MATLAB の使用方法

コマンドラインに

```
>> a = 1
```

と入力すると、変数 'a' に実数値'1' が代入される。MATLAB の変数は基本的に行列やベクトルであり、例えば

```
>> b = [ 1 2 3 ]
```

によって横ベクトル

```
>> c = [ 1; 2; 3 ]
```

によって縦ベクトル

```
>> D = [ 1 2 3; 4 5 6; 7 8 9 ]
```

によって行列を代入することができる。
行列の各成分には、

```
>> e = D( 3, 1 )
```

のようにしてアクセスできる。
等差数列成分をもつベクトルは次のような作りかたができる。

```
>> b = 1:1:3
```

これは 1 からスタートして 1 ステップ刻みで 3 までの数列ベクトル、を意味し、

```
>> b = [ 1 2 3 ]
```

と同じ意味である。
ステップが 1 であるときは省略して、

```
>> b = 1:3
```

と書くこともできる。
行列の成分へのインデックスに、自然数値ベクトルを使うことができる。

```
>> b0 = D( 1, 1:3 )
>> b1 = D( 2, [1 2 3] )
>> B2 = D( [1 3], [1 3] )
```

行列の第 1 行目の横ベクトル全体を、

```
>> b = D( 1, : )
```

と書くこともできる。

```
>> D( :, : )
```

とすれば行列全体である。行列やベクトル同士のかけ算は成分に分解せずに、

```
>> f = b * c
>> g = D * c
```

のように行列まるごとでおこなうことができる。また、

```
>> h = D * b
```

のような不可能な演算にはエラーメッセージを出す。
次元の等しい行列 (ベクトル) の各要素に関するかけ算割り算が、

```
>> a = [ 1 2 ]
>> b = [ 1 3 ]
>> c = a ./ b
>> d = a .* b
```

のようにして可能である。
行列 (ベクトル) にスカラーを足すと、全成分に足したことになる。

```
>> a + 1
```

関数はひととおり揃っている。

```
>> sin( 2*pi )
>> log( exp( 5 ) )
>> sqrt( 2 )
```

スカラー値関数に行列を入れると、各成分が演算される。

```
>> log10( [ 10 100; 1 0.1] )
```

行列関数というのもいろいろある。

```
>> A = [ 8 2; 2 10 ]
>> sum( A )  % 和
>> A'        % 転置
>> min( A )  % 最小値
>> inv( A )  % 逆行列
>> eig( A )  % 固有値
```

行列をつくる関数もいろいろ。

```
>> eye( 3 )     % 単位行列
>> zeros( 3,2 ) % ゼロ値行列
>> rand( 3,3 )  % 一様乱数行列
>> randn( 2,4 ) % 正規乱数行列
```

ベクトルのプロットも簡単。

```
>> x = -pi:0.1:pi
>> y = sin( x )
>> plot( x, y )
```

計算結果画面出力の抑制には、行末にセミコロンをつける。変数の中身が知りたいときは、変数名を打ちこめばよい。

```
>> X = 1:10 ;
>> X
```

オンラインヘルプによって各種関数の解説が得られる。

```
>> help sum
>> help plot
```

MATLABのソースファイルはその名前に拡張子'.m'がつき、m-filesと呼ばれる。その実体はテキストファイルであり、メモ帳などのエディタで編集ができる。MATLAB標準のソース編集ソフトウェアを使うのも良い。

m-filesには、scriptとfunctionの二種類がある。
script fileはその file名から'.m'を除いたものをコマンドラインに入力することで実行される。script fileの実行は、fileに書かれた各行をコマンドラインから入力することと同じ意味を持つ。
function fileも基本的に同様だが、引数を持つ。また出力以外の内部変数は隠蔽されており外から参照できない。
詳しくはサンプルを参照のこと。

演習 1：RBF

コマンドプロンプトで

```
>> RBF1demo
```

と入力すると、グラフが出力される。横軸が x 軸、縦軸が y 軸であり、青色の分布点は学習データセット、赤色の折れ線はパラメータ学習後のRBFの出力である。緑色の線分は基底関数の位置と大きさを表しており、中心が基底中心 μ_j、長さが 2σ である。

ここでご覧いただいた`RBF1demo`の実体は、ソースファイル`RBF1demo.m`である。ソースファイルをエディタで開いて読みながら、以下の解説を読みすすめられたい。

> **注意事項：** MATLABのStudent's editionを使用の場合、メモリ制限のためにエラーを発生する場合がある。エラーが発生したときは問題のサイズを決定する定数 (`RBF1demo.m`では`Ndata`及び、`M`) がデモプログラムの最初で定義されているので、これを編集して変更すること。

`RBF1demo.m`の前半では、学習データ $(x(t), y(t)), t = 1, \ldots, T$ と基底の中心位置 μ_j と基底の分散半径 σ が定義される。デフォルトとして学習データはabs(sin)曲線に小さな正規ノイズが乗ったもの。基底個数 $M = 20$ 個、基底中心はその x-rangeを等間隔に切るようなもの、基底分散半径は x-rangeに比例し基底個数に反比例するような値、にそれぞれ設定されるようになっている。これらは任意に変更可能であるがまず、基底の個数 M の値を変えることによって、関数近似の具合の変化を見よ。

後半では、これらを引数とする関数`RBF1learn`によって最小二乗法にもとづくパラメータ a_j の推定を行い、`RBF1out`によってその関数形を計算し、`RBF1plot`によってグラフにプロットする。

まず`RBF1out.m`を読んでこれがテキスト (6) 式

$$y = \sum_{j=1}^{M} a_j \exp\left(-\frac{1}{2\sigma^2}|x - \mu_j|^2\right)$$

で定義される出力を行うことを確かめよ。

次に`RBF1learn`は最小二乗法によるパラメータ推定を行うメインのアルゴリズムである。このプログラムの中で、

$$k_{j,t} = \exp\left(-\frac{1}{2\sigma^2}|x(t) - \mu_j|^2\right)$$

を補助的に用いつつ、テキスト (9a)(9b) 式

$$z_i \equiv \frac{1}{T}\sum_{t=1}^{T} k_{i,t} y(t)$$

$$K_{i,j} \equiv \frac{1}{T}\sum_{t=1}^{T} k_{i,t} k_{j,t}$$

を計算し、(12) 式

$$a = K^{-1} z$$

により、パラメータを求めている。このことを確かめよ。

演習 2: AR法

デモ`AR.m`を実行して出力されるプロットについて、青の分布点は学習データセットであり、赤のラインは学習データセットの前半部のみにもとづいて、後半部をAR法によって予測したものである。

デモプログラムの前半で、データ数`Ndata`とAR次数`M`が定義されている。まずはこれをいろいろと変えて試してみよ。

デモに用いられているデータセット`x`は次のように定義されている。

```
x = sin(t*4*pi) ...
    + sin(t*4.1*pi) ...
    + randn(N*2,1)*0.1;
```

これは適当なsin関数の重ね合わせに、正規ノイズを加えたものである。データセットの大きさは、`2*N`である。これを後半で、学習用と予測評価用とで`N`ずつに分けて用いる。`sin`の中身を変えることによって、波形を変えることができる。また`randn(N*2,1)*0.1`の項は中心値0標準偏差0.1の正規ノイズを成分として持つ $2N$ 次元の縦ベクトルを作っており、この0.1とされている値を変更するとノイズの大きさを変えられる。[1]

`AR_estimate.m`はパラメータ推定を行う。アルゴリズムは以下のとおりである。

$$A_{i,j} = \sum_{t=M}^{T-1} x(t-i+1)x(t-j+1)$$

$$b_j = \sum_{t=M}^{T-1} x(t+1)x(t-j+1)$$

$$a = A^{-1} b$$

ARからはデータ列`x`の前半`N`個のデータが`AR_estimate`に入力されているので、上式 T の値は`N`と等しい。

[1] ノイズを0とすると、パラメータ推定の計算が発散してエラーを出すが、これはAR法それ自体が持つ弱点である。

参考： サンプルプログラムの中で、`t = M:(T-1);` はデータインデックス列 $t = \{M, M+1, \ldots, T-1\}$ を意味する。これを用いて `x(t)` と書くと、これはデータ列 $\{x(M), x(M+1), \ldots, x(T-1)\}$ を意味する。したがって、`A(i,j) = x(t-i+1)'*x(t-j+1);` の右辺は `t` に関する和を、ベクトルの内積によって計算していることになる。MATLAB では行列演算をするほうが `for` 文でループを回すよりも高速であるので、こういう書きかたをすることがある。

`AR_predict.m` はパラメータに基いて、データの続きを予測する。

演習 3: EM 法によるクラスタリング

デモ `EM.m` を実行せよ。スペースキーを押すと、E-step と M-step が 1 epoch ずつ進行する。収束したと思ったら、ctrl-C によって止めること。何度か実行して、毎回同じ解を得ることができているか、確かめよ。

黄色の分布点は、2 次元散布点の学習データセット $\{x(t)|t = 1, \ldots, T\}$ であり、赤円は各クラスタ中心を中心点とし標準偏差 σ を半径とした円を表す。

`EM.m` において、データ数 `Ndata` およびクラスタ数 `M` の値を様々に変えて試してみよ。

メインプログラム `EM.m` はモデルパラメータの初期化を行なう `EM_init.m` を呼んだ後、E-step,M-step を行なう `EM_dostep` を繰り返し呼びつつ、`EM_mixGaussPlot` によって計算経過を表示している。

モデルパラメータ及び十分統計量は構造体変数 `Model` に格納されるが、これはテキストで使用されているノーテーションと次のような対応関係を持つ。

$$< 1 >_i = \texttt{Model.r(i).z}$$
$$< x >_i = \texttt{Model.r(i).zx}$$
$$< |x|^2 >_i = \texttt{Model.r(i).zx2}$$
$$\mu_i = \texttt{Model.p(i).mu}$$
$$\sigma_i = \texttt{Model.p(i).sigma}$$
$$\nu_i = \texttt{Model.p(i).nu}$$

`EM_init` では上の構造体を構築しつつ、その値の初期化を行っている。初期化に当たって、μ_i すなわち `Model.p(i).mu` の値については、データ点からランダムに `m` 個取り出してきてその値とした。σ_i は全クラスター律に 0.1 という値、ν_i も全クラスター律に $1/m$ という値を与えた。とくに σ_i の値は任意に調整が可能であるので、試してみよ。

`EM_dostep` では E-step を (45) 式

$$P(i|x(t), \bar{\theta}) = \frac{\exp(-\frac{1}{2\sigma^2}|x(t) - \bar{\mu}_i|^2)}{\sum_{j=1}^{M} \exp(-\frac{1}{2\sigma^2}|x(t) - \bar{\mu}_i|^2)} = \texttt{z\_i(i,t)}$$

で計算する。(上式の分母=`LL_i(i,t)`、分子=`LL(t)` である) それに基づいて M-step を (49abc) と (51) 式すなわち、

$$< 1 >_i = \frac{1}{T}\sum_{t=1}^{T} \texttt{z(i,t)}$$
$$< x >_i = \frac{1}{T}\sum_{t=1}^{T} \texttt{x(:,t)*z(i,t)}$$
$$< |x|^2 >_i = \frac{1}{T}\sum_{t=1}^{T} \texttt{x(:,t)'*x(:,t)*z(i,t)}$$
$$\mu_i = < x >_i \; / \; < 1 >_i$$

によって計算して構造体 `Model` を更新している。

演習課題 3-2 分散 σ^2 を可変パラメータとしたとき、M-step において

$$\frac{\partial Q}{\partial\{\sigma^2\}} = 0$$

から $\{\sigma^2\}$ の更新式を求めよ。ヒントはどこかにあります。

演習 4: Kalman filter

```
>> cd pendulum
```

によってディレクトリ移動したのち、デモ `pendulum_demo` を実行する。

二つのグラフが並べて表示されるが、上のグラフはカルマンフィルターの結果であり講義ノートに準拠している。下のグラフは上とは独立に、最初のデータ点だけを用いて行ったカルマン予測の結果である。内部変数 $z(t)$ の真値 `true`、観測値 $x(t)$`observed` に重ねて、上のグラフではフィルターによる期待値 $\hat{z}(t)$ とその分散が、下のグラフでは予測による期待値 $\tilde{z}(t)$ とその分散が、それぞれ表示されている。記号と線の意味はグラフ上の凡例を見ること。

力学モデルその他各種パラメータを変えて試してみるには `pendulum_demo.m` を編集すればよい。とくに、データ数 `Ndata` を変えてみよ。

`pendulum_demo.m` のソースコードでは力学モデルの定義の後、カルマンフィルター `kalman_filter.m`、およびカルマン予測 `kalman_predict.m` をそれぞれ呼び、その結果を上下に分けて表示している。

`kalman_filter.m` は、特定時刻の推定値 $\hat{z}(t), Q(t)$ と観測データ $x(t+1)$ から、次の時刻の推定値 $\hat{z}(t+1), Q(t+1)$ を求める `kalman_update.m` を各時刻ごとに呼ぶことでフィルターを実現している。その結果が上のグラフに表示される。ここで使われる `kalman_update.m` は、テキストの式 (62)(61b)(69b)(69c)(69a) によって各時刻の推定値を更新している。ソースコードから、このことを確かめよ。

次に、`kalman_predict.m` は状態変数の初期値 $x(1)$ だけが与えられているとき、その後の状態変数の時間遷移を推定する。その結果は下のグラフに表示される。これのアルゴリズムは簡単であって、特定時刻の推定値 $\hat{z}(t), Q(t)$ から、

$$\hat{z}(t+1) = A\hat{z}(t)$$
$$Q(t+1) = U + AQ(t)A'$$

によって次の時刻の推定値を逐次的に求めていくだけである。ソースコードからこれを確かめよ。