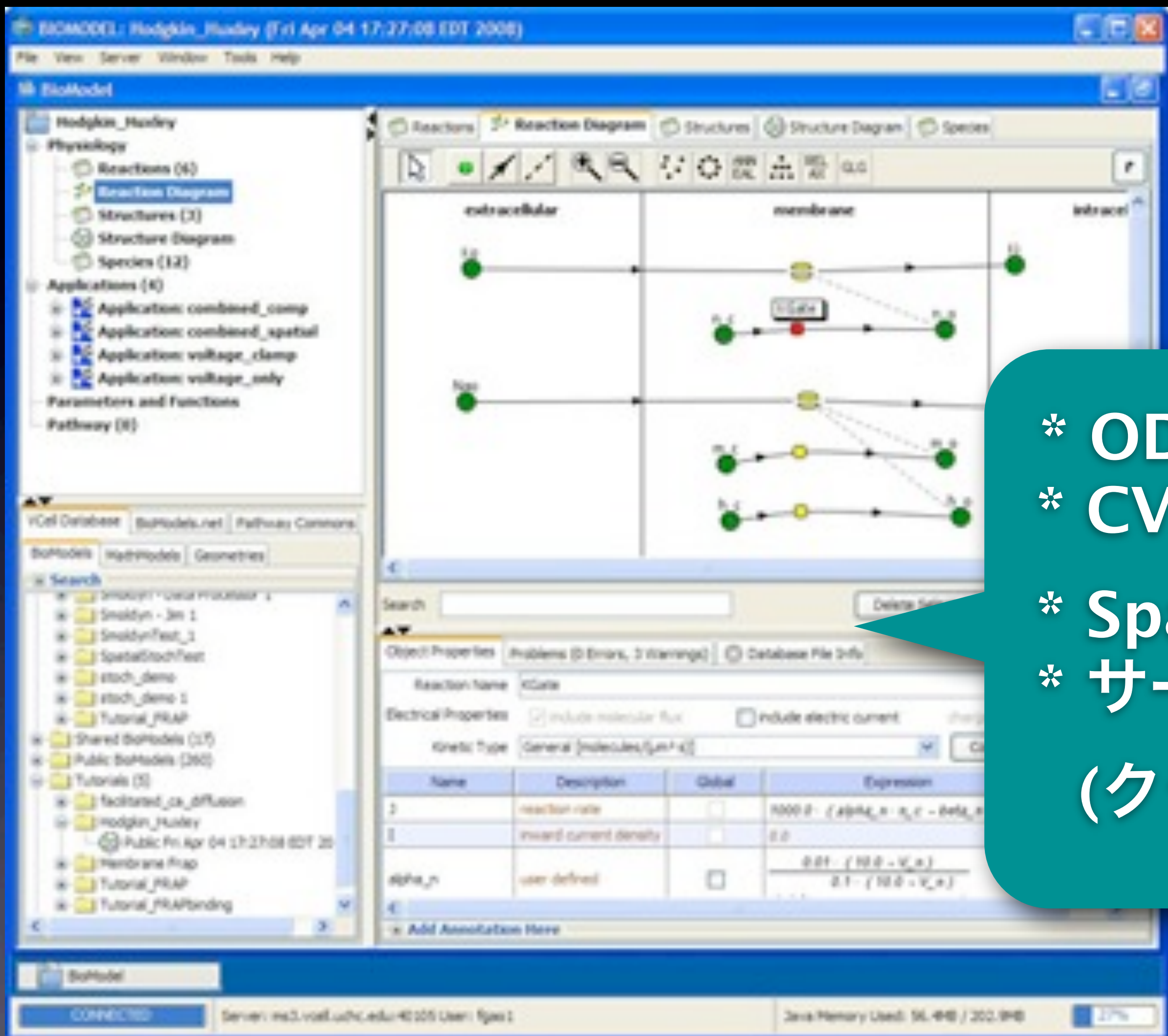


* ODE, PDE(FEM)
* モジュール構造
* C++, Java

Osaka University, OIST

http://physiodesigner.org/

VCell



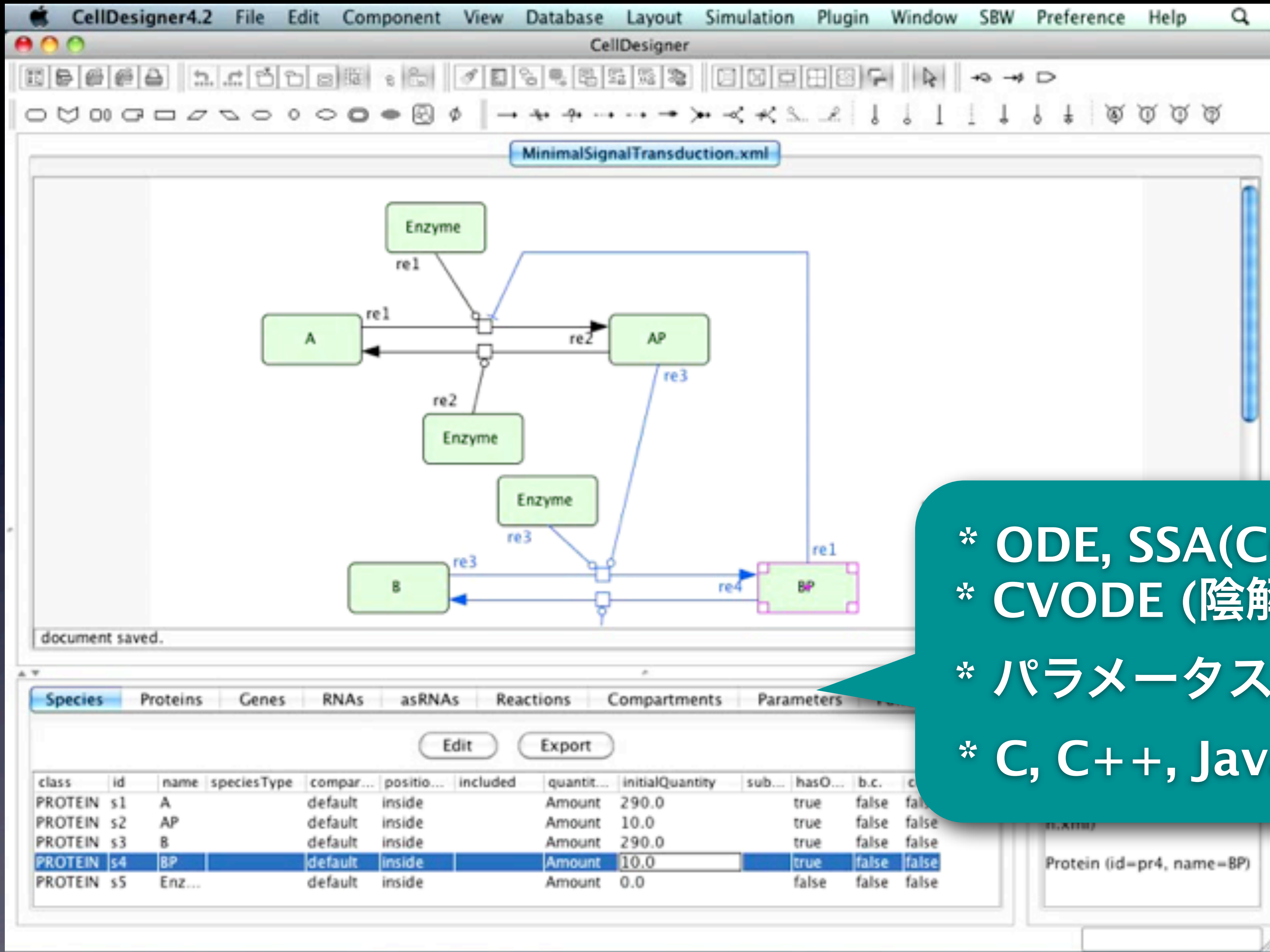
* ODE, SSA ,PDE, DAE
* CVODE/IDA (陰解法もOK)
* Spatial Stochastic (Smoldyn)
* サーバサイド

(クライアント版もリリース)

University of Connecticut Health Center

http://vcell.org/

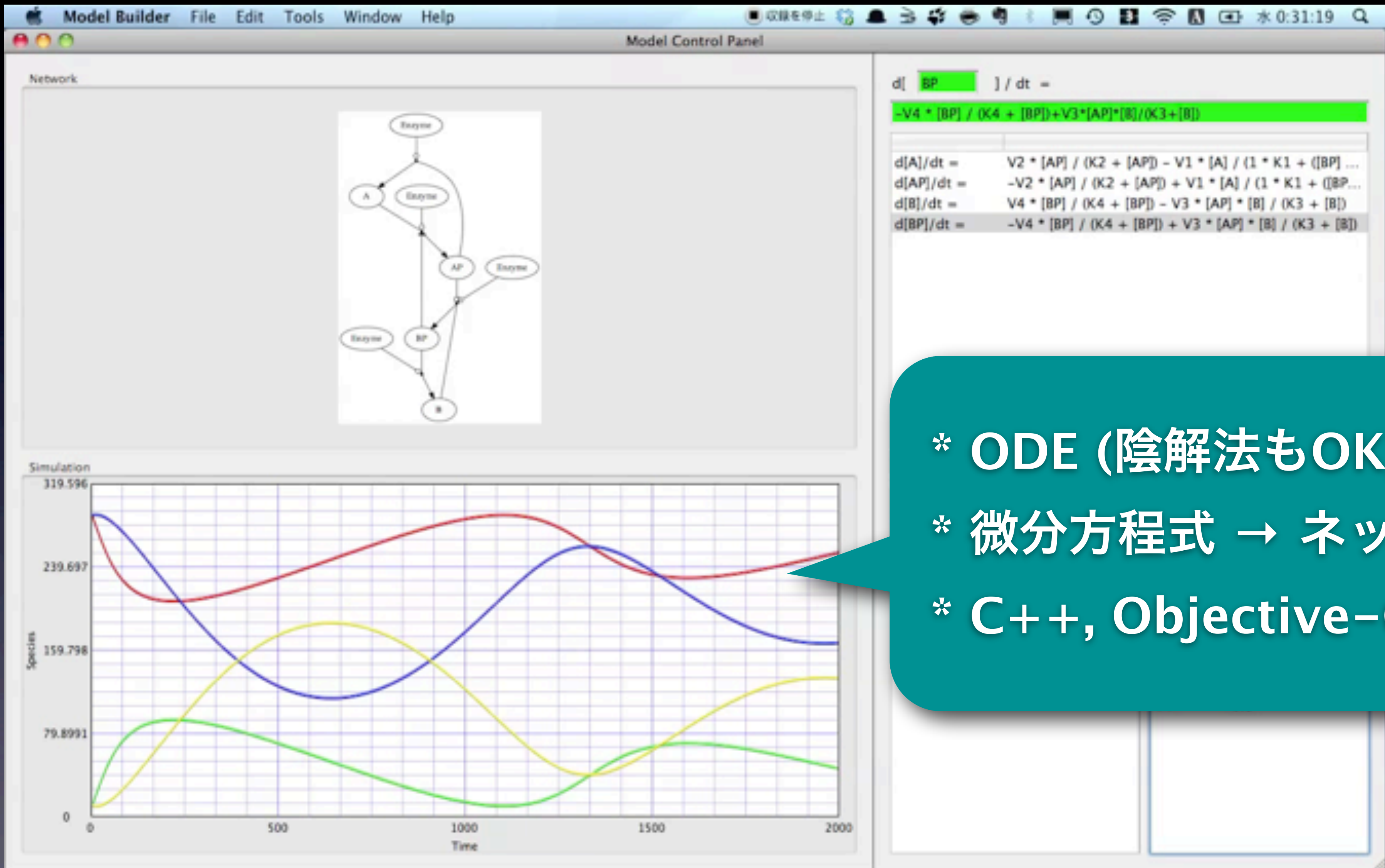
CellDesigner



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

http://celldesigner.org/ JST, SBI, Keio University

最速モデル構築

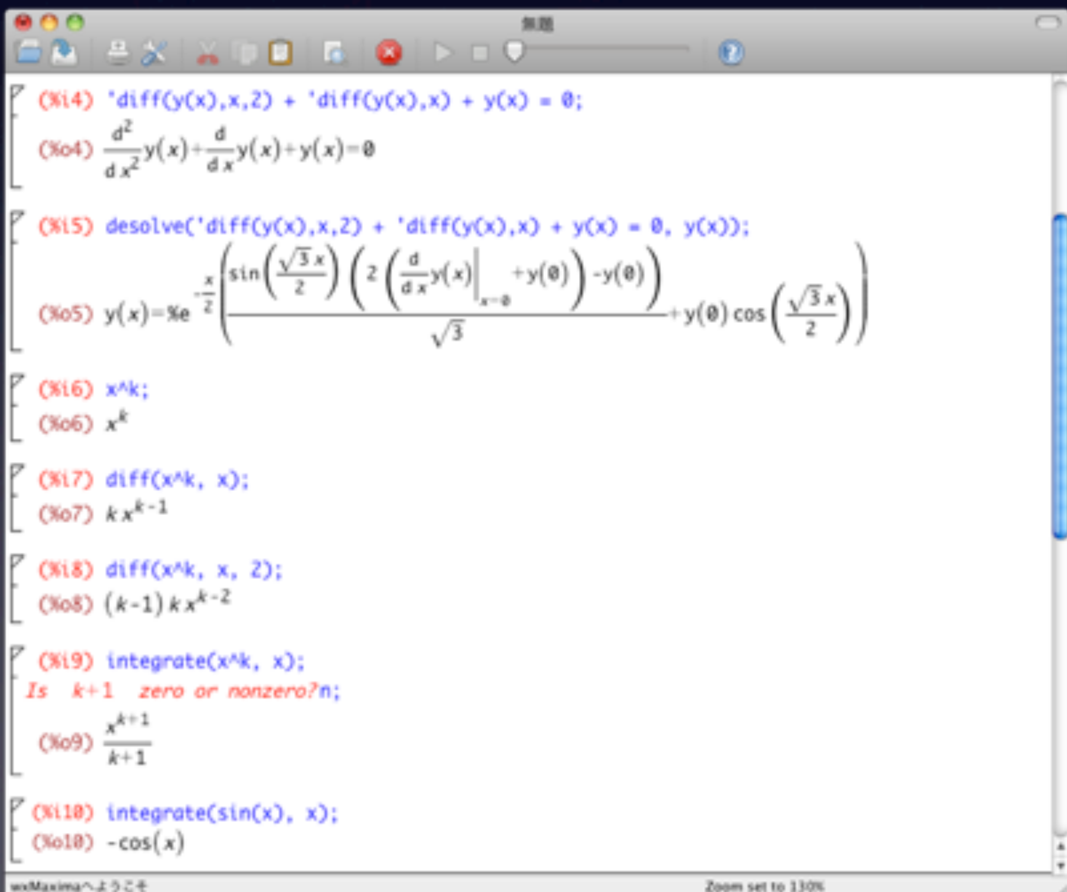


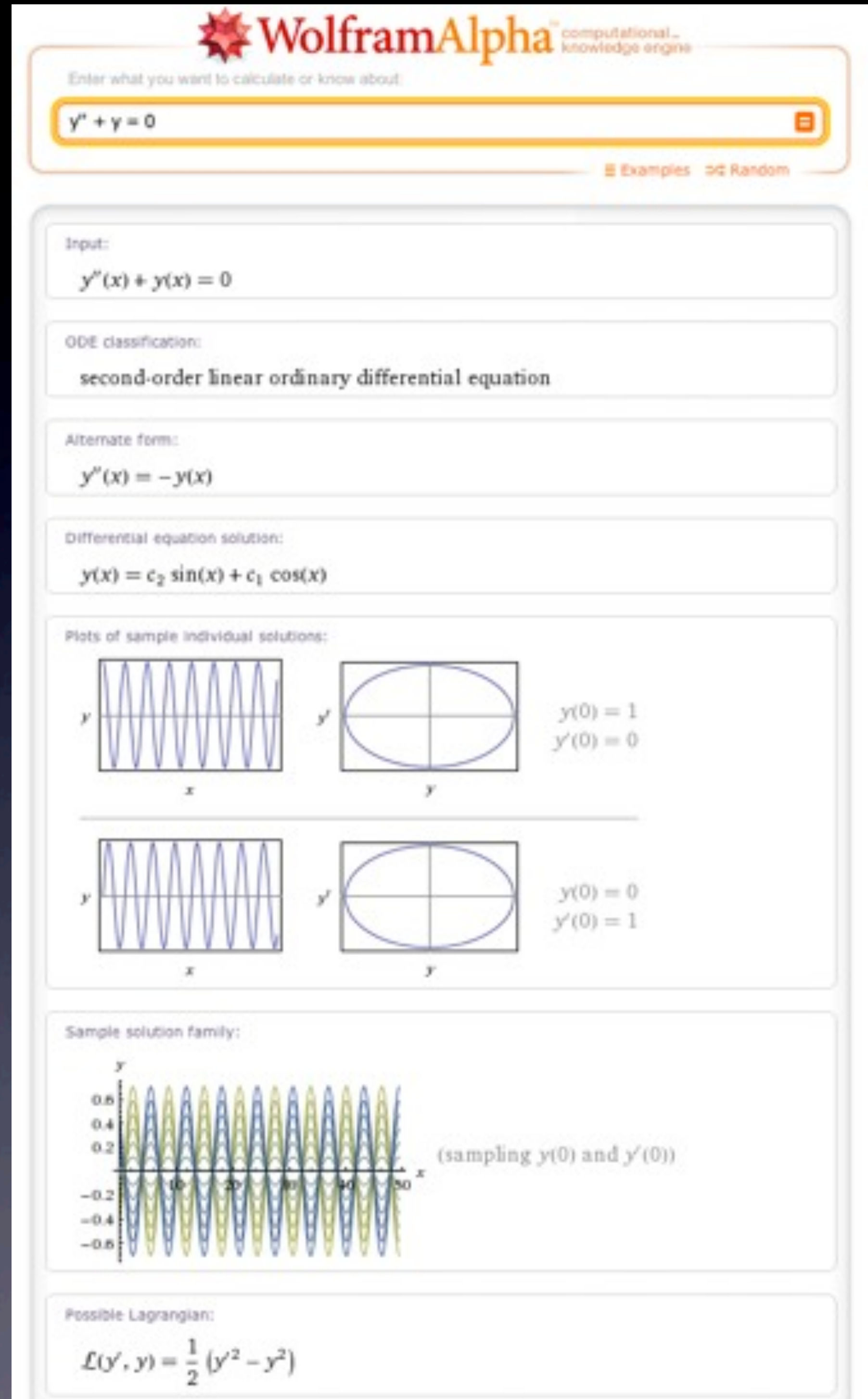
* ODE (陰解法もOK)
* 微分方程式 → ネットワーク
* C++, Objective-C

Keio University

数式処理

- Maxima, wxMaxima
- Wolfram Alpha





謝辞

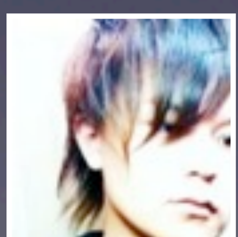
- 東京大学 生産技術研究所 小林徹也 准教授
- 慶應義塾大学 理工学部 広井賀子 専任講師

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)



近原鷹一
可変ステップ幅