R でシミュレーションしてみよう

重要なパラメータは３つ

- α：自己相関の大きさ ($-1 < \alpha < 1$ で定常 (＝平均０），自己相関係数）
- σ_p：プロセス誤差の大きさ（ランダムウォークの振れ幅）
- σ_o：観測誤差の大きさ

システムモデル

$$x_{t+1} = \alpha x_t + \mathrm{N}(0, \sigma_p)$$

観測モデル

$$y_t = x_t + \mathrm{N}(0, \sigma_o)$$

R のコード

$$x_{t+1} = \alpha x_t + \mathrm{N}(0, \sigma_p)$$

$$y_t = x_t + \mathrm{N}(0, \sigma_o)$$

```
alpha <- 0.5; sigmap <- 0.3; sigmao <- 0.5 # パラメータ

x <- y <- numeric() # 空のベクトル
x[1] <- 1 # x の初期値
N <- 100 # 時間の長さ
# システムモデル
for(i in 1:(N-1))  x[i+1] <- alpha*x[i] + rnorm(1,0,sigmap)
# 観測モデル
for(i in 1:N) y[i] <- x[i] + rnorm(1,0,sigmao)
matplot(cbind(x,y),type="l",ylab="x",xlab="time")
```

AMDB による推定：ADMB とは？

- 多数のパラメータ推定に特化したフリーのプログラミング言語
- `http://admb-project.org/`
- 多くのパラメータの推定を必要とする資源評価モデルでも活用
- テストコード (test1.tpl, test2.tpl) を紹介
- R でもできる (dlm など）が，ADMB で書くことによって，カウントデータ（negative binomial) や 0/1 データへの拡張が容易（尤度関数の部分を書き換えるだけでよい）

ADMB の構造

- データを読み込む（DATA_SECTION）
- パラメータのタイプを決定（PARAMETER_SECTION）
- 計算の実施（PROCEDURE_SECTION）
- 他には、INITIALIZATION_SECTION, REPORT_SECTION, TOP_OF_MAIN_SECTION など
- 関数の定義（FUNCTION）も

データの読み込み（Data_Section）

(init かどうか？)_(型) 変数名
init = 読み込むデータ

```
DATA_SECTION
  init_int phase1 //  beta を推定するか？
  init_int n // 時系列の長さ
  init_vector y(1,n) // 時系列データ
```

パラメータの定義（PARAMETER_SECTION）

(init かどうか？)_(オプション)_(型) 変数名
init = 推定するパラメータ

```
PARAMETER_SECTION
  // 推定するパラメータ（init_と頭につく）
  init_bounded_number logit_beta(-3,10,1)
  init_bounded_number log_sigma1(-10,3,1)
  init_bounded_number log_sigma2(-10,3,1)

  number beta;  // その他のパラメータ
  number sigma1;
  number sigma2;

  random_effects_vector x(1,n,1) // ランダム変数
  objective_function_value f  // 最小化する値
```

実際の計算（PROCEDURE_SECTION）

```
PROCEDURE_SECTION
  beta = 1/(1+mfexp(-logit_beta));
  sigma1 = mfexp(log_sigma1);
  sigma2 = mfexp(log_sigma2);

  // システムモデル
  // 尤度の計算（x_i と beta*x_i-1 の差が正規分布に従う）
  for (int i=2;i<=n;i++){
    f -= -log(sigma1) -.5*square((x(i)-beta*x(i-1))/sigma1);
  }

  // 観測モデル
  // 尤度の計算（y_i と x_i の差が正規分布に従う）
  for(i=1;i<=n;i++){
    f -= -log(sigma2) -.5*square((y(i) - x(i))/sigma2);
  }
```

ADMB の実行 (test1.tpl)

コマンドプロンプトにて

- $>$ admb -r test1 // コンパイル $\rightarrow$ test1.exe というファイルができる
- $>$ test1.exe // 実行

R から

- $R >$ shell(ädmb -r test1¨) // コンパイル $\rightarrow$ test1.exe というファイルができる
- $R >$ shell(¨test1.exe¨) // 実行

いろいろなファイル

実行に必要なファイル

- ファイル名.tpl # プログラムファイル
- ファイル名.dat # 読み込むデータ
- （ファイル名.pin # パラメータの初期値。なくてもよい）

結果を示すファイル

- ファイル名.par # 推定されたパラメータ
- （ファイル名.rep # REPORT_SECTION の結果）

R によるシミュレーション＆推定 1

```
#------ simulation ------------
alpha <- 0.5; sigmap <- 0.3; sigmao <- 0.5 # パラメータ

x <- y <- numeric() # 空のベクトル
x[1] <- 1 # x の初期値
N <- 100 # 時間の長さ
# システムモデル
for(i in 1:(N-1))  x[i+1] <- alpha*x[i] + rnorm(1,0,sigmap)
# 観測モデル
for(i in 1:N) y[i] <- x[i] + rnorm(1,0,sigmao)

#------ estimation ------------
write(c(1,N,y),file="test1.dat")
shell("admb -r test1") # 毎回は必要ない
shell("test1.exe")
pars <- scan("test1.par") # 推定されたパラメータの読み込
み
```

R によるシミュレーション＆推定 2

```
#------ show result ------------
logit <- function(x)  1/(1+exp(-x))
c(logit(pars[1]),exp(pars[2:3]))
matplot(cbind(x,pars[-1:-3],y),type=c("l","l","b"),col=1:2)
```

おまけ (test2.tpl): SEPARABLE_FUNCTION の利用

- 普通のコードよりも高速
- SEPARABLE_FUNCTION 内で尤度関数を定義する
- 様々な制約がある
  - SEPARABLE_FUNCTION の行は改行してはいけない
  - 変数変換などはすべて SEPARABLE_FUNCTION 内で行う（= int_のパラメータ以外は, SEPARABLE_FUNCTION 外で定義したものを使えない）
  - 他にも？

おまけ (test2.tpl): SEPARABLE_FUNCTION の利用

```
PROCEDURE_SECTION
  for (i=2;i<=n;i++)
  {
    sf1(log_sigma1,logit_beta,x(i),x(i-1),i);
  }
  for(i=1;i<=n;i++){
    sf2(log_sigma2,x(i),i);
  }

SEPARABLE_FUNCTION void sf1(const dvariable& ls,  const dvariable& lcoef,
   dvariable coef = 1/(1+mfexp(-lcoef));
   dvariable sigma = mfexp(ls);
   f -= -log(sigma) -0.5*square((u2-coef*u1)/sigma);

SEPARABLE_FUNCTION void sf2(const dvariable& ls, const dvariable& ui, int
   dvariable  sigma = mfexp(ls);
   f -= -log(sigma) -.5*square((y(i) - ui)/sigma);
```

R での状態空間モデル

- パッケージ（dlm）
- なぜか ADMB と結果が違うが，原因はまだ不明です（初期値の設定の違い？）

```
library(dlm)
build.1<-function(theta){
   dlmModPoly(order=1,dV=exp(theta[1]),dW=exp(theta[2]))
}
fit.1<-dlmMLE(y,parm=c(1,1),build.1)
tmp <- build.1(fit.1$par)
par2 <-dlmFilter(y,tmp)
points(as.numeric(par2$m[-1]),type="l",col=2,lty=2)
legend("bottomright",col=c(2,2,1),legend=c("ADMB","dlm","Tr
```