

Matlabの基本的操作

2019年 夏期実習

守田 啓悟

資料は、Webからダウンロードしてください。

<http://kurodalab.bi.s.u-tokyo.ac.jp/class/Summer/2019>

→直打ち

または

①Googleで「東京大学黒田研究室」検索

②黒田研のHPを開き、<http://kurodalab.bi.s.u-tokyo.ac.jp/> 以下にclass/Summer/2019 を追加

本日の目標

とにかく手を動かして、Matlabに慣れる

慣れている人は、周りを助けながら、先に進んでください

Matlabとは

- 米国MathWorks社の数値解析ソフトウェア（有償）
- 長所
 - ベクトル、行列計算が得意
 - グラフ表示機能なども充実
 - 対話形式とプログラム形式の両方に対応
- 対話形式（初学者向き）
 - 命令を一つずつ実行する。電卓と同様。
- プログラム形式
 - 複数の命令をまとめて実行

ドキュメンテーション

<https://jp.mathworks.com/help/matlab/>

非常に充実している(日本語もある)

- ・疑問が生じたとき
 - ・エラーがでたとき
 - ・教員が変なことを言っている気がしたとき
- に活用できる

The screenshot shows the MathWorks documentation website for MATLAB. The header includes the MathWorks logo and navigation links for products, solutions, academia, support, community, and events. A button for 'MATLAB を入手する' (Get MATLAB) and a contact icon are also present. The main navigation bar highlights 'ドキュメンテーション' (Documentation) with sub-links for 'すべて' (All), '例' (Examples), and '関数' (Functions). A search bar on the right shows 'R2019a のドキュメンテーションを' (Search R2019a documentation for) and a dropdown for 'ドキュメンテーション' (Documentation). The left sidebar lists various MATLAB topics, with 'MATLAB' selected. The main content area features a message about the latest release, a 'MATLAB' section header, and a description of the software. A right sidebar contains links for 'リリースノート' (Release Notes) and 'PDF 版ドキュメンテーション' (PDF Documentation). The bottom of the page shows a list of related products like Simulink and various toolboxes.

MathWorks® 製品 ソリューション アカデミア サポート コミュニティ イベント [MATLAB を入手する](#)

ドキュメンテーション [すべて](#) [例](#) [関数](#) [R2019a のドキュメンテーションを](#) [ドキュメンテーション](#)

三 目次 [閉じる](#) [評価版](#) [製品アップデート](#)

MATLAB

- MATLAB 入門
- 言語の基礎
- 数学
- グラフィックス
- データのインポートと解析
- スクリプトと関数のプログラミング
- アプリの作成
- 高度なソフトウェア開発
- デスクトップ環境
- サポートされているハードウェア

Simulink
5G Toolbox
Aerospace Blockset
Aerospace Toolbox
Antenna Toolbox

最新のリリースでは、このページがまだ翻訳されていません。このページの最新版は英語でご覧になれます。

MATLAB

技術計算言語

世界中の何百万人もの技術者や科学者が MATLAB® を使用して、世界を変えるシステムや製品を解析および設計しています。行列ベースの MATLAB 言語は、世界で最も自然に計算数学を表現できます。組み込みのグラフィックスによって容易にデータを可視化して考察できます。デスクトップ環境での実験、調査および検出が可能です。これらの MATLAB のツールや機能は、すべて厳密にテストされており、連携して機能するように設計されています。

MATLAB では、デスクトップの枠を越えてアイデアを実現します。大規模なデータセットに対して解析を実行し、クラスターやクラウドに拡張することができます。MATLAB コードを他の言語と統合して、アルゴリズムやアプリケーションを Web システム、エンタープライズ システムおよび運用システムに展開することができます。

MATLAB 入門

MATLAB の基礎を学ぶ

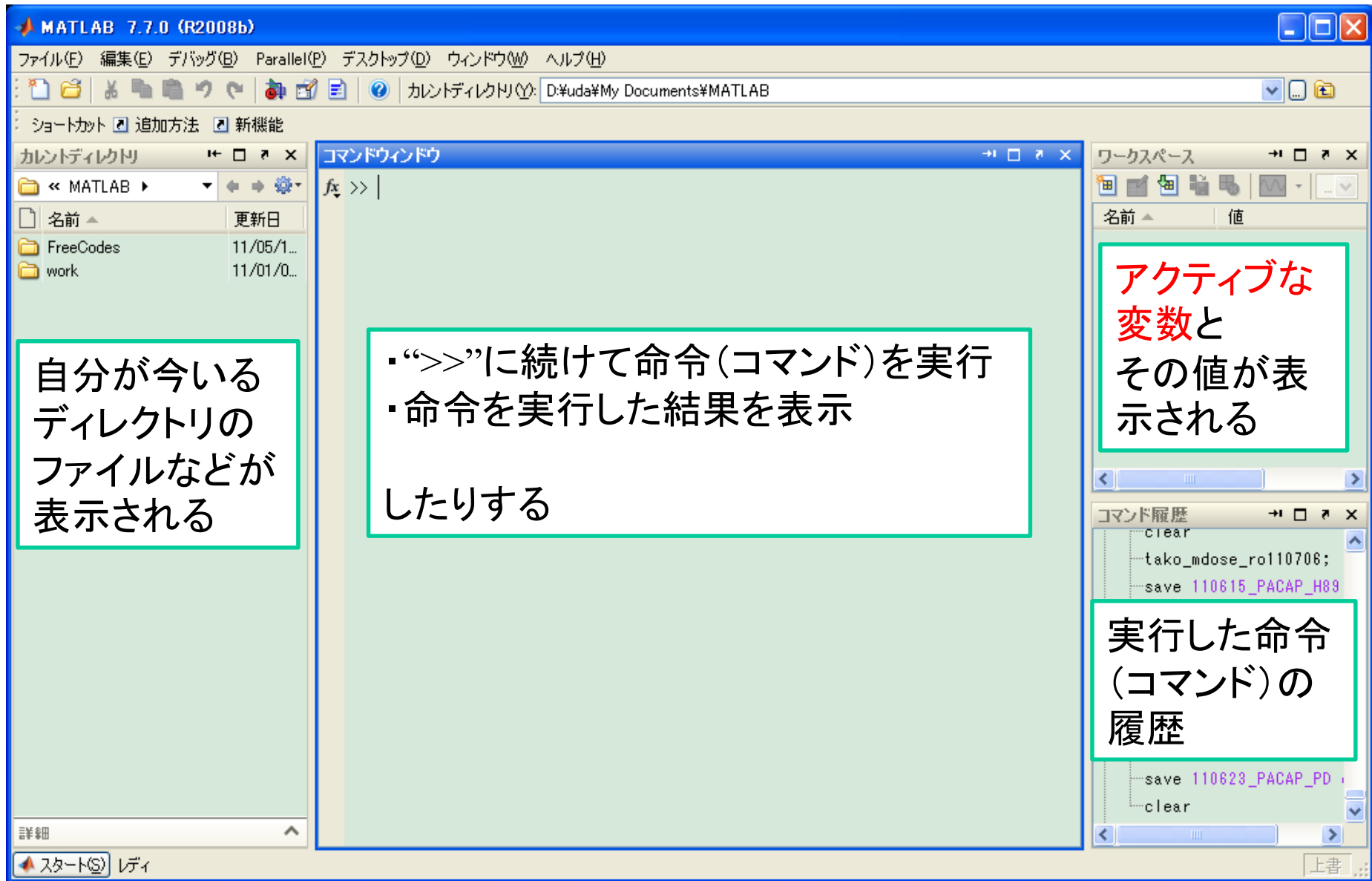
言語の基礎

構文、変数、型、制御フロー、関数、データ型、オブジェクト指向プログラミング

[リリースノート](#)
[PDF 版ドキュメンテーション](#)

R2018b

画面の説明



「変数」とは？

- 数学における定数・変数と同じと考えてよい
 - 数値を一時格納する「箱」

$$a^2 + b^2 = c^2 \quad \begin{cases} a = 3 \\ b = 4 \\ c = 5 \end{cases}$$

- Matlabの変数は、数値だけでなく文字も格納する
 - ほとんどのプログラム言語も同様

```
n = 'Pythagoras'
c = 'Greece'
```

変数に値を代入する

「=」で左辺の変数に値を代入できる
(c.f. 等号は「==」で表す)

```
>> a = [1,2,3]
```

a = 行ベクトル

1 2 3

```
>> b = [4;5;6]
```

b =

4
5 列ベクトル
6

```
>> X= [ 1,2;3,4]
```

X =

1 2
3 4

行列

```
>> Y= [ 5,6;7,8]
```

Y =

5 6
7 8

※大文字と小文字は区別される

変数に値を代入する

「=」で左辺の変数に値を代入できる

(c.f. 等号は「==」で表す)

注目

列の作り方

```
>> a = [1, 2, 3]
```

a = 行ベクトル

1 2 3

```
>> b = [4; 5; 6]
```

b =

4

5

6

列ベクトル

```
>> X = [1, 2; 3, 4]
```

X =

1 2

3 4

行列

```
>> Y = [5, 6; 7, 8]
```

Y =

5 6

7 8

※大文字と小文字は区別される

$$a = [1, 2, 3]$$

$$b = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$Y = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

転置

「`'`」で転置される
つまり、タテとヨコが入れかわる

```
>> a'
```

```
ans =
```

```
1
2
3
```

```
>> b'
```

```
ans =
```

```
4
```

※代入操作以外の
実行結果は
「ans」という変数に
自動的に格納される

```
5
```

```
6
```

```
>> X'
```

```
ans =
```

```
1    3
2    4
```

```
>> Y'
```

```
ans =
```

```
5    7
6    8
```

はじめは何に使うんだって感じだけど結構使う

$$a = [1, 2, 3]$$

$$b = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$Y = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

変数同士の演算

```
>> X+Y
```

```
ans =
```

```
      6      8
     10     12
```

```
>> X-Y
```

```
ans =
```

```
     -4     -4
     -4     -4
```

足し算

引き算

```
>> X*Y
```

```
ans =
```

```
      19      22
      43      50
```

```
>> X.*Y
```

```
ans =
```

```
      5      12
      21      32
```

```
>> X./Y
```

```
ans =
```

```
      0.2000      0.3333
      0.4286      0.5000
```

行列の掛け算

「.」を付けると
成分ごとの
演算になる

値の参照

数列を作って任意の値を取り出してみよう
(準備)-2から2まで1刻みの数列を代入

```
>> x = 
```

```
x =
```

```
-2    -1     0     1     2
```

値の参照

数列を作って任意の値を取り出してみよう
(準備)-2から2まで1刻みの数列を代入

```
>> x = -2:1:2
```

```
x =
```

```
    -2    -1     0     1     2
```

値の参照

数列を作って任意の値を取り出してみよう
(準備)-2から2まで1刻みの数列を代入

```
>> x = -2:1:2
```

```
x =
```

```
    -2    -1     0     1     2
```

```
>> x(2)    2番目の値を参照
```

```
ans =
```

```
    -1
```

値の参照

数列を作って任意の値を取り出してみよう
(準備)-2から2まで1刻みの数列を代入

```
>> x = -2:1:2
```

```
x =
```

```
    -2    -1     0     1     2
```

```
>> x(2)    2番目の値を参照
```

```
ans =
```

```
    -1
```

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
>> X(1,2)
```

(1,2)成分を
参照

```
ans =
```

```
    2
```

```
>> X(1,:)    第1行を  
              参照
```

```
ans =
```

```
    1
```

```
    2
```

```
>> X(:,2)    第2列を  
              参照
```

```
ans =
```

```
    2
```

```
    4
```

よく使う操作など

行列の設定

```
>> B=zeros(4,3)
```

```
B =  
  
    0    0    0  
    0    0    0  
    0    0    0  
    0    0    0
```

```
>> B(2,:)=a
```

```
B =  
  
    0    0    0  
    1    2    3  
    0    0    0  
    0    0    0
```

第2行にベクトルaを代入

```
>> x>0
```

```
ans =
```

```
    0    0    0    1    1
```

```
>> x>=0
```

```
ans =
```

```
    0    0    1    1    1
```

```
>> x==1
```

```
ans =
```

```
    0    0    0    1    0
```

```
>> x(end)
```

```
ans =
```

```
    2
```

真→1, 偽→0
の論理値を返す

「end」は、
配列の最後
を表す

```
>> a
```

```
a =
```

```
    1    2    3
```

```
>> b
```

```
b =
```

ベクトルaで
値が2の所の
indexを
ベクトルb
で参照

```
>> b(a==2)
```

```
ans =
```

```
    5
```

配列サイズの取得

- `length(x)` ベクトル`x`の要素数を返す
(※2次元以上の配列の場合, 最大の要素数を返す)
- `size(A)` 配列`A`のサイズを返す

```
>>  
>> x=[1,2,3,4,7]  
  
x =  
  
     1     2     3     4     7  
  
>> length(x)  
  
ans =  
  
     5
```

```
>> A = zeros(5,3,8);  
>> size(A)  
  
ans =  
  
     5     3     8
```

実行例

繰り返しループ

kを変えながら、
繰り返す範囲

```
>> aa = 0;
>> for(k = 1:10)
    aa = aa + k;
end
>> aa
```

aaの結果を
表示

aa =
55

※「;」を付けると結果の
出力を表示しなくなる
(メモリには保存される)

Annotations:

- aaに0を代入
- kを変える値.
k=1,2,...,10
- 繰り返しの間,
実行される命令

Advanced) kの間隔を変えたいときどうするか？

1, 3, 6, 9, 12でfor文を回したいなど

条件判定

```
>> x = 1
```

```
x =
```

```
1
```

```
>> y = 0
```

```
y =
```

```
0
```

```
>> if x > 0
```

```
y = 1
```

```
end
```

```
y =
```

```
1
```

x=1のとき

```
>> x = -2
```

```
x =
```

```
-2
```

```
>> y = 0
```

```
y =
```

```
0
```

```
>> if x > 0
```

```
y = 1
```

```
end
```

```
>>
```

x=-2のとき

条件式
条件式が
成り立つ(真)
のときだけ、
実行される命令

結果

Advanced)
≤とかはどう指定する？

図の表示(1)

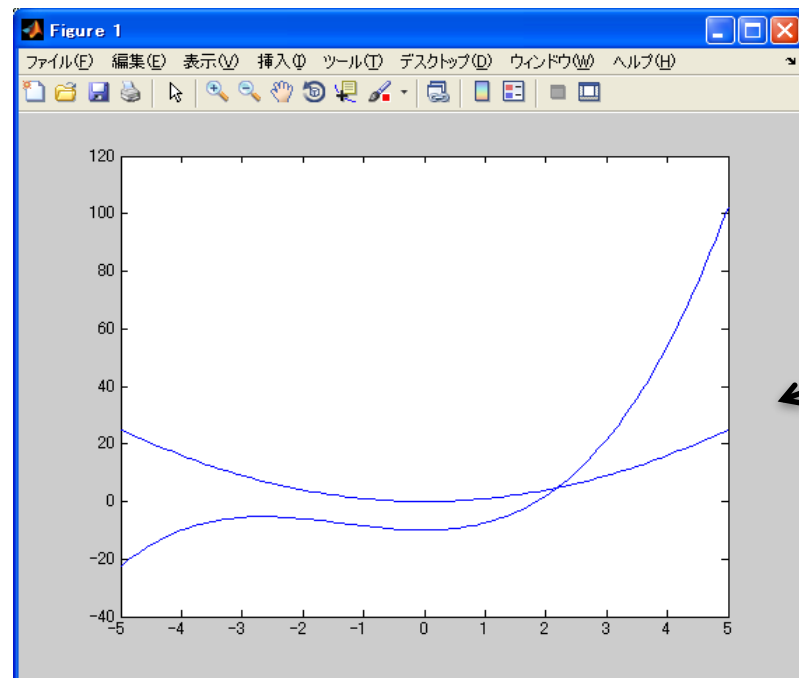
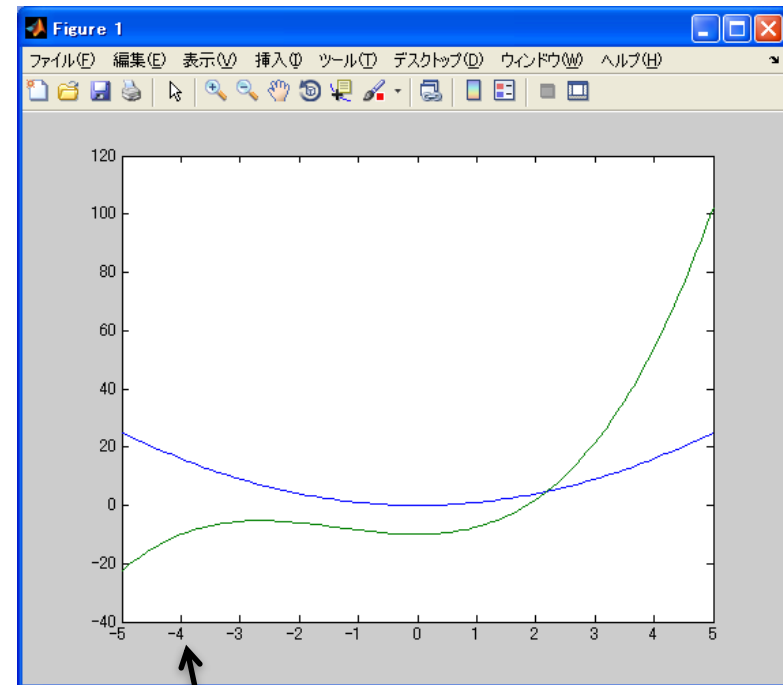
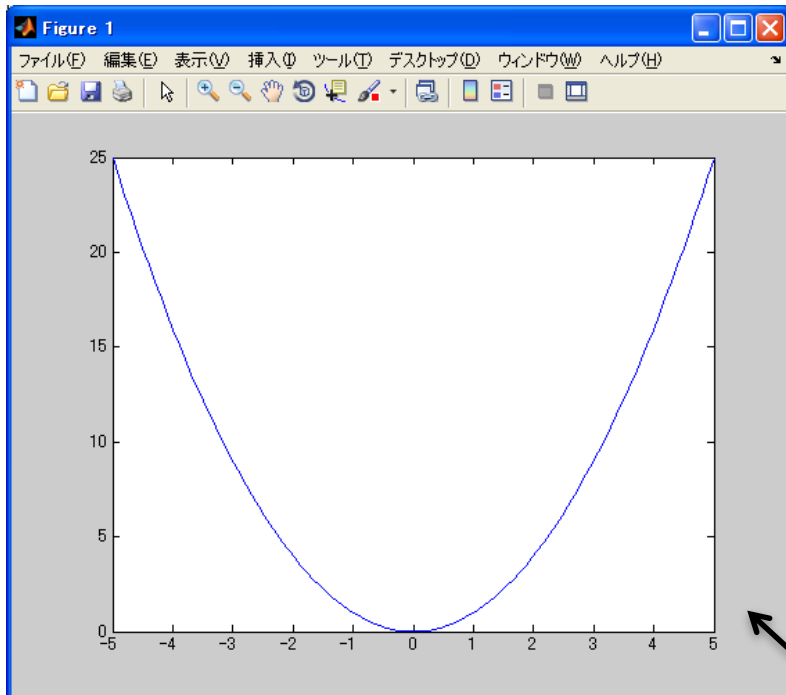
```
1 %%  
2 - x = linspace(-5,5,100);  
3 - y1 = x.^2;  
4 - plot(x,y1)  
5  
6 %%  
7 - y2 = 0.5*x.^3 + 2*x.^2 -10;  
8 - plot(x,y1,x,y2)  
9  
10 %%  
11 - plot(x,y1)  
12 - hold on  
13 - plot(x,y2)  
14
```

→ 区間 $[-5,5]$ で, 等間隔な100点を作る

→ 横軸 x , 縦軸 y_1 の図を描く

→ 2本の線を描く

→ 重ねて描く



実行結果

図の表示(2)

```
%%
```

```
figure(1)
```

```
plot(x,y1,'r','linewidth',8)
```

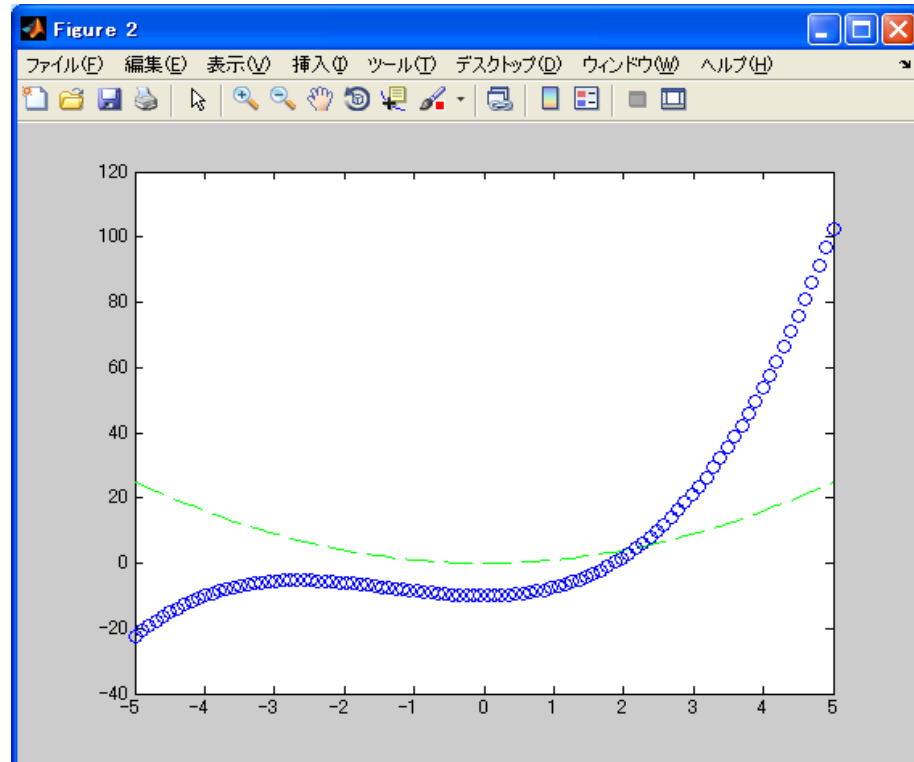
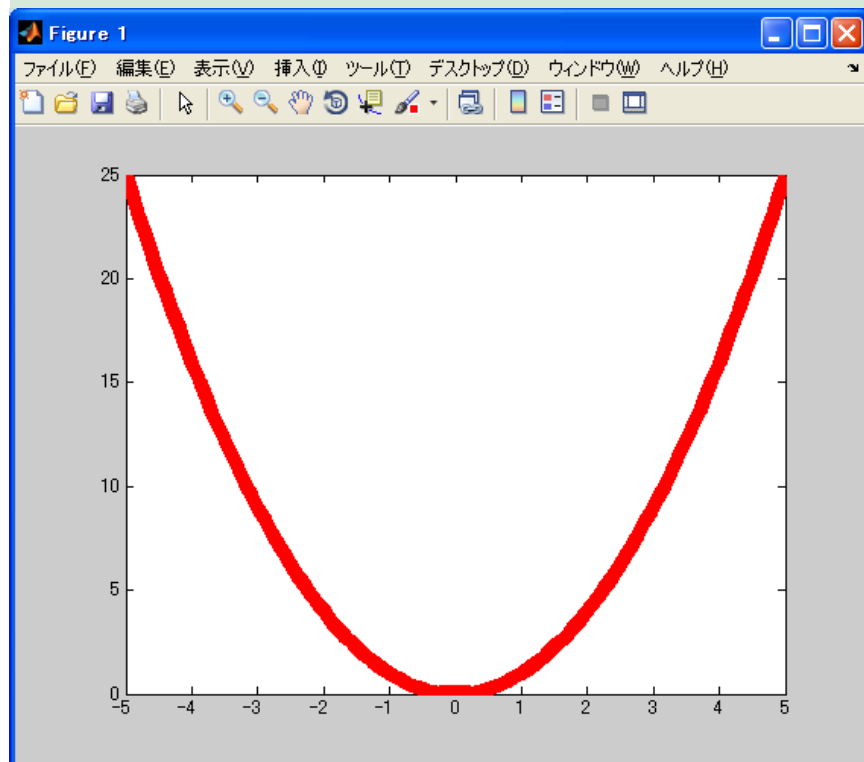
```
figure(2)
```

```
plot(x,y1,'g--',x,y2,'bo')
```

図の番号を指定

線の種類や色, 太さを
指定できる

※コマンドラインから,
“>help plot”で説明が出る



実際の作業では

毎回いちいちコマンドラインから打ち込むのは面倒



- 「*.m」という拡張子のファイルを作る
- コマンドをファイルに書き込む
- 複数のコマンドを、一つの関数にまとめる

※ editコマンドや, GUIから”ファイル”→”新規作成”→
“ファンクション Mファイル”で作れる

「関数」とは？

- 数学における関数(函数)と同じと考えてよい
 - 数値を入力すると、数値が出力されるブラックボックス

`exp(x)` ($x = 1$ を入力 $\rightarrow$ 2.718... を出力)

- Matlabの関数は、数値だけでなく文字も入出力する
 - ほとんどのプログラム言語も同様

`upper(x)` ($x = \text{'Euclid'}$ を入力 $\rightarrow$ `'EUCLID'` を出力)

関数: 2変数の足し算

```
function [c] = fun_add(a,b)
    c = a + b;
end
```

返回值 関数名 引き数

処理を書く

- ・ `function` で始まり, `end` で終わる
- ・ 先頭の関数名とファイル名は, 同じにする

```
>> fun_add(1,3)
```

```
ans =
```

```
4
```

```
>> y = fun_add(2,3)
```

```
y =
```

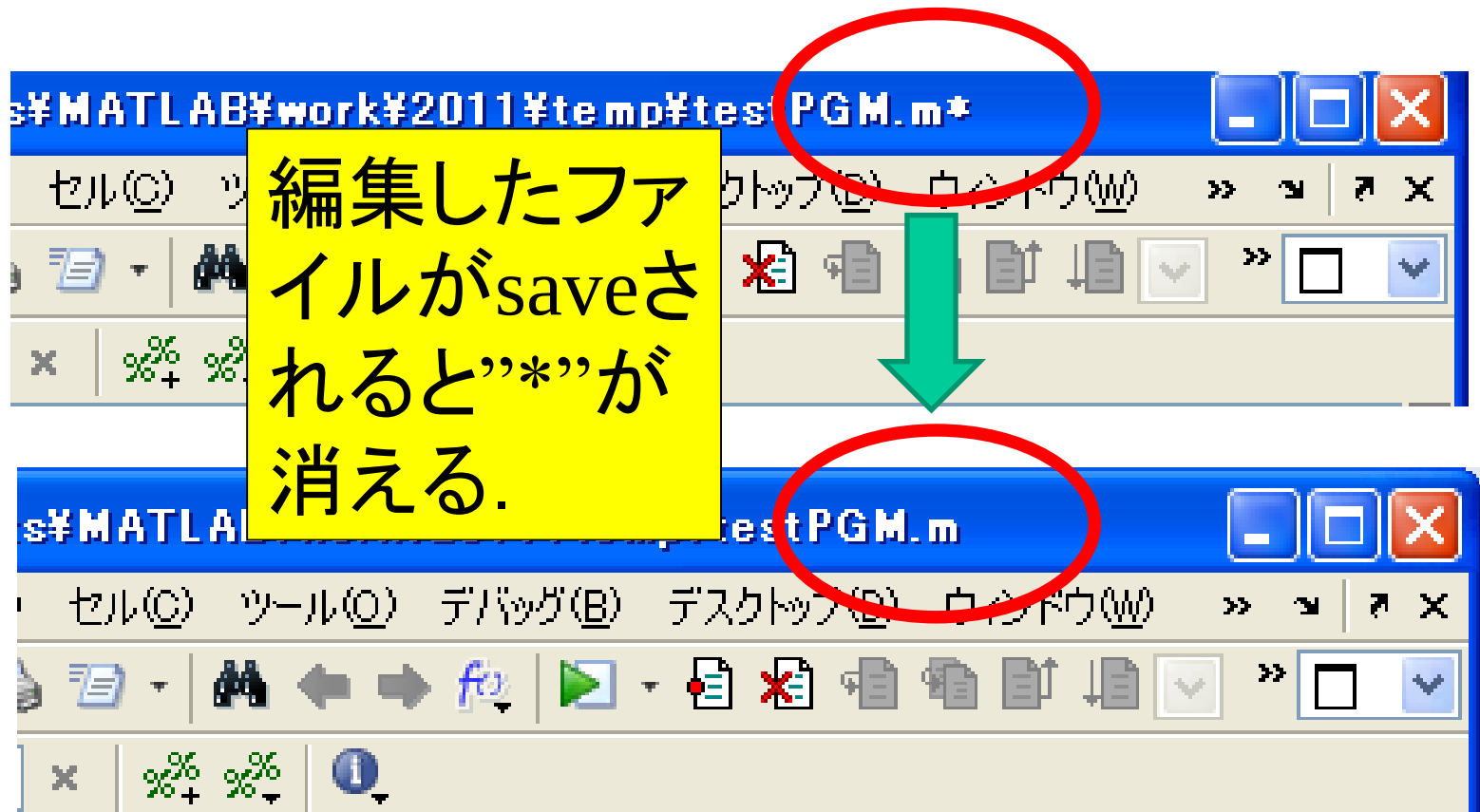
```
5
```

実行結果例

①エディタを開く. → ②関数を書く
→ ③fun_addで”save”してコマンドラインから実行, 確認する

プログラムを更新したら，saveしましょう

- 特に，実行前には，確認しましょう



関数内の変数

ひとつのファイル
内に、複数の関
数を記述可能

```
function [b] = testPGM(a)
```

```
    b = fun_F(a);
```

```
end
```

メイン関数:
先頭に書かれ、
最初に実行される

```
function [y] = fun_F(x)
```

```
    a = 10;
```

```
    y = x * a;
```

```
end
```

サブ関数

関数が違えば、同じ変数名でも異なる変数になる
⇒通常は、関数から他の関数内の変数を見ることはできない

関数内から関数を呼び出す
ことも可能

コマンドラインで関数を書きたいときは？

- 無名関数という機能を使う
- コマンドラインで
`@([変数1, 変数2, ...]) 式`
と書く。これが無名関数。
- 例: $y=x^2$ は `@( x ) x^2` と、書く

使用例

```
>> @( x ) x^2
```

```
ans =
```

```
@(x)x^2
```

```
>> ans( 2 )
```

```
ans =
```

```
4
```

無名関数を変数に格納する

使用例

- 無名関数を代入式の右辺におく
- 以後、左辺の変数名で関数を呼び出せる
 - 右の例では `sqr`

```
>> sqr = @( x ) x^2
```

```
sqr =
```

```
@(x)x^2
```

```
>> sqr( 2 )
```

```
ans =
```

```
4
```

ODEを解く(1)

常微分方程式 (Ordinary Differential Equation; ODE) を解く.

例えば, $\frac{dx}{dt} = -\frac{1}{\tau} x$ は,

手で解けて, 解析解 $x(t) = x_0 e^{-\frac{t}{\tau}}$ を得る

しかし, 手で解けるものは限られている
実用上は手で解けない場合が多い
(手で解けても, 特殊関数が現れたりする)



数値解法の高需要

ODEの数値解法の背後には テイラー展開がある

- テイラー展開

$$y(x + \Delta x) = y(x) + \Delta x \cdot y'(x) + \frac{(\Delta x)^2}{2!} y''(x) + \dots$$

- 高次の微分を求めるのは難しい

- 同等の近似精度の公式 = 数値解法

- 数値解法の次数

- n 次の項までテイラー展開と一致 = n 次の解法

オイラー法(Euler's method)

- 1次微分の項までテイラー展開と一致

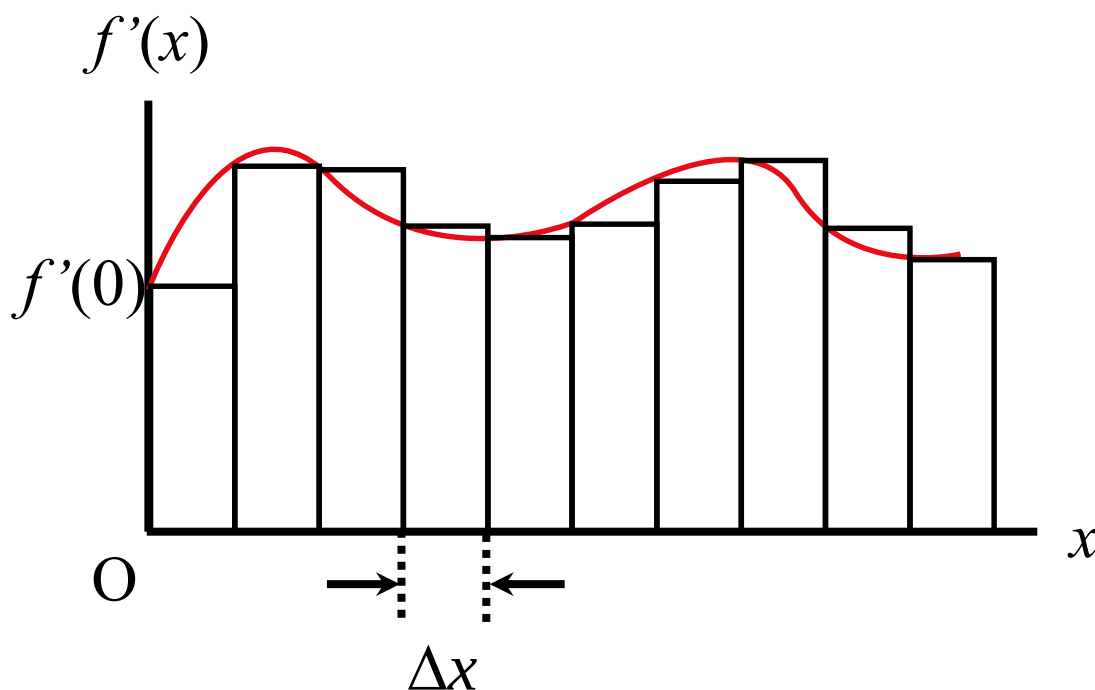
テイラー $y(x + \Delta x) = y(x) + \Delta x \cdot y'(x) + \frac{(\Delta x)^2}{2!} y''(x) + \dots$

オイラー $y(x + \Delta x) = y(x) + \Delta x \cdot y'(x)$

- 1次の数値解法
 - 最も簡単な数値解法である

ODEの数値解法を幾何学的に言うと

- 曲線下面積を近似的に求めること
- オイラー法では面積を小さな短冊に分割



実習で使うMATLABのODEソルバー

- ode45 (Dormand-Prince 5(4))
 - 4次と5次の二つの解法で計算
 - 二つの解法の結果が大差なければ、短冊の幅を広くする
- ode15s (後退差分公式)
 - 速い化学反応と遅い化学反応が混在する系に用いる

ODEを解く(2)

簡単なODEを数值的に解いて解析解と比べてみよう

微分方程式は、関数で定義される

ODE solver も関数なので、手続き上、関数に関数を渡す必要が生じる。関数に関数を渡すには？

➡ 関数ハンドルを使う。"@”を関数の前につける。

```
1 function [] = test_ode()
2
3     tau = 1; % パラメータ
4     tspan0 = [0,5]; % 解く時刻の範囲
5     x0 = 10; % xの初期値
6
7     [t,x] = ode15s(@(t,x) -1/tau*x,tspan0,x0);
8
9     plot(t,x)
10    hold on
11
12 end
```

$$\frac{dx}{dt} = -\frac{1}{\tau} x$$

➡ ODEを解く

ODE Solver 関数 (ODEの式, tの範囲, 初期値)

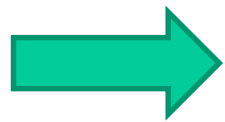
無名関数のアタマの"@”は関数ハンドル

test_odeで”save”してコマンドラインから実行, 確認する

関数ハンドル

微分方程式は、関数で定義される。

ODE solver も関数なので、手続き上、関数に関数を渡す必要が生じる。関数に関数を渡すには？



関数ハンドルを使う。"@ "を関数の前につける。

実行例

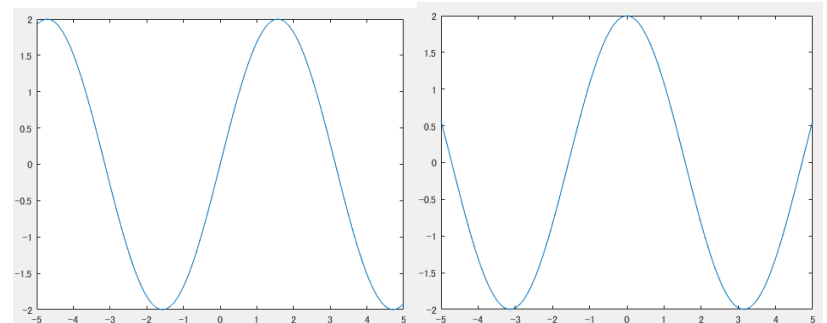
```
function [x,y] = test_FunHandle(fun1)

    x = linspace(-5,5,100);
    y = fun1(x) * 2;

end
```

プログラム例

```
>> [x,y]=test_FunHandle(@sin);
>> plot(x,y)
>> [x,y]=test_FunHandle(@cos);
>> plot(x,y)
```



無名関数を復習してみる

```
>> y = @(x) x.^2;  
>> y([1,3,5])  
  
ans =  
  
     1     9    25
```

関数ハンドルを使って、便宜的に関数名を省略して関数を作れる
これが無名関数だった！

```
[t,x] = ode15s(@(t,x) ode_eq(x,t,tau),tspan0,x0);
```

無名関数 $@(t,x)$ の中身は関数 $\text{ode_eq}(x,t,\text{tau})$

ODEを解く(3)

ODEの式を外部で関数として定義して解いてみよう.

```
1 function [] = test_ode()
2
3     tau = 1; % パラメータ
4     tspan0 = [0,5]; % 解く時刻tの範囲
5     x0 = 10; % xの初期値
6
7     [t,x] = ode15s(@ (t,x) ode_eq(x,t,tau), tspan0, x0);
8
9     plot(t,x)
10    hold on
11
12 end
```

ODE Solver関数 (ODEを定義した関数, tの範囲, 初期値)

```
14 function [dx] = ode_eq(x,t,tau)
15     % 解く微分方程式
16
17     dx = -1/tau * x;
18
19 end
```

ODEを定義

※ “@ (t,x) ode_eq(x,t,tau)” ⇒ ode_eqを用いて定義されるパラメタ, xの無名関数

test_odeで”save”してコマンドラインから実行, 確認する

発展:オイラー法のソルバーを作ってみよう

- ODEソルバーは中で何をやっているのか？
- ode.Euler.mを開き、記入して実行してみよう
- **f**: 解きたいODE
— 無名関数

```
function [ x_ans , y_ans ] = odeEuler( f , x_range, y0 )
    dx = 0.1;           % 短冊の幅
    x = x_range(1);     % xの初期値
    y = y0;             % yの初期値

    solution = [ x , y ]; % ODEの解を格納する変数

    while x <= x_range(end)
        dy = f( x ) *   ;           % オイラー法の公式
        x = x + dx;       % xに増分を加える
        y =   ;           % yに増分を加える
        solution = [ solution ; x , y ]; % 計算結果を格納
    end

    x_ans = solution(:,1); % solution の第1列はx
    y_ans = solution(:,2); % solution の第2列はy
end
```

実行例 (微分方程式 $y'=x^2$ を解く)

```
>> [x,y] = odeEuler(@(x) x^2 , [0:1], 0);
>> plot(x,y)
```

発展:オイラー法のソルバーを作ってみよう

- ODEソルバーは中で何をやっているのか？
- ode.Euler.mを開き、記入して実行してみよう
- **f**: 解きたいODE
— 無名関数

```
function [ x_ans , y_ans ] = odeEuler( f , x_range, y0 )
    dx = 0.1;           % 短冊の幅
    x = x_range(1);     % xの初期値
    y = y0;             % yの初期値

    solution = [ x , y ]; % ODEの解を格納する変数

    while x <= x_range(end)
        dy = f( x ) * dx; % オイラー法の公式
        x = x + dx;       % xに増分を加える
        y = y + dy;       % yに増分を加える
        solution = [ solution ; x , y ]; % 計算結果を格納
    end

    x_ans = solution(:,1); % solution の第1列はx
    y_ans = solution(:,2); % solution の第2列はy
end
```

実行例 (微分方程式 $y'=x^2$ を解く)

```
>> [x,y] = odeEuler(@(x) x^2 , [0:1], 0);
>> plot(x,y)
```

マニュアルなど

○コマンドラインから, ”>help ****”で, コマンド*****に関する説明が表示される. 例:>help ode15s

○コマンドラインから, ”>lookfor ****”で, マニュアル内の文字列****を検索できる. 例:>lookfor ode

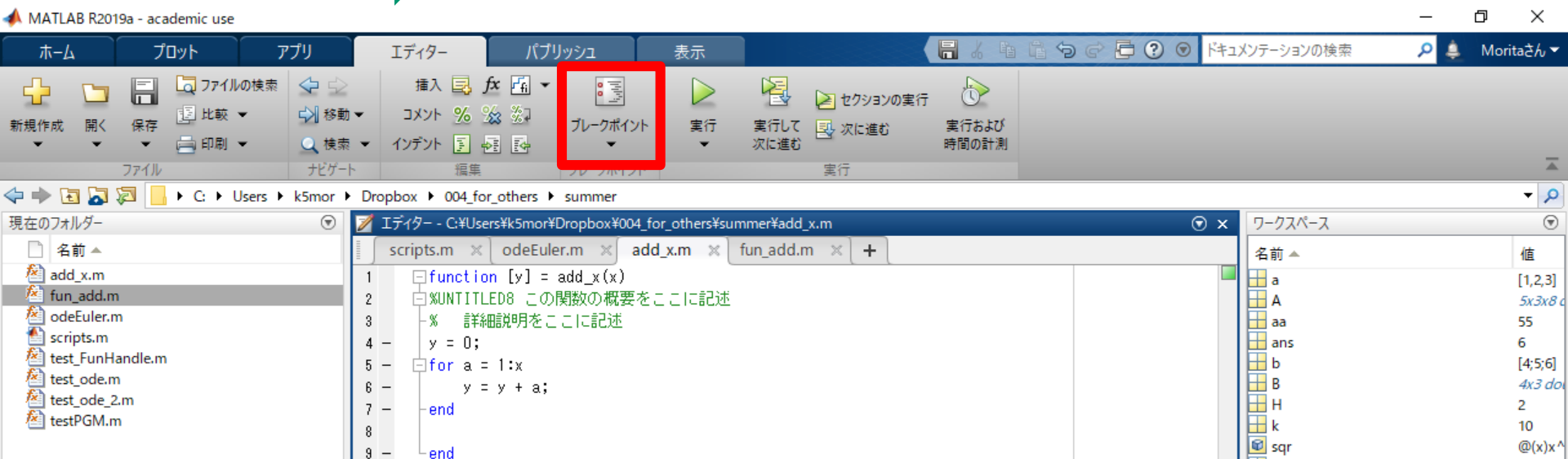
Webから, マニュアルを参照できる. ↓

<http://jp.mathworks.com/help/matlab/index.html>

デバッグ :ブレイクポイント 1/2

通常は、関数の内部(変数)を外から見ることはできない。

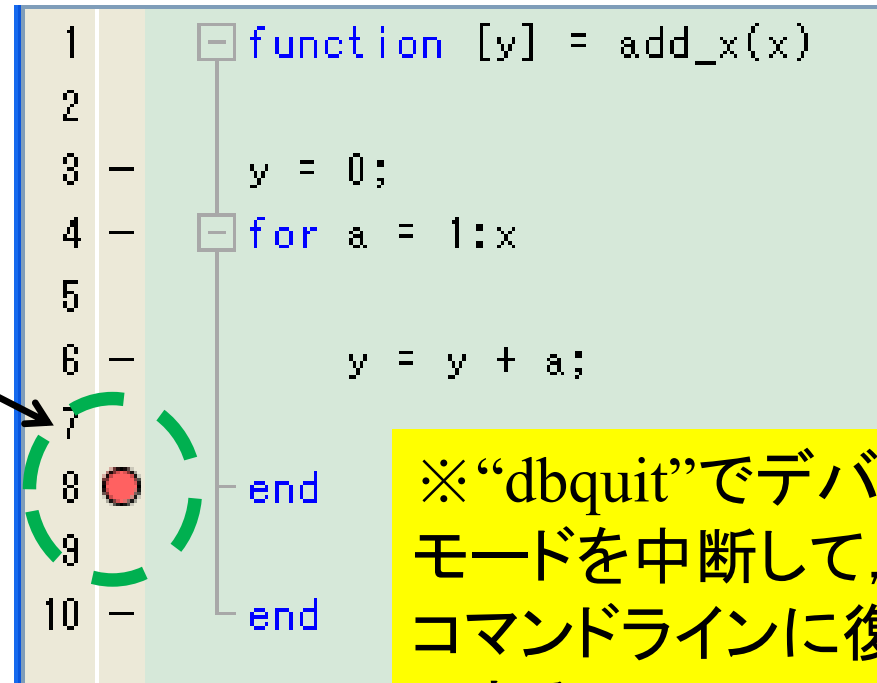
➡ バグの原因を解明するには不便。



デバッグ : ブ레이크ポイント 1/2

クリックでブレイクポイント(赤丸)を設置して実行

ブレイクポイントの直前で止まり、関数の中がのぞける。



```
1  function [y] = add_x(x)
2
3  -    y = 0;
4  -    for a = 1:x
5
6  -        y = y + a;
7
8  -    end
9
10 -    end
```

The image shows a MATLAB script editor with a function named `add_x`. The function has three lines of code: `y = 0;`, a `for` loop `for a = 1:x`, and `y = y + a;`. The `for` loop is followed by `end` and the function is closed with `end`. A red circle, representing a breakpoint, is placed on line 8, which is the `end` of the `for` loop. A green dashed circle highlights the breakpoint. A black arrow points from the text 'ブレイクポイントの直前で止まり、関数の中がのぞける。' to the breakpoint. The line numbers 1 through 10 are visible on the left side of the editor.

※“dbquit”でデバッグモードを中断して、コマンドラインに復帰できる。

F10でそこから1ステップずつ実行させることが可能。

デバッグ:ブレイクポイント 2/2

ワークスペース	
名前 ▲	値
a	7
x	10
y	28



“F10”でプログラムの動作を
1ステップずつチェック.

ワークスペース	
名前 ▲	値
a	8
x	10
y	36

ワークスペースの変化を逐一追える.

計算上の注意事項: NaN

NaNとは, 一般には”Not a Number”のことで, 演算結果が定まらないときに返される.

コマンドライン”>>”から, ”0/0”, ”1/0”などを計算してみよう.

※ Matlabでは, 無限大はInfとなる.

計算機言語によっては, NaNを検出するとwarningなどが出ることもありますが, Matlabでは基本的にはそのまま進んでしまうので気をつけましょう.

その他の頻出する関数

- max, min: 2つの数や配列に対して, 最大, 最小を返す.
- sum: 配列に対して和を計算する.
- while: 条件を満たすまで繰り返す.
- 作図関連で, xlabel, ylabel, title, legend...
など.

よく使う可能性が高いので, 時間に余裕があれば,
helpなどで一度調べて, 自分で試してみましょう.

休憩

実習の前に...

黒田研の実習では、ファイルをダウンロードしてもらうことが多くあります。

まずは、下記のホームページを開いてください。

<http://kurodalab.bi.s.u-tokyo.ac.jp/class/Summer/2019>

→直打ち

または

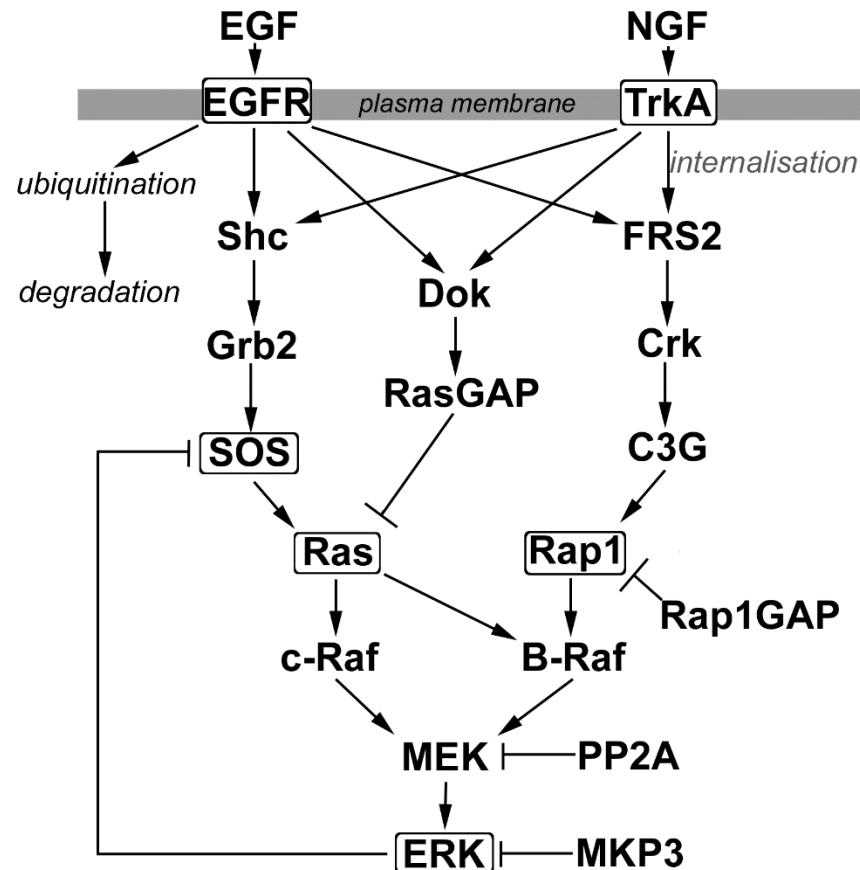
①Googleで「東京大学黒田研究室」検索

②黒田研のHPを開き、<http://kurodalab.bi.s.u-tokyo.ac.jp/> 以下にclass/Summer/2019を追加

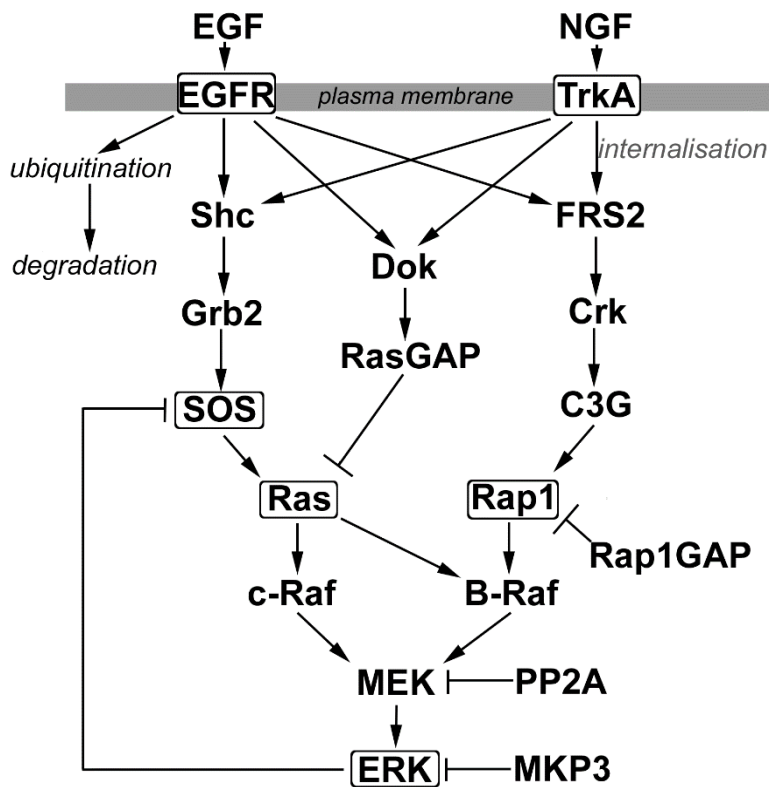
シグナル伝達(経路)とは？

細胞は外界などからの信号(シグナル)を受けて応答をする。
シグナルを受けてからから応答の様式を決定するまでの情報を伝える(シグナル伝達)経路を指す。シグナル分子のリン酸化やセカンドメッセンジャーによる生化学反応がその実行を担う。

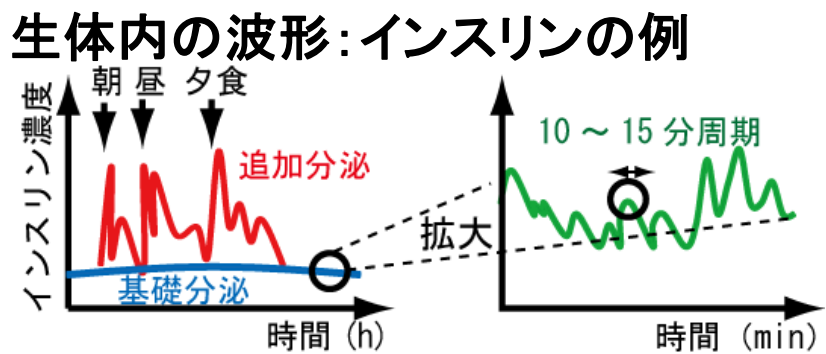
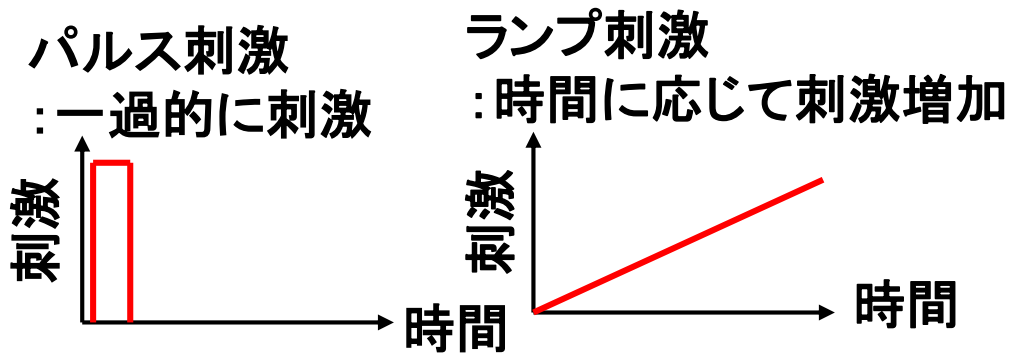
例) ERKシグナル伝達経路: 細胞の増殖・分化に関与



何故モデルを作成する？



○シミュレーション出来る！
PC上で様々な実験ができる！
どんな応答か予測できる！

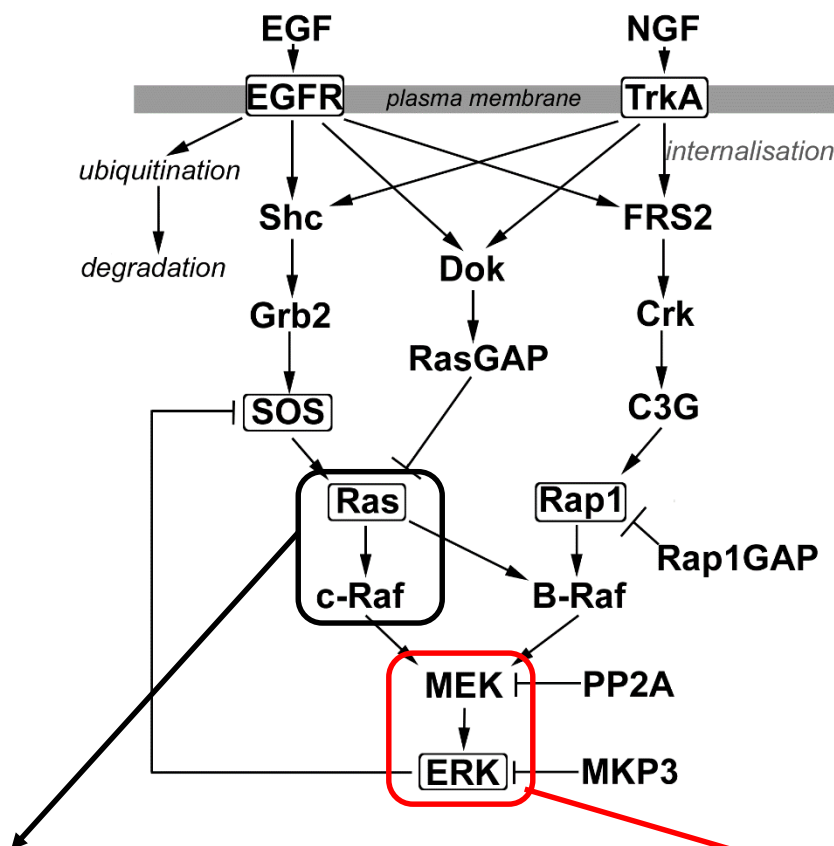


○理解が深まる
経路の特性を理解しやすい！

経路の特性を理解する方法として、説明したい現象を表現するモデルを作成し、解析する手法がある。
→生化学反応を微分方程式で表現した**生化学反応の微分方程式モデル**は、反応の動的特性を解析上で非常に強力である。本授業では微分方程式モデルの作成の仕方、そして解析例を基に、シグナル伝達経路解析の様式を学んでもらう。

○生化学反応シミュレーション **—基礎—**

・生化学反応は微分方程式で記述できる！



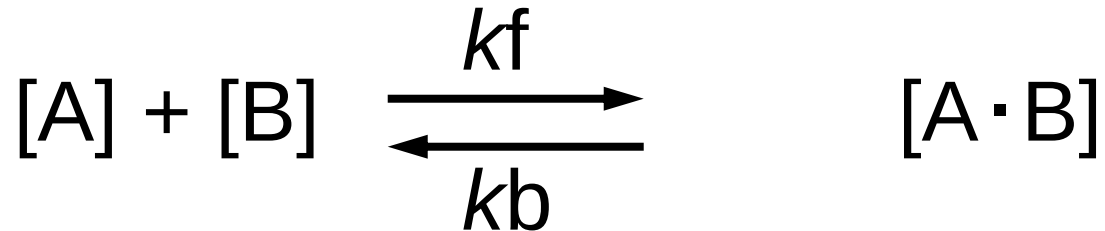
1. 分子間相互作用

2. 酵素反応

○生化学反応シミュレーション **—基礎—**

- ・生化学反応は微分方程式で記述できる！

反応の素過程①：分子間相互作用



$$Kd = \frac{kb}{kf} \quad Kd: \text{dissociation constant}$$

$$\frac{d[A]}{dt} = kb[A \cdot B] - kf[A][B]$$

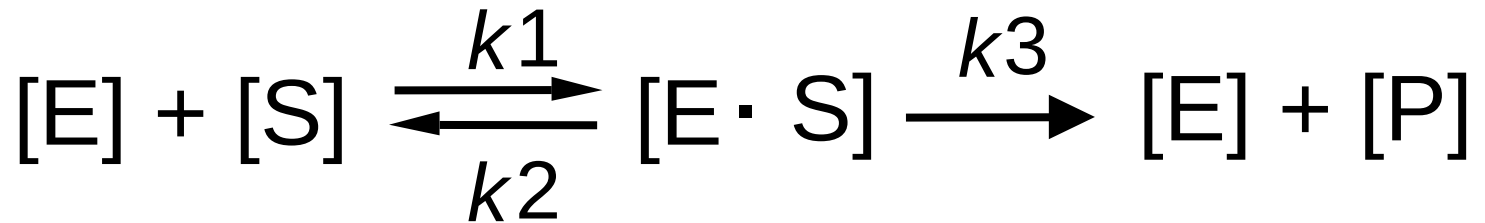
Ligand and receptor

G proteins and effectors

○生化学反応シミュレーション **—基礎—**

- ・生化学反応は微分方程式で記述できる！

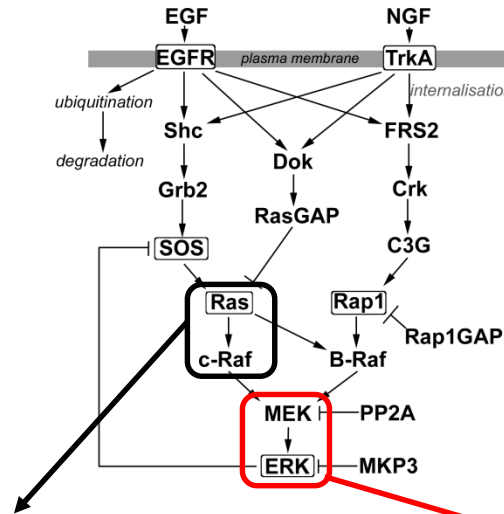
反応の素過程②：酵素反応



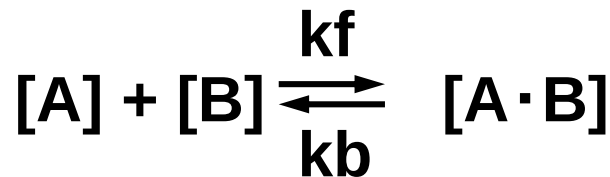
$$\frac{d[E \cdot S]}{dt} = k_1 \times [S] \times [E] - k_2 \times [ES] - k_3 \times [ES]$$

○生化学反応シミュレーション **—基礎—**

・生化学反応を微分方程式で記述する

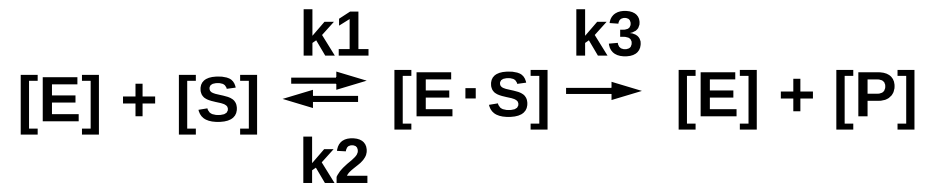


1. 分子間相互作用



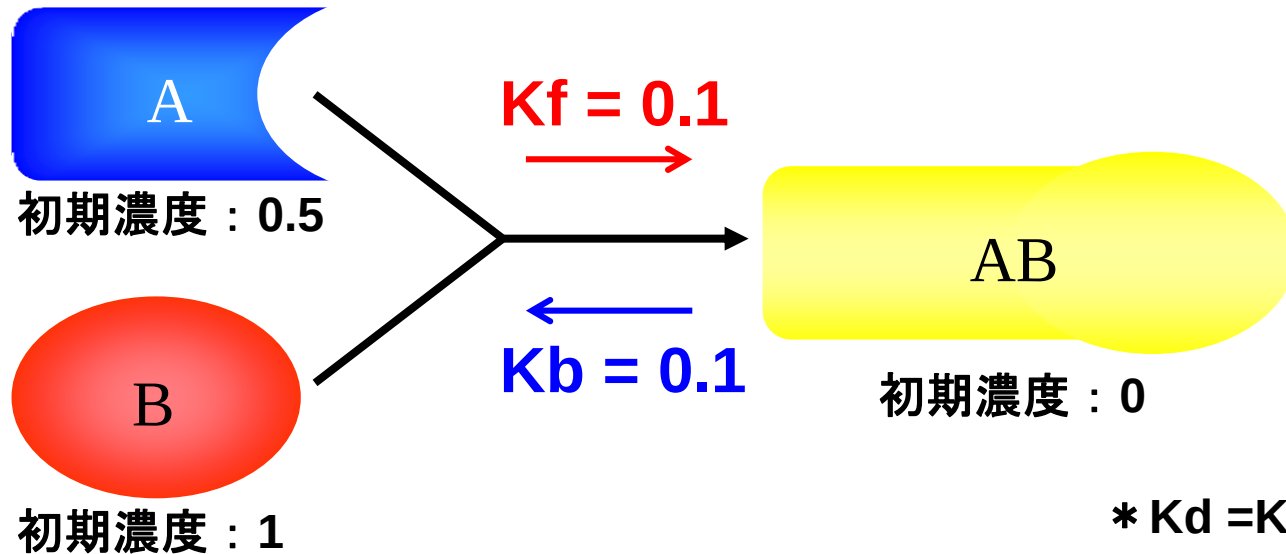
$$\frac{d[AB]}{dt} = kf \times [A] \times [B] - kb \times [AB]$$

2. 酵素反応



$$\frac{d[ES]}{dt} = k1 \times [S] \times [E] - k2 \times [ES] - k3 \times [ES]$$

モデルの作成 1 ー分子間相互作用ー



○微分方程式を作成し、その時間変動をプロットする

各分子濃度の時間変化を微分方程式で表現する

各分子濃度 : [A], [B], [AB] パラメータ : K_f , K_b

$$\frac{d[A]}{dt} = -K_f \times [A] \times [B] + K_b \times [AB]$$

$$\frac{d[B]}{dt} = -K_f \times [A] \times [B] + K_b \times [AB]$$

$$\frac{d[AB]}{dt} = K_f \times [A] \times [B] - K_b \times [AB]$$

モデルの作成 1 ー分子間相互作用ー

① モデルの作成: 下記を写してシミュレーションを行おう!

ひな型: <http://kurodalab.bi.s.u-tokyo.ac.jp/class/Summer/2019/Day1/MI.m>

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

初期値の設定
ODEを解く
Figureの作成

```
function MI() %関数名→保存するときのファイル名と同じに

time = 0:1:100; % 計算時間 (1秒毎に100秒まで)
param = [0.1,0.1]; % Kf,Kbのパラメータ
y0 = [0.5,1,0]; % A,B,ABの初期濃度

[t,time_course] = ode15s(@(t,y) ODE(t,y,param),time,y0); % 下にあるサブ関数に
                                                         % 移動してODEを解く

figure; % figure windowの形成
plot(t,time_course); % 上記[t,time_course]を作図
Molecular_Name = {'A','B','AB'}; % 測定分子
xlabel('time'); % x軸のラベル名
ylabel('concentration'); % y軸のラベル名
title('MI'); % タイトルのラベル名
legend(Molecular_Name); % レジェンド名
end
```

ode15sを使って、下のODE(t, y, param)に
time (計算時間), y0 (初期値) の条件で
シミュレーション

分子の値
時間

中身を表示するとこんな感じ

→X軸にt, Y軸にtime_courseを図示

時間

t <101x1 double>	
	1
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10
12	11
13	12
14	13
15	14
16	15

分子の値

time_course <101x3 double>			
	1	2	3
1	0.5000	1	0
2	0.4557	0.9557	0.0443
3	0.4209	0.9209	0.0791
4	0.3934	0.8934	0.1066
5	0.3715	0.8715	0.1285
6	0.3540	0.8540	0.1460
7	0.3400	0.8400	0.1600
8	0.3287	0.8287	0.1713
9	0.3197	0.8197	0.1803
10	0.3123	0.8123	0.1877
11	0.3064	0.8064	0.1936
12	0.3015	0.8015	0.1985
13	0.2977	0.7977	0.2023
14	0.2945	0.7945	0.2055
15	0.2920	0.7920	0.2080
16	0.2899	0.7899	0.2101

各t, time_courseとも100秒まで

* Matlab上で%以下はコメントアウトされる。
各行の出力を非表示にするため;を入れる。

モデルの作成 1 ー分子間相互作用ー

① モデルの作成（続き）

```
13 - title('MI'); % タイトルのラベル名
14 - legend(Molecular_Name); % レジェンド名
15 - end
16
17 - function dydt = ODE(t,y,param) % ODE(t,y,param)という関数の定義
18 -     A = y(1); % Aの濃度
19 -     B = y(2); % Bの濃度
20 -     AB = y(3); % ABの濃度
21
22 -     Kf = param(1); % Kfはparamの1番目の値
23 -     Kb = param(2); % Kbはparamの2番目の値
24
25 -     dydt(1,:) = -Kf*A*B+Kb*AB; % dydtの1行(分子A)目の式の定義
26 -     dydt(2,:) = -Kf*A*B+Kb*AB; % dydtの2行(分子B)目の式の定義
27 -     dydt(3,:) = Kf*A*B-Kb*AB; % dydtの3行(分子AB)目の式の定義
28 - end
```

微分方程式の定義

警告を表示：カーソルをあてることで内容が確認できる

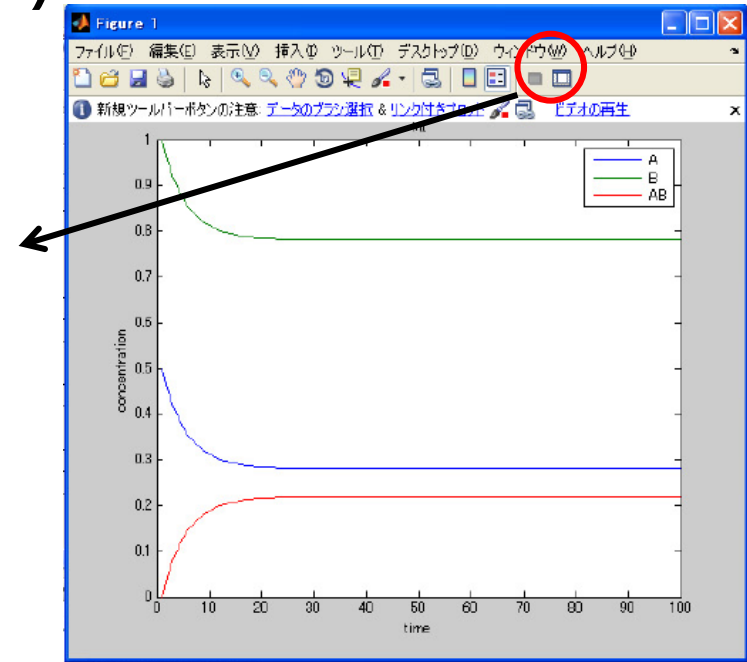
モデルの作成 1 ー分子間相互作用ー

② プログラムの実行

- 1, まずはデスクトップに夏期実習というフォルダを作成（今後、使用）
- 2, ファイルを保存（MIという名前で保存して下さい：ファイル＞保存）
- 3, コマンドウィンドで MIと打ち、リターンを押すことで実行
（エディタの上の緑の三角ボタンでもO.K.）

③ 図の確認

- ・ figure windowが開き、グラフを表示
- ・ 細かい設定を変えたければ（x, y軸の変更等）ここをクリックしてプロットツールを表示する。



④ 課題（1）：適当な言葉を入れよ

初期値が同じ場合、同じKdの値を持つ[Kf, Kb]の組み合わせにおいてグラフの（A）は同じとなるが、（A）に達するまでの（B）が異なる。このような条件の場合、Kbの値が（C）ほど（A）に達する（B）は早くなり、（D）ほど遅くなる。
→反応の刺激に対する応答性に関与

$$* Kd = Kb / Kf$$

モデルの作成 1 ー分子間相互作用ー

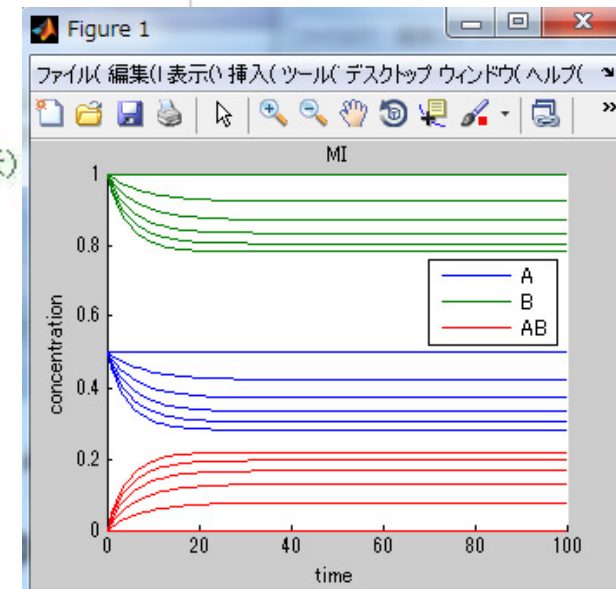
- ⑤ 検討 2 for loop (繰り返しの演算) を用いてパラメータの検討を行う
(例: いろいろなKfを試してA, B, ABの挙動の変化をみる (Kb固定))

http://kurodalab.bi.s.u-tokyo.ac.jp/class/Summer/2019/Day1/MI_loop.m

```
7 - Molecular_Name = {'A','B','AB'}; % 測定分子
8 - figure; % figure windowの形成
9 - hold on; % 図を固定
10 - for n=0:1:5; % nを0から5まで1刻みで変動
11 -     param(1)=n*0.02; % param(1)(=Kf)をnを用いて表現
12 -     [t,time_course] = ode15s(@(t,y) ODE(t,y,param),time,y0); % ODE(t,y,param)を解く
13 -     plot(t,time_course);
14 - end
15 - xlabel('time'); % x軸のラベル名
16 - ylabel('concentration'); % y軸のラベル名
17 - title('MI'); % タイトルのラベル名
18 - legend(Molecular_Name); % レジェンド名 (上記のMolecular_Nameを表示)
19 - end
```

⑥チャレンジ課題

⑤ではKfを動かした。⑥ではKbも動かしてみる。
ただし、Kdは一定とする。



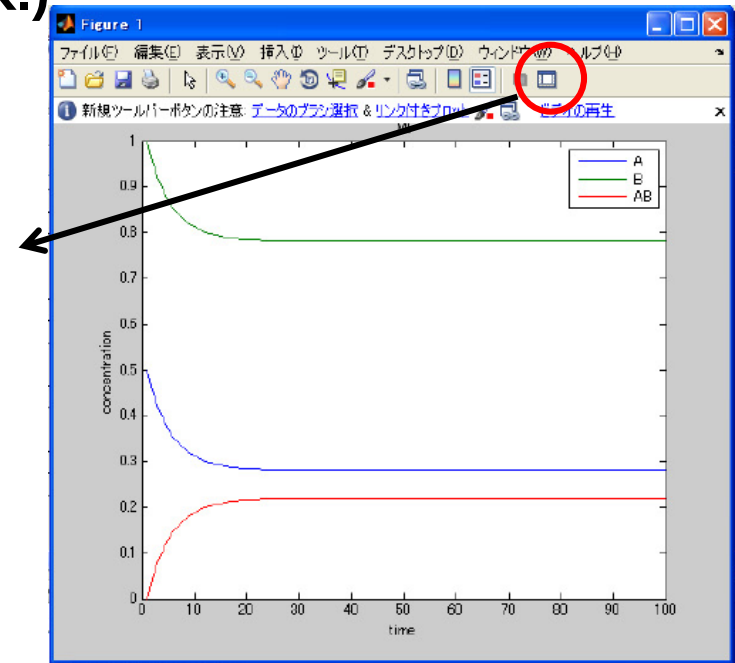
モデルの作成 1 ー分子間相互作用ー

② プログラムの実行

- 1, まずはデスクトップに夏期実習というフォルダを作成（今後、使用）
- 2, ファイルを保存（MIという名前で保存して下さい：ファイル＞保存）
- 3, コマンドウィンドで MIと打ち、リターンを押すことで実行
（エディタの上の緑の三角ボタンでもO.K.）

③ 図の確認

- ・ figure windowが開き、グラフを表示
- ・ 細かい設定を変えたければ（x, y軸の変更等）ここをクリックしてプロットツールを表示する。



④ 課題：適当な言葉を入れよ

初期値が同じ場合、同じKdの値を持つ [Kf, Kb]の組み合わせにおいてグラフの（A**最終値**）は同じとなるが、（A）に達するまでの（B **時間**）が異なる。このような条件の場合、Kbの値が（C**大きい**）ほど（A）に達する（B）は早くなり、（D**小さい**）ほど遅くなる。

→反応の刺激に対する応答性に関与

$$* Kd = Kb / Kf$$