生物資源解析学演習 2022 Lecture 1

イントロダクション: R の使い方基礎

北門　利英（東京海洋大学海洋生物資源学科）

2022 年 10 月 4 日

目次

Point:

- Lecture 1 はイントロですので気楽に R の使い方の基本について学んで下さい．（今回は宿題や提出課題もありませんので）

1 はじめに

1.1 R のインストール

Windows の場合，R の下記サイトで”Download R for Windows” を選択し，“Base” の”Download R 4.x.x for Windows “を選択してインストーラーファイルをダウンロードすれば，自宅などの PC に簡単にインストールできます．(Mac も同様です)

https://cran.r-project.org/

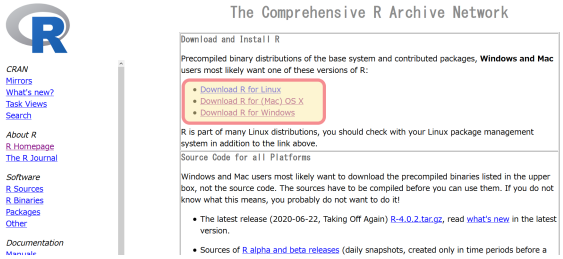


図1 R のダウンロードサイト

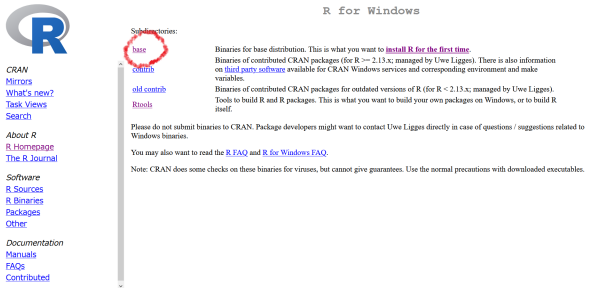


図2 Base を選択

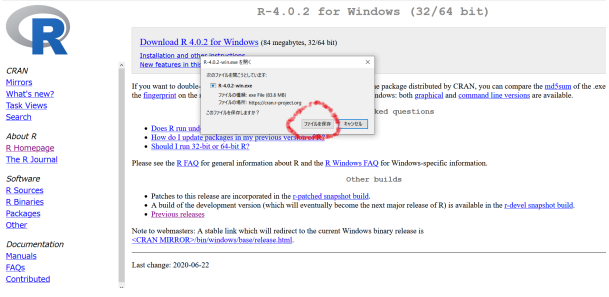


図3 インストーラーをダウンロードし実行

インストーラーをダウンロード後、ダブルクリックでインストールを開始してください。

インストールの際，”MESSAGE TRANSFORMATION” のチェックを外してください．そうすることで，Rstudio 使用時に問題が発生する可能性が低くなります．

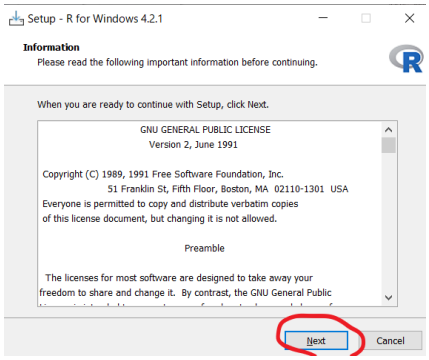


図4 “R intallation: step 1”

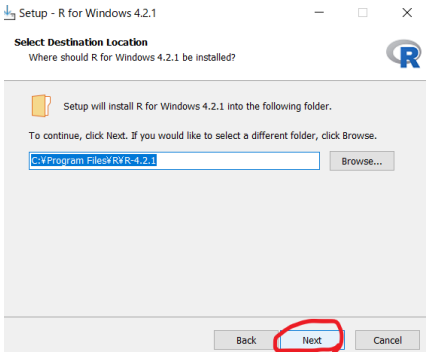


図5 “R intallation: step 2”

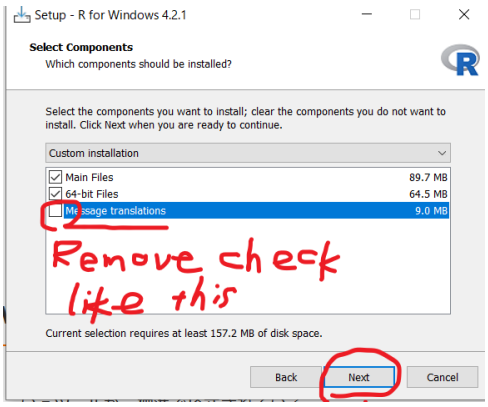


図6 “R intallation: step 3”

1.2 Rstudio のインストール（参考までに）

R を使うための大変便利な Rstudio というソフトがあります．ごく稀にインストールがうまくできないことがありますので，授業では状況に応じて R と Rstudio の併用という形で進める予定です．

https://rstudio.com/products/rstudio/download/#download

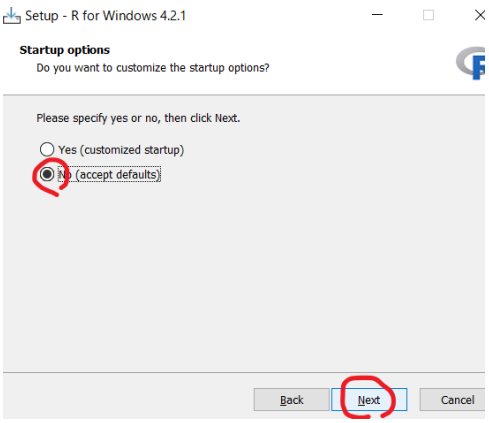


図7 “R intallation: step 4”

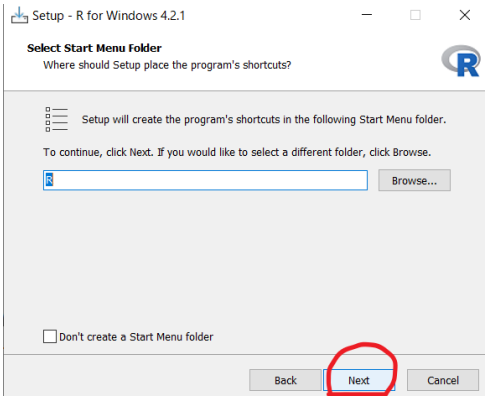


図8 “R intallation: step 5”

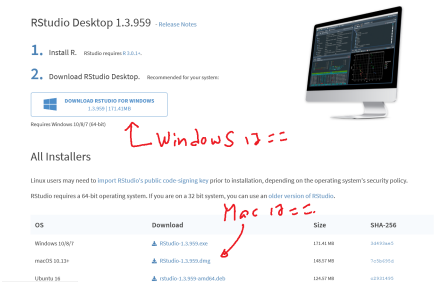


図9 Rstudio のサイト

2　R を用いた簡単な計算 (1) 第一歩

- デスクトップの R のアイコンをダブルクイックして R を立ち上げる
- ためしに以下の計算をしてみる

入力の仕方は 2 通り：

- 直接 R のコンソール画面に入力（コードを打ち間違えたときに，いちからやり直しのが面倒．打ち間違えて入力し直すときには Esc キーを押していったんその行を無効にしてから再度入力してください）

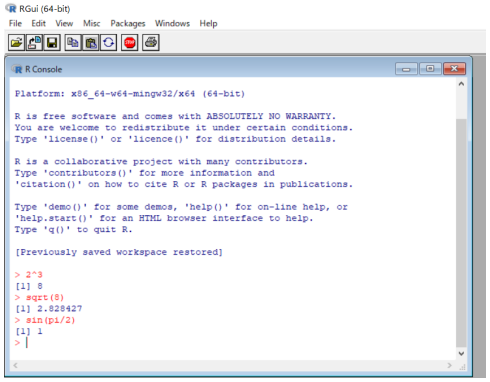


図10　直接入力する場合

- R 付属のエディターにコードを入力し，実行したいコードを選択して，「コントロール +R」で実行．

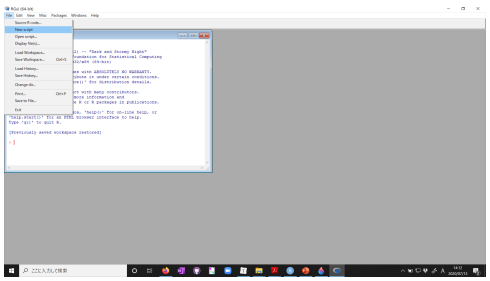


図11　R のエディターを利用する場合

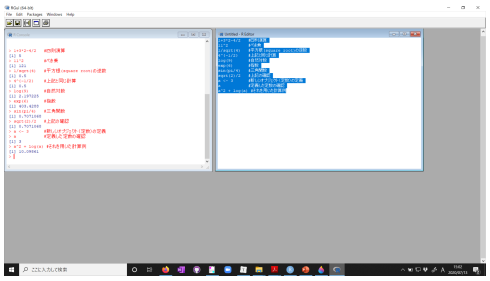


図12　実行したいコードを選択して，「コントロール +R」で実行

2.1 基本演算

```
1+3*2-4/2    #四則演算
```

```
[1] 5
```

```
11^2         #べき乗
```

```
[1] 121
```

```
1/sqrt(4)    #平方根（square root）の逆数
```

```
[1] 0.5
```

```
4^{-1/2}     #上記と同じ計算
```

```
[1] 0.5
```

```
log(9)       #自然対数
```

```
[1] 2.197225
```

```
exp(6)       #指数
```

```
[1] 403.4288
```

```
sin(pi/4)    #三角関数
```

```
[1] 0.7071068
```

```
sqrt(2)/2    #上記の確認
```

```
[1] 0.7071068
```

```
a <- 3       #新しいオブジェクト（定数）の定義
a            #定義した定数の確認
```

```
[1] 3
```

```
a^2 + log(a) #それを用いた計算例
```

```
[1] 10.09861
```

2.2 ベクトル（配列）の扱い

```
a <- c(1,3,5,7,9)    #新たにベクトルをオブジェクトとして定義
a                    #定義したベクトルの確認
```

```
[1] 1 3 5 7 9
```

```
sum(a)               #合計
```

```
[1] 25
```

```
mean(a)              #平均
```

```
[1] 5
```

```
a+10                 #ベクトルのすべての要素に 10 を加える
```

```
[1] 11 13 15 17 19
```

```
5*a                  #ベクトルのすべての要素を 5 倍する
```

```
[1]  5 15 25 35 45
```

```
a^2                  #ベクトルのすべての要素を 2 乗する
```

```
[1]  1  9 25 49 81
```

```
log(a)               #ベクトルのすべての要素の対数を取る
```

```
[1] 0.000000 1.098612 1.609438 1.945910 2.197225
```

```
a[3]           #ベクトル a の 3 番目の要素を取り出す
```

```
[1] 5
```

```
a[c(1,3,5)]    #ベクトル a の 1,3,5 番目の要素を取り出す
```

```
[1] 1 5 9
```

```
a[-c(2,4)]     #ベクトル a の 2,4 番目以外の要素を取り出す（上記と同じ）
```

```
[1] 1 5 9
```

```
b <- seq(10,30,5) #同じ長さのベクトル b を新たに定義（"seq(x,y,z)" は x から y まで z 刻みで）
a+b  #要素どおしの足し算
```

```
[1] 11 18 25 32 39
```

```
a*b  #要素どおしの掛け算
```

```
[1]  10  45 100 175 270
```

```
b/a  #要素どおしの割り算
```

```
[1] 10.000000  5.000000  4.000000  3.571429  3.333333
```

2.3 繰り返し計算

1 から 10 までの足し算を行う計算例.

```
ss <- 0
 for(i in 1:10){
  ss <- ss + i
}
ss
```

```
[1] 55
```

表1　初等的な演算および関数

算術演算	コマンド
足し算，引き算, 掛け算，割り算	+, - , *, /
べき乗	^ , **
四捨五入，切り捨て，切り上げ	round，floor，ceiling
初等関数	**コマンド**
絶対値	abs()
平方根	sqrt()
指数関数	exp(x)
対数関数	log(x，base=exp(1))
三角関数	sin()，cos()，tan()
論理演算	**コマンド**
論理演算子 (等号，不等号)	==，!=，!
論理演算子 (不等号)	<=，>=，<，>
論理演算子 (and)	&，&&
論理演算子 (or)	|，||

```
xxx <- seq(1,10,1)
sum(xxx)
```

```
[1] 55
```

ここで一旦作業を終了し，これまでの内容を保存してみる．保存の方法はいくつかあるが，作業内容毎に独立したフォルダーを作成するのが便利である．そこで，仮にデスクトップに「生物資源解析学演習」という名前のフォルダーの中に「Lecture01」を用意してあるとし (名前と場所は任意である)，ここに作業内容を保存しみてる．R で「ファイル」→「ディレクトリの変更」で先ほど用意したフォルダーを指定し「ＯＫ」を押す．次に，R のウインドウの右上の「×」印を押すか，コンソールで quit()と入力して R をいったん終了させると，フォルダー内に「.Rdata」というファイルが作成される．このアイコンをダブルクリックすると先ほどの続きから作業を再開することができる．また，あるフォルダー内に保存したデータファイルを利用する場合も，これを作業フォルダーとして選択するか，あるいは作業フォルダーにデータファイルを保存することで，ファイルへのアクセスのし易さが格段に向上する (後述の「データの入出力と編集」を参照)．

3　R を用いた簡単な計算 (2) いろいろな関数

3.1　R のオブジェクトと関数

前節でa というベクトルを定義しこれをオブジェクト (目的物) とよんだ．R ではこのようなベクトルや行列，それにデータセット全体といったデータを保存したオブジェクトをデータオブジェクト，そして sum，mean などの関数を関数オブジェクトとよび区別する．オブジェクト名は任意に付与できるが，ケーススペシフィック (大文字と小文字を区別する) であり幾つかの禁則文字もある．もちろん，アルファベットと数字を混合させてもよい．

R には多くの基本関数の他，解析やプログラミングを行う非常に多くの便利な関数が備わっている．また，フリー言語の特性を活かしてその発展も日進月歩である．また，関数は自作することもできる．例えば，ベクトルの b 乗と c 乗の和を計算する関数を作りたいとする．そこで名前をsum.power とし次のように定義する．

```
sum.power <- function(x, b=1, c=2){
 obj1 <- sum(x^b)
 obj2 <- sum(x^c)
 obj <- c(obj1, obj2)
 obj
}
```

そして，

```
sum.power(x=a, b=1, c=2)
```

```
[1]  25 165
```

```
sum.power(x=a, b=2)
```

```
[1] 165 165
```

```
sum.power(x=a^2, b=1/2)
```

```
[1]   25 9669
```

とa の要素をすべて b, c 乗し足し合わせた値が返される．なお上記は

```
sum.power(a, b=1, c=2)
```

は

```
sum.power(a, 1, 2)
sum.power(a)
```

としても同じである (変数指定がない場合は定義した際の引数の順番通りに読み込まれる．関数を定義するときに値を入れている場合，指定が無いとデフォルト値としてそれが利用される)．

一般に R の関数は次のように定義される．

```
関数名（引数1=引数1の入力値，引数2=引数2の入力値...）
```

上記のsum.power の場合には 2 つの異なるタイプの引数が定義されていて，x はベクトルで入力が必要であるが，b にはあらかじめ $b = 1$ という値がデフォルトで割り当てられているため (これを仮引数とよぶ)，

```
sum.power(a)
```

```
[1]  25 165
```

のように指定がなければ単にベクトルの和を計算することになる．

このほか，特定の解析や手法のために開発された関数や例となるデータセットを 1 つにまとめたパッケージも大変豊富であり，これらも先述の R のサイトからダウンロード可能である．

3.2 R における演算と基本関数

R による演算や基本関数の例は既に述べているが，初等的なものを以下にまとめる．

3.3 R のデータタイプと要素の抽出

R では，データとしてスカラー，ベクトル，行列の他にさらに高次の配列を定義することができる．さらに，それぞれの要素に整数，実数，文字列など異なった型を割り当てることもできる．例えば，行列A を 2×5 の行列とし，1 行目を先ほどのa, 2 行目に適当に長さ 5 のベクトルを入れ，さらに行と列に名前を付けてみよう．

```
A <- matrix(0, nrow=2, ncol=5)  # すべての成分を 0 とする 2×5 の行列
A[1,] <- a                      #A の 1 行目に a を割り当てる
A[2,] <- 5:1                    #A の 2 行目に c(5,4,3,2,1) を割り当てる
A                               # 確認のために出力
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    5    4    3    2    1
```

ここでA[1,]などは行列からベクトルを抽出することを意味し，同様にして，この行列から一部の列や行だけを取り出して部分行列を改めて定義することもできる．

```
A[1,5]                          #1 行 5 列成分の取り出し
```

```
[1] 9
```

```
Asub1 <- A[, c(1,3,5)]; Asub1   #1,3,5 列目の列の取り出し
```

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    5    3    1
```

```
Asub2 <- A[, -c(2,4)]; Asub2    #2,4 列目以外の列の取り出し（上と同じ）
```

表2　ベクトルと行列に関するコマンド

ベクトル処理など	コマンド
最大値，最小値	max(), min()
ベクトルの長さ	length()
並べ替え	sort(x, decreasing = FALSE)
順位	order(x, decreasing = FALSE)
数列作成	seq(from, to, by=1), seq(from, to, length.out)
複製	rep(x, times=1)
行列の演算と関数	**コマンド**
行列の掛け算	A%*%B
/	
/	
行列の転置, 逆行列	t(A), solve(A)
対角成分の取り出し	diag(A)
固有値, 特異値分解	eigen(A), svd(A)
行あるいは列に関する和	rowSums(x), colSums(x)

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    5    3    1
```

行列は 2 次元の配列であるが，さらに高次元の配列を定義することもできる．また，長さが一致していない配列を扱えるリストというオブジェクトの定義の仕方もある．

4 R を用いた簡単な計算 (3) R のグラフィックス機能

R には多種多様なグラフィック機能が備わっている

表3 基礎的なグラフィックス関数 (1)

グラフィックス関数	
`plot`	汎用的な関数
`barplot`	棒グラフ
`boxplot`	箱ひげ図
`curve`	関数プロット
`hist`	ヒストグラム
`pairs`	散布図
`contour`	等高線
`qqplot`	Q-Q プロット
グラフィックス関数の代表的な引数	
`type`	プロットの形式 (`p`= 点, `l`= 線, `b`= 点と線の両方, `n`= 何も書かない)
`xlim, ylim`	座標軸の範囲
`xlab, ylab`	座標軸の名前
`cex, col, pch`	点や文字のサイズ, 色, 形
`lty, lwd`	線の種類 (デフォルトは `lty=1`= 実線，`lwd=1`)
`main`	メインタイトル

表4 基礎的なグラフィックス関数 (2)

作図した図に点，線，テキストなどを加えるコマンド	
`points`	点を加える
`abline`	直線を加える
`legend`	線や点の説明を加える
`text, mtext`	図中にテキストを加える
グラフィックス制御関数 par() における代表的な引数	
`mar`	軸ラベルのためのマージンの設定 c(下, 左, 上, 右)
`oma`	外部マージンの設定 c(下, 左, 上, 右)

4.1 データの可視化

気温の年変化を図で表してみよう．東京の気温と降水量のデータを”Data_Tokyo.csv” というファイルに保存してあるとします．R ではテキスト形式や CSV 形式で保存されたファイルを読み込むことができる．例えば，CSV 形式で保存された”test.csv” というファイルが作業フォルダーに保存されているとする．また，R の作業フォルダーがその作業フォルダーであるとき，

> オブジェクト名 <- read.csv(file=” ファイル名.csv”, header=T)

とすればデータを読み込むことができる．なお，1 行目が表題であるとき”header=T”, そうでないときには”header=F” とする．読み込まれたデータはデータフレームという R の特別な書式で保存される．

なお，データファイルがテキストファイルで保存されている時には，`read.table()`という関数を利用する．また，データを読み込んでベクトルとして保存する場合には`scan()`も便利である．一方，データの出力には

> write.csv(オブジェクト名, file=” ファイル名”)

を利用すれば CSV ファイルが自動的に生成される．

```
Data_Tokyo <- read.csv("Data/Data_Tokyo.csv", header=T) #CSV ファイルの読み込み
head(Data_Tokyo, 10) # 読み込みの確認（最初の 10 行）
```

```
   Year Month Temperature Rainfall
1  1900     1         1.5     69.2
2  1900     2         2.6     31.3
3  1900     3         5.2     69.3
4  1900     4        12.3    168.6
5  1900     5        17.5    127.3
6  1900     6        20.4     89.4
7  1900     7        23.8    122.1
8  1900     8        26.6     66.3
9  1900     9        22.6    173.9
10 1900    10        15.4    113.6
```

```
tail(Data_Tokyo, 10) # 読み込みの確認（最後の 10 行）
```

```
     Year Month Temperature Rainfall
1383 2015     3         9.4     94.0
1384 2015     4        15.6    129.0
1385 2015     5        21.2     88.0
1386 2015     6        22.8    195.5
1387 2015     7        27.2    234.5
1388 2015     8        28.3    103.5
1389 2015     9        22.9    503.5
1390 2015    10        18.1     57.0
1391 2015    11        14.5    139.5
1392 2015    12         9.0     82.5
```

```
names(Data_Tokyo)     #Data_Tokyo のオブジェクトの構成要素を確認
```

```
[1] "Year"        "Month"       "Temperature" "Rainfall"
```

```
dim(Data_Tokyo)       #Data_Tokyo の配列次元の確認 1392×4 の行列であることが分かる
```

```
[1] 1392    4
```

データは Data_Tokyo という名前のオブジェクトに保存されている．ここから気温や降水量を取り出したい時には，Data_Tokyo$Temperature'や Data_Tokyo$Rainfall' として取り出せばよい．あるいは Data_Tokyo[,3] とすれば気温を取り出せる．また 1 月の気温だけを取り出す場合には，下記のようにすればOK.

```
Data_Tokyo$Temperature[Data_Tokyo$Month==1]
```

```
 [1] 1.5 4.2 1.7 3.8 1.5 4.3 1.6 3.8 3.2 2.5 4.4 2.8 2.3 1.8 3.5 3.2 4.8 1.4 1.0
[20] 2.9 3.3 3.6 1.3 1.5 2.7 2.4 2.2 2.7 3.6 1.7 2.7 3.4 4.6 2.7 1.1 3.2 0.9 4.0
[39] 2.2 2.0 2.3 4.6 3.0 1.8 2.9 0.9 3.4 3.8 3.5 4.6 5.2 3.5 4.1 2.9 5.1 3.2 3.6
[58] 4.3 4.0 3.4 3.9 2.6 3.2 1.1 5.5 3.3 3.4 3.2 3.4 4.8 3.6 4.3 7.1 6.2 3.2 4.0
[77] 3.7 2.5 4.7 5.7 4.2 2.4 3.8 4.8 2.2 3.1 2.8 4.8 5.9 6.8
 [ reached getOption("max.print") -- omitted 26 entries ]
```

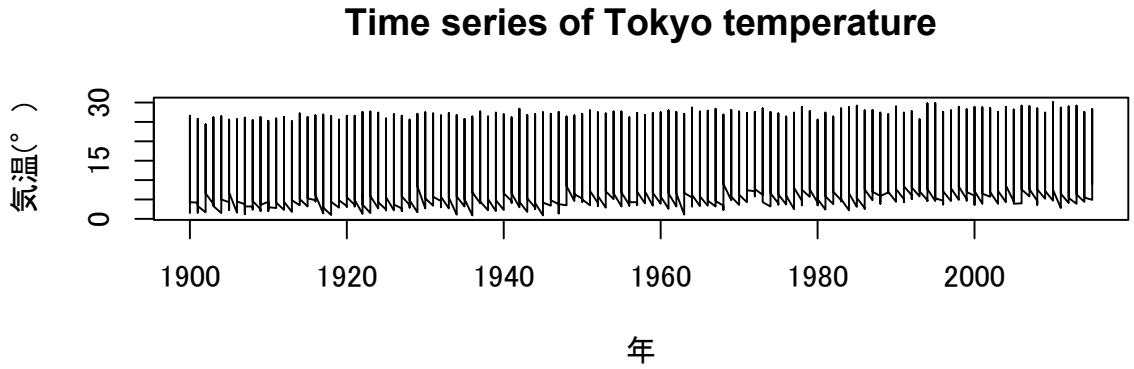
毎回ドルマークを付けて取り出すのが面倒な場合は，以下のように最初に attach という関数を利用して Data_Tokyo の名前を覚えさせてから実行してもよい（あまり乱用しない方がよいやり方ではあるが）.

```
attach(Data_Tokyo)
```

```
Temperature[Month==1]
```

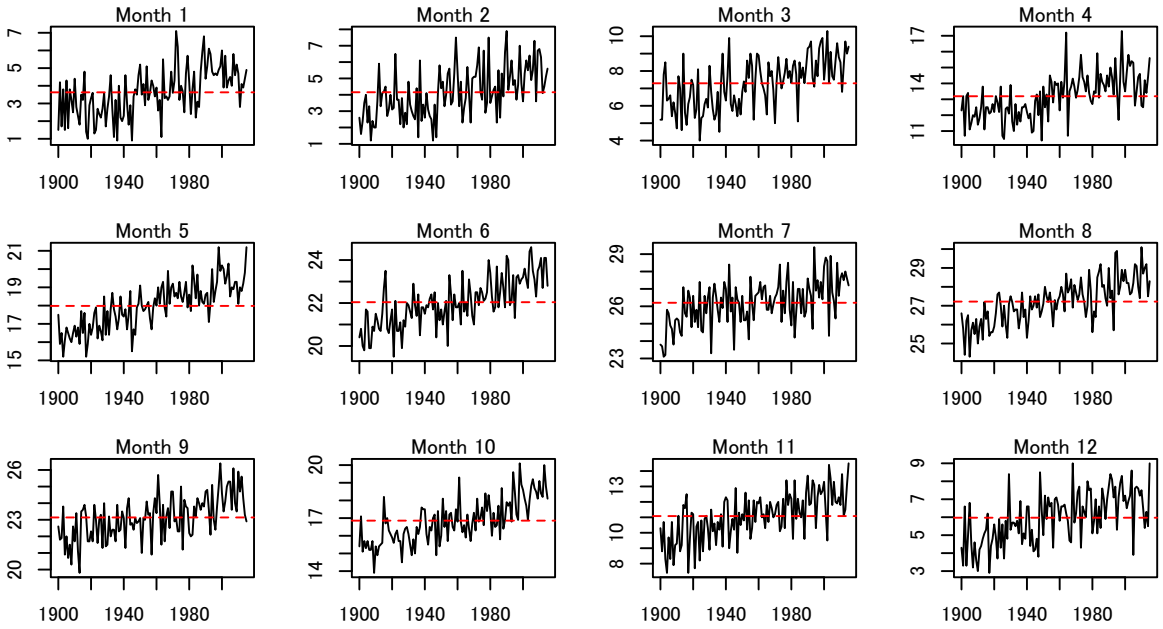
4.2 気温データの図示（標準的な方法）

```
#横軸に年，縦軸に気温，それを"line"で結ぶ
plot(Year, Temperature, type="l", main="Time series of Tokyo temperature",
     xlab="年", ylab="気温(°)")
```



月毎にプロットすると変化の様子が分かりやすい.

```
par(mfrow=c(3,4), mar=c(2,2,2,2)) #図を 3×4 で表示
MEAN <- numeric(12)  #月別の平均気温を計算して保存するためのオブジェクト
for(i in 1:12){
 MEAN[i] <- mean(Temperature[Month==i], na.rm=T) #月別の平均気温の計算
 plot(Year[Month==i], Temperature[Month==i], type="l") #月別の気温変化プロット
 abline(MEAN[i], 0, col="red", lty=2)  #切片 MEAN[i], 傾き 0 の赤い直線 (鎖線)
 mtext(paste("Month", i), side=3, cex=0.7) #月を記入
}
```



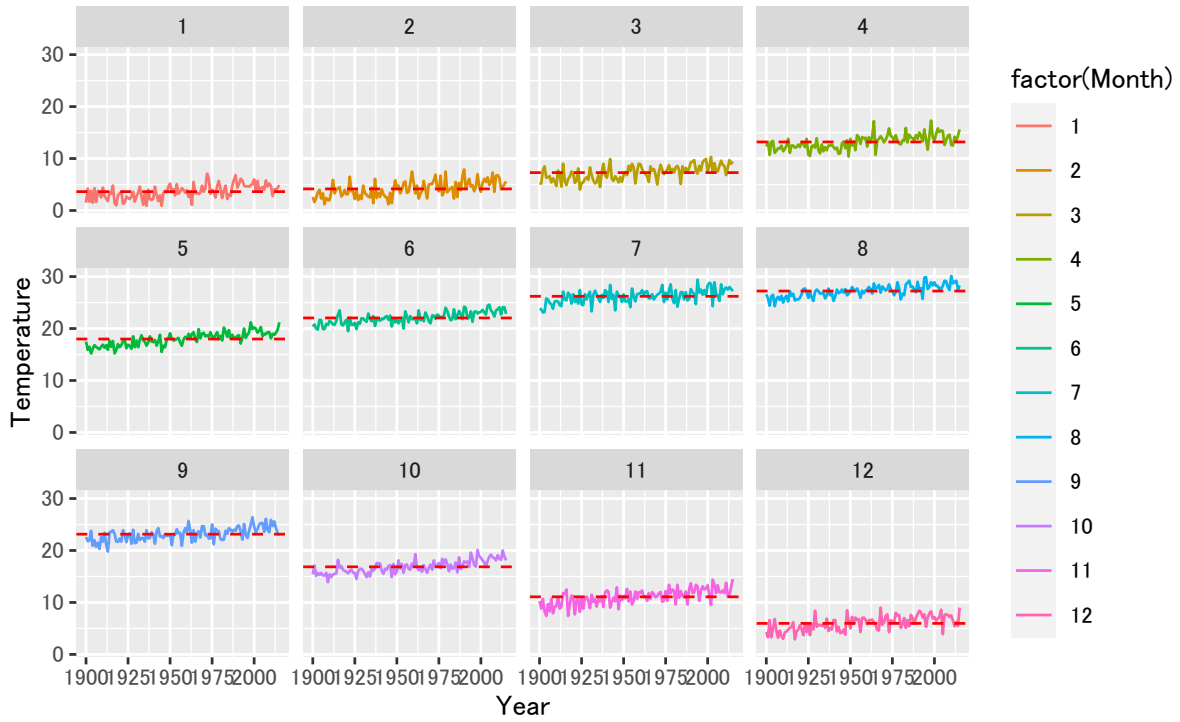
4.3 気温データの図示（ggplot で自動的に綺麗な図を書く方法）

ggplot2 というパッケージを使うと図が大変きれいに書けます (ちょっと文法を覚えるのが面倒ですが). 始めて使う場合にはまずパッケージをインストールします.

```
#install.packages("ggplot2")   #インストール（1 回だけ実行すれば OK）
library(ggplot2)              #パッケージの読み込み（R を使う毎に毎回実行）
```

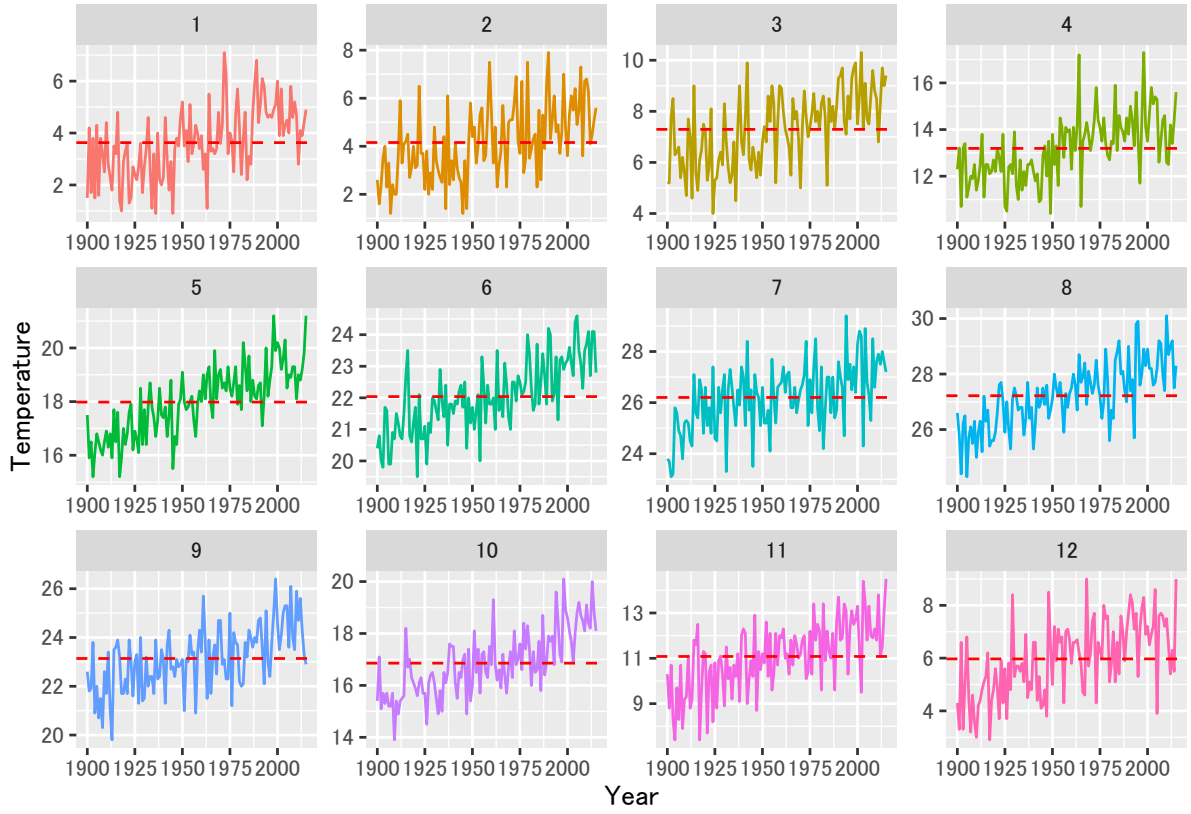
先ほどの月別気温の変化をカラフルに書いてみます.

```
df <- data.frame(Month,MEAN)
tmp <- ggplot(Data_Tokyo, aes(Year, Temperature, col=factor(Month))) +
   geom_line() + facet_wrap(.~Month, ncol=4) +
   geom_hline(data=df, aes(yintercept=MEAN), linetype="dashed", color="red")
tmp
```



縦軸のスケールを変えた方が年変化の様子が分かりやすく，また凡例もこの場合は不要なので削除.

```
tmp <- ggplot(Data_Tokyo, aes(Year, Temperature,col=factor(Month))) +
   geom_line(show.legend=FALSE) + facet_wrap(.~Month, ncol=4, scales="free") +
   geom_hline(data=df, aes(yintercept=MEAN), linetype="dashed", color="red")
tmp
```



5 個体群動態モデル (参考までに，また今後扱います)

5.1 Pella-Tomlinson 余剰生産モデル (確率変動なし)

5.1.1 モデルの定義

$$P_{t+1} = P_t + rP_t\left\{1 - \left(\frac{P_t}{K}\right)^z\right\} - C_t$$

5.1.2 初歩的に漸化式を利用して個体群動態を計算

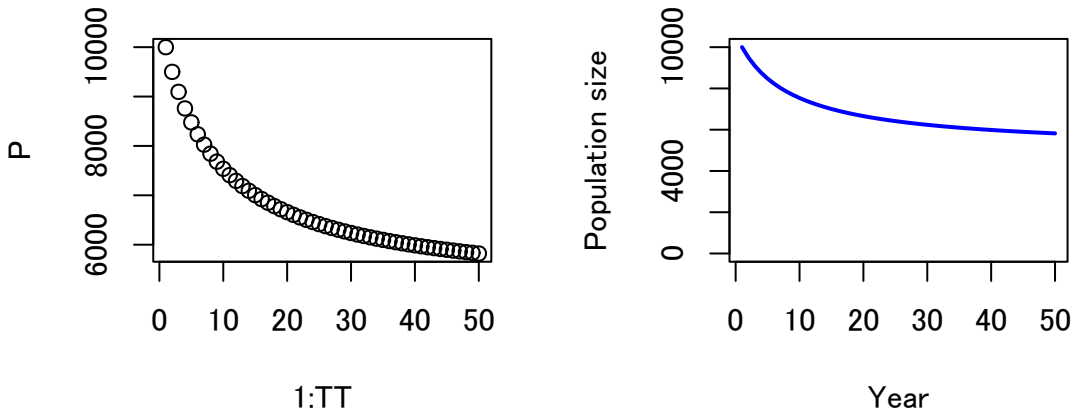
```
r <- 0.2
K <- 10000
z <- 1
TT <- 50

Catch <- rep(500, TT)

TT1 <- TT-1
P <- numeric(TT)
P[1] <- K
for(t in 1:TT1)
{
```

```
 tmp <- P[t] + r*P[t]*(1-(P[t]/K)^z) - Catch[t]
 P[t+1] <- max(tmp, 0.001)
}

print(P, digits=0)
```

```
 [1]10000 9500 9095 8760 8477 8235 8026 7843 7681 7537 7409 7293 7187 7092 7004
[16] 6924 6850 6781 6718 6659 6604 6552 6504 6459 6416 6376 6338 6303 6269 6236
[31] 6206 6177 6149 6123 6097 6073 6050 6028 6007 5987 5967 5949 5931 5913 5897
[46] 5881 5865 5850 5836 5822
```

5.1.3 計算結果の図示

```
par(mfrow=c(1,2))
plot(1:TT, P)
plot(1:TT, P, type="l", lwd=2, col="blue", xlab="Year", ylab="Population size", ylim=c(0,K))
```



5.1.4 もう少し進んだ方法：個体群動態計算用関数 PDM.PT を作成

```
#Pella-Tomlinson Production model (without stochastic process error)
PDM.PT<-function(r, K, z, P1, TT, Catch)
{
 TT1 <- TT-1
 P <- numeric(TT)
 P[1] <- P1
 for(t in 1:TT1)
 {
  tmp <- P[t] + r*P[t]*(1-(P[t]/K)^z) - Catch[t]
  P[t+1] <- max(tmp, 0.001)
 }
 return(P)
```

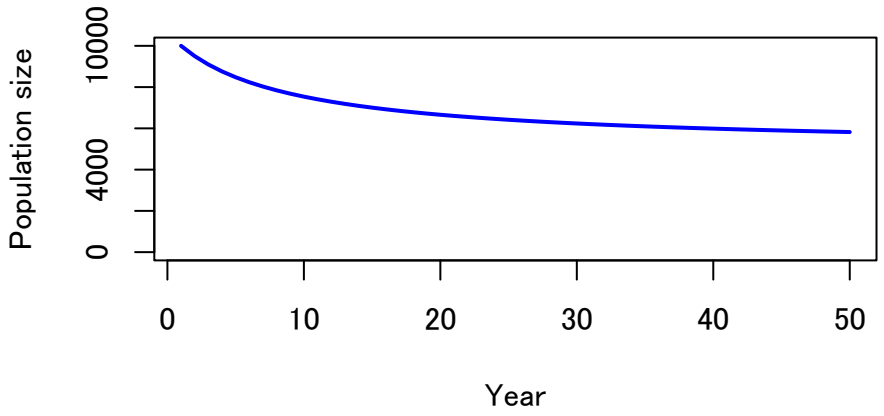
```
}
```

5.1.5 新しく作った個体群動態計算用関数 PDM.PT を実行

```
Res <- PDM.PT(r=0.2, K=10^4, z=1, P1=10^4, TT=50, Catch=rep(500,TT))
print(Res, digits=0)
```

```
 [1]10000 9500 9095 8760 8477 8235 8026 7843 7681 7537 7409 7293 7187 7092 7004
[16] 6924 6850 6781 6718 6659 6604 6552 6504 6459 6416 6376 6338 6303 6269 6236
[31] 6206 6177 6149 6123 6097 6073 6050 6028 6007 5987 5967 5949 5931 5913 5897
[46] 5881 5865 5850 5836 5822
```

```
plot(1:TT, Res, type="l", lwd=2, col="blue", xlab="Year", ylab="Population size", ylim=c(0,K))
```



5.2 Pella-Tomlinson 余剰生産モデル (確率変動あり)

5.2.1 モデルの定義

$$P_{t+1} = \left[P_t + rP_t \left\{ 1 - \left(\frac{P_t}{K} \right)^z \right\} - C_t \right] * \exp(\varepsilon_t), \quad \varepsilon_t \sim N(0, \sigma^2)$$

5.2.2 さっき作った個体群動態計算用関数 PDM.PT を書き換え (上書き更新)

```
#Pella-Tomlinson Production model (with stochastic process error)
PDM.PT<-function(r, K, z, P1, TT, Catch, sigma=0)
{
 TT1 <- TT-1
 P <- numeric(TT)
 Epsilon <- rnorm(n=TT, mean=0, sd=sigma)
```

```
 P[1] <- P1
 for(t in 1:TT1)
 {
  tmp <- P[t] + r*P[t]*(1-(P[t]/K)^z) - Catch[t]
  tmp <- tmp*exp(Epsilon[t])
  P[t+1] <- max(tmp, 0.001)
 }
 return(P)
}
```

5.2.3 再定義した PDM.PT を実行

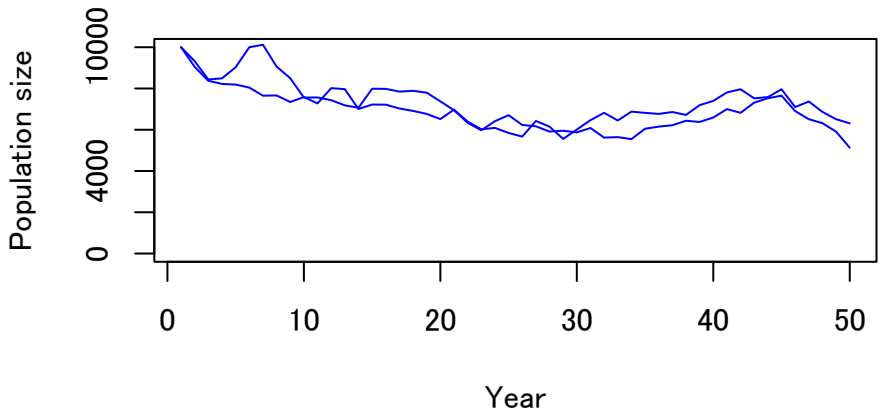
```
K <- 10^4
TT <- 50

Res <- PDM.PT(r=0.2, K=K, z=1, P1=K, TT=50, Catch=rep(500,TT), sigma=0.05)
print(Res, digits=0)
```

```
 [1]10000 9049 8371 8221 8189 8046 7651 7663 7344 7585 7277 8016 7967 7005 7221
[16] 7217 7030 6913 6767 6517 6975 6394 6022 6090 5845 5668 6425 6149 5556 6010
[31] 6465 6820 6449 6878 6818 6768 6859 6718 7193 7397 7809 7963 7520 7584 7960
[46] 7100 7369 6863 6511 6312
```

```
plot(1:TT, Res, type="l", col="blue", xlab="Year", ylab="Population size", ylim=c(0,K))

Res <- PDM.PT(r=0.2, K=K, z=1, P1=K, TT=50, Catch=rep(500,TT), sigma=0.05)
print(Res, digits=0)
```

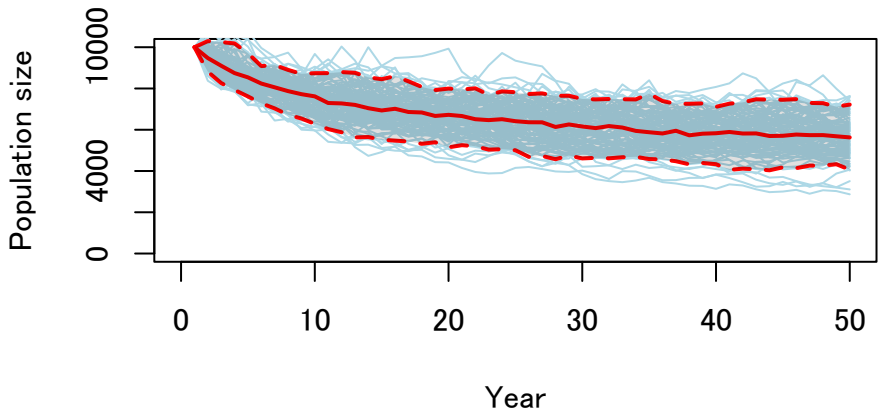
```
 [1]1.000000e+049.322242e+038.435280e+038.491935e+039.029453e+039.998606e+031.012019e+049.062235e+03
[16]7.981144e+037.849277e+037.887517e+037.797486e+037.374925e+036.947505e+036.337877e+035.974759e+03
[31]6.084305e+035.617948e+035.641970e+035.545169e+036.049011e+036.150528e+036.218524e+036.436582e+03
[46]6.906242e+036.511256e+036.318266e+035.905327e+035.129792e+03
```

```
points(1:TT, Res, type="l", col="blue")
```



5.2.4 PDM.PT を繰り返し実行して結果を図示

```
Nsim <- 100
Pmat <- array(NA, c(Nsim, TT))
plot(0, type="n", xlab="Year", ylab="Population size", xlim=c(0,TT),  ylim=c(0,K))

for(i in 1:Nsim)
{
 Pmat[i,] <- PDM.PT(0.2, 10^4, 1, 10^4, 50, rep(500,TT), 0.05)
 points(1:TT, Pmat[i,], type="l", col="lightblue")
}
P.med <- apply(Pmat, 2, median);
P.L5per <- apply(Pmat, 2, quantile, 0.05)
P.U5per <- apply(Pmat, 2, quantile, 0.95)
points(1:TT, P.med, type="l", lwd=2, col="red")
points(1:TT, P.L5per, type="l", lty=2, lwd=2, col="red")
points(1:TT, P.U5per, type="l", lty=2, lwd=2, col="red")

polygon(
  c(1:TT, TT:1), c(P.L5per, rev(P.U5per)),
col = "#00000020", border = NA)
```

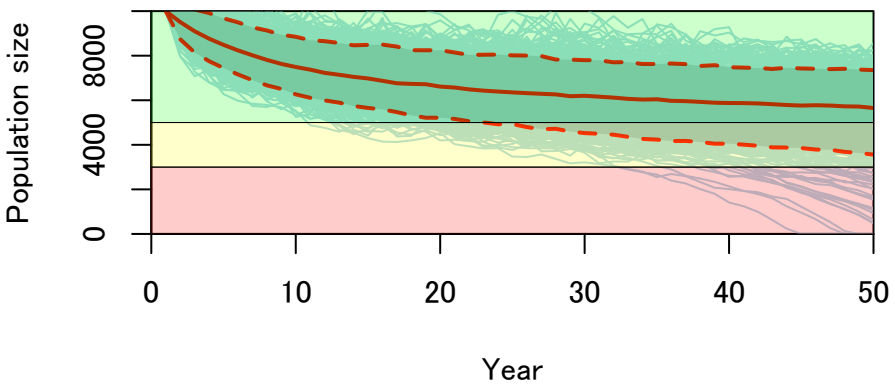


5.2.5 上記を関数化

```
PDM.PT.sim <- function(r, K, z, P1, TT, Catch, sigma=0, Nsim)
{
 Pmat <- array(NA, c(Nsim, TT))
 plot(0, type="n", xlab="Year", ylab="Population size",
      xlim=c(0,TT), ylim=c(0,K), xaxs="i", yaxs="i")

 for(i in 1:Nsim)
 {
  Pmat[i,] <- PDM.PT(0.2, 10^4, 1, 10^4, 50, rep(500,TT), 0.05)
  points(1:TT, Pmat[i,], type="l", col="lightblue")
 }
 P.med <- apply(Pmat, 2, median);
 P.L5per <- apply(Pmat, 2, quantile, 0.05)
 P.U5per <- apply(Pmat, 2, quantile, 0.95)
 points(1:TT, P.med, type="l", lwd=2, col="red")
 points(1:TT, P.L5per, type="l", lty=2, lwd=2, col="red")
 points(1:TT, P.U5per, type="l", lty=2, lwd=2, col="red")

 polygon( c(1:TT, TT:1), c(P.L5per, rev(P.U5per)), col = "#00000020", border = NA)

 rect(xleft=-1, ybottom=0, xright=TT, ytop=0.3*K, lwd=0, col=rgb(1,0,0,alpha=0.2))
 rect(xleft=-1, ybottom=0.3*K, xright=TT, ytop=0.5*K, lwd=0, col=rgb(1,1,0,alpha=0.2))
 rect(xleft=-1, ybottom=0.5*K, xright=TT, ytop=K, lwd=0, col=rgb(0,1,0,alpha=0.2))

 return(Pmat)
}
```

5.2.6 実行！

```
Res <- PDM.PT.sim(0.2, 10^4, 1, 10^4, 50, rep(500,TT), 0.05, Nsim=1000)
```




```
nf <- layout(matrix(c(1,2,0,0),2,2,byrow=TRUE), c(3,1), c(4,0), FALSE)
par(mar=c(4,4,3,1))
Res <- PDM.PT.sim(0.2, 10^4, 1, 10^4, 50, rep(500,TT), 0.05, Nsim=1000)
par(mar=c(3,0,2,1))
Hist.final <- hist(Res[,TT], breaks=seq(0,K,500), plot=FALSE)
top <- max(Hist.final$counts)
barplot(Hist.final$counts, axes=FALSE, xlim=c(0, top), space=0, horiz=TRUE)
```