「学習者コーパス論」（杉浦正利）：R でテキスト処理（2011 年度版）

11 月 24 日

R のダウンロード

http://cran.md.tsukuba.ac.jp/

変数
　　数字と文字
　　文字はダブルクオートに入れる

<- で入れる

配列＝入れ物が複数ならんでいる変数

<- c(　,　,　,　) で入れる

変数や配列に何があるか調べる
　　ls()

要素がいくつ入っているか調べる
　　length(配列名）

作業スペースを USB メモリーに保存しておくと、作業内容をとっておける。

●作業ディレクトリーがどこに名ているか確認

　　getwd()

●ファイルとして保存されているデータの読み込み

scan(file="ファイルの場所と名前", what="char")

　　文字データなので、what="char"を指定

ウインドを開いて、GUI で選ぶ

scan(choose.files(), what="char")

セパレーターの話
　　データの単位を何で区切るか
　　デフォルトは　スペース

　　行単位で読み込むには、セパレーターを指定する
　　sep="¥n"

読み込んだファイルを R の中にデータとして保存する

配列名「ns.lines」という名前にすることにする

```
ns.lines <- scan(choose.files(), what="char", sep ="¥n")
```

●中身を見るにはどうするか？

先頭部分を見る

　　head(変数名, 要素数)

末尾部分を見る

　　tail(変数名, 要素数)

　　★下からデータを取る＝上のデータを削除する
　　　マイナスの値を指定することで、上からその数だけ削除したものが出力される

　　　tail (変数名, -1)
　　　　最初の一行の削除

先頭１４行を見る

末尾６７行を見る

１５行目から１７行目だけを見るには？

上から１７行目出力した後で、下から３行とれば、結果的に１５、１６、１７行目が出る。

```
head(ns.liles, 17)
```

```
tail(   , 3)

tail(head(ns.lines, 17), 3)
```

●特定の行だけ取り出す

```
ns.lines[79]

ns.lines[c(77, 78, 79)]

ns.lines[77:79]
```

●文字列の検索

```
grep("検索文字列", 変数名)

grep("NS001", ns.lines)
```

要素番号（＝行番号）が表示される。

●該当する行のデータを取り出す

```
ns.lines[grep("NS001", ns.lines)]
```

●要素番号でなく、中身そのものを出力するオプション

```
value=T

grep("NS001", ns.lines, value=T)
```

●一つ目の要素は該当しないもの
tail の応用で削除

●母語話者データのデータ部分のみ抽出

```
ns.data <- tail(grep("NS001", ns.lines, value=T), -1)
```

単語リストを作る

ns.data

●各行単位で入っているデータを、単語単位にバラバラにする

●文字列をバラバラにする命令

strsplit(変数名, "セパレータ")

strsplit(ns.data, " ")

ns.data をスペースで切ってバラバラにする

バラバラにした結果は「リスト」に入る（２次元データ）

```
[[1]]
     [1] [2] ...
[[2]]
     [1] [2] ...
[[3]]
     [1] [2] ...
```

ns.data.list <- strsplit(ns.data, " ")

●リストの要素をバラバラにする

unlist(変数名)

unlist(ns.data.list)

unlist(strsplit(ns.data, " ")) でも同じ

●バラバラになった単語を並べ替える

sort(変数名)

sort(unlist(ns.data.list))

●並べ替えた単語の一覧の重複をまとめる（異なり語）

unique(変数名)

```
unique(sort(unlist(ns.data.list)))
```

■細かい問題点

●話者記号「*NS001:¥t」が入っている　＝＞　削除

●削除するとは、、、何もなしで置き換える

●置換コマンド

gsub(元の文字列, 新しい文字列, 対象データ)

不要なのは「*NS001:¥t」

で、やってみよう。

うまくいかない。

特殊文字「*」は、「エスケープ」する必要がある
＝＞　¥¥　を前につけてエスケープ

```
gsub("¥¥*NS001:¥t", " ", ns.data)
```

この結果を、ns.data2 という変数で保存

この細かい問題を解決したうえで、再度、単語リスト（異なり語）を作成してみよう。

●出来上がったリストの出力（ファイルとして保存）

```
write(変数名, file="ファイル名")
```

```
write(ns.data.words, file="NS001-wordlist.txt")
```

復習として、別のファイルでやってみよう

総語数（述べ語数）（token）と
異なり語数（type）を出してみよう。

そうなると、TTR もだせるか。

```
Type/Token
```

となると、Guiraud Index も出せるな。

```
Type/sqrt(Token)
```

■オマケ

●行数を数える

```
nrow()
```

●ちなみに列数を数える

```
ncol()
```

●変数の名前を変える

```
新しい名前 <- 古い名前
rm(古い名前)
```

１２月１日

gsub("置き換え前の文字列", "置き換え後の文字列", 変数)

削除は何もなしで置き換える。

gsub("置き換え前の文字列", "", 変数)

授業で扱わなかった「細かいこと」

１）記号類の処理（＝英数字以外を削除）

正規表現で表記

阿部君の使った方法　[^a-zA-Z0-9]

gsub("[^a-zA-Z0-9]", "", 変数)

もう一つの方法　¥W

gsub("¥¥W", "", 変数)

R での正規表現は以下を参照のこと
http://www.okada.jp.org/RWiki/?R%20%A4%CB%A4%AA%A4%B1%A4%EB%C0%B5%B5%AC%C9%BD%B8%BD#content_1_3

２）大文字と小文字の取り扱い（＝全部小文字にそろえる）

阿部君の使った方法　tolower

tolower(変数)

★コマンド（関数）の使い方は、メニューの「ヘルプ」から「R の関数」を選ぶ

★コツをまとめたページも参照のこと
http://sugiura-ken.org/wiki/wiki.cgi/exp?page=R

★同じような処理を何回もする。＝＞　プログラミングしてひとつの「命令」としてまとめる（後でやります）

■やってきた処理の復習

１）何をしたいのか？

２）単語（異なり語）の一覧表
３）使われている単語（述べ語）をリストにする
４）文をバラバラの単語にする
　　スペースをセパレーターに
　　記号類は削除
　　全部小文字に
５）本文だけ抜き出す
　　話者記号の削除
６）発話の行だけ抜き出す
７）行単位でファイルからデータを読み込む

異なり語数と、述べ語数がわかれば、いろいろわかる。
１）テキストの長さ（含まれる単語数＝述べ語数）
２）ＴＴＲ
３）Guiraud index

さて、今日は、単語の頻度一覧表
---

table(述べ語数の変数)

これだけ！

```
NS001word.list <- table(NS001word.token)
```

ファイルに保存しましょう。

```
write(NS001word.list, "保存ファイル名")
```

（どこに保存されるかは、カレントディレクトリーを確認： getwd())

見てみる。あ、だめだ、、、

```
write.table(NS001word.list, "保存ファイル名")
```

これで、表として保存される。

エクセルで開いてみよう。

区切りは、デフォルトは　スペース

区切りをタブに変えるには、オプションを付けて保存。

```
write.table(NS001word.list, "保存ファイル名", sep="¥t")
```

■同じような処理をまとめて一度にする

■まずは、データを読み込んで、本文のテキスト部分だけを変数に入れるには

```
myData <- function(){
    lines.tmp <- scan(choose.files(), what="char", sep="¥n")
        #ファイルを選択。
    data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
        #*で始まる行のみ。
    body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
        #行頭のタグを削除。
    body.tmp <- body.tmp[body.tmp != ""] #空行を削除。
}
```

あとは、

```
jpn.body <- myData()
```

を実行。開くファイルを聞かれるので、指定するだけ。

■では、TTR をひとまとめのコマンドに

```
myTTR <- function(a){
    jpn.body.lower <- tolower(a)
    jpn.words <- unlist(strsplit(jpn.body.lower, "¥¥W"))
    length(unique(jpn.words))/length(jpn.words)
}
```

あとは、

```
myTTR(jpn.body)
```

を実行。

■では、同様に Guiraid index

```
myGI <- function(a){
    jpn.body.lower <- tolower(a)
    jpn.words <- unlist(strsplit(jpn.body.lower, "¥¥W"))
    length(unique(jpn.words))/sqrt(length(jpn.words))
}
```

次に、平均単語長を出してみよう

平均単語長とは、単語を構成する文字数の平均（Average Word Length)

総文字数　÷　総単語数　ででるはず。

総単語数は、述べ語数なので、わかってる。

じゃ、総文字数はどう出すか？

★文字数を数えるコマンド（関数）：　nchar(変数)

```
nchar(jpn.body)
```

これだと、各文に含まれる単語の総文字数がでる。

じゃ、単語を全部スペースなしでつなげたものが入った変数をつくればよい。

★要素をつなげるコマンド：　paste(変数, collapse="")

```
paste(jpn.words, collapse=""))
```

こうしておいて、総文字数を調べればよい。

```
myAWL <- function(a){
    jpn.words <- unlist(strsplit(a, "¥¥W"))
    nchar(paste(jpn.words, collapse=""))/length(jpn.words)
}
```

これであとは、

```
myAWL(jpn.body)
```

じゃ、もひとつ、平均文長（Average Sentence Length）

一文に含まれる単語数の平均

総単語数　÷　総文数

あ、これは簡単だ。自分で myASL を作ってみよう。

```
--------------------
```
ここまできたら、ファイルを指定したらこれらを全部まとめて処理した結果を出すプログラムも書けますね。

```
述べ語数　異なり語数　Guiraud Index　平均単語長　平均文長
--------------------
```

１２月８日

■平均文長（一文の中に平均単語がいくつあるか）

（Ａ）
（１）文ごとに単語がいくつ入っているか調べる。
（２）単語の数の平均を出す。

（Ｂ）
（１）ファイル全体で文がいくつあるか調べる。
（２）ファイル全体で総単語数を調べる。
（３）単語数を文数で割ると平均が出る。

■データを読み込んで、本文のテキスト部分だけを変数に入れるには

```
myData <- function(){
    lines.tmp <- scan(choose.files(), what="char", sep="¥n")
        #ファイルを選択。
    data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
        #*で始まる行のみ。
    body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
        #行頭のタグを削除。
    body.tmp <- body.tmp[body.tmp != ""] #空行を削除。
}
```

あとは、

```
jpn.body <- myData()
```

これで、データのみが jpn.body に入る。

■jpn.body に本文データが入っているとして、
▲（Ｂ）の方法で、平均文長を求めるには、

```
myASL <- function(a){
     words.tmp = unlist(strsplit(a,"¥¥W"))
     words.tmp = words.tmp[words.tmp !=""]
     length(words.tmp)/length(a)
     }
```

これで、独自の関数 myASL を作っておいて、

```
myASL(jpn.body)
```

を実行すればよい。

★阿部君の発案。いちいち jpn.body を保存しなくても、「入れ子」にしてしまえばよい。

```
> myASL(myData())
```

▲（A）方法で、平均文長を求めるには

各行（＝文）ごとに、単語数を出す。

ということは、各行を配列とみなして、その要素数をだせばよい。

★まず、記号類を削除しておく

```
myNoKigou <- function(a){
     b <- gsub(" ", "ZSPACEZ", a)   # スペースは温存したい
     c <- gsub("¥¥W", "", b)        # 英数文字以外は削除
     d <- gsub("ZSPACEZ", " ", c)   # スペースを戻す
     gsub("  *", " ", d)           # 複数のスペースを一つに
     }
```

＞ `strsplit(myNoKigou(jpn.body), "¥¥W")`

これで二次元の配列（リスト）ができる。

二次元配列の一行目は `[[1]]`

```
strsplit(myNoKigou(jpn.body), "¥¥W")[[1]]
```

わかりやすくするために、いったんリストに `x` という名前を付けて保存。

```
x <- strsplit(myNoKigou(jpn.body), "¥¥W")
```

なので、一行目は、

`x[[1]]` で出る。

一行目の要素数（単語数）は、

```
length(x[[1]])
```

これで、各行の単語数が出る。

あとは、これをすべての行について、「繰り返して行う」。

全部で何行あるかというと、

```
length(x)
```

▲繰り返して実行（制御：プログラムの流れをコントロールする）

```
for (条件) {
     すること
     すること
     }
```

★「条件」部分の書き方

変数 in その範囲

例： i in 1:19

1から19まで、順に i に入れる

例： i in 1:length(x)

1から、xの数（文の数）まで、順に i に入れる

```
for (i in 1:length(x)){
     print(length(x[[i]]))    # print()は「出力」する命令
     }
```

これを実行すると、各行の単語数が一覧表示

「出力」するのでなく、データとして y に保存する

```
for (i in 1:length(x)){
     y <- (length(x[[i]]))
     }
```

あ、これはだめ。最後のしか保存されてない。

★ xのi番目のをyのi番目に保存しないといけない。

```
for (i in 1:length(x)){
     y[i] <- (length(x[[i]]))
     }
```

これで、y に入った。

★平均を出す関数は mean()

```
mean(y)
```

■ファイルを指定すると、Type と Token の両方を一度に出力する関数は？

```
myTT <- function(a){
     lines.tmp <- scan(choose.files(), what="char", sep="¥n")
     data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
     body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
     body.tmp <- body.tmp[body.tmp != ""] #空行を削除。

     body.tmp.lower <- tolower(body.tmp)                # 小文字に
     words.tmp <- unlist(strsplit(body.tmp.lower, "¥¥W")) # 単語に
     words.tmp = words.tmp[words.tmp !=""]              # あるのだけに
     c(length(unique(words.tmp)), length(words.tmp))     # 二つの要
素を配列に
```

```
}
```

で、あとは、以下を実行：

```
myTT()
```

■複数のファイルに対して自動的に実行するには、、、、

まず、ファイル名を指定して実行するには、

`scan(choose.files(), what="char", sep="¥n")` の箇所で、
具体的にファイル名を指定する。
★話を分かりやすくするために、
　作業ディレクトリーを、デーのあるディレクトリーに変更
　＝＞メニューの「ファイル」から、「ディレクトリの変更」
　＝＞`getwd()` で確認。

　そこにあるファイル一覧を表示
　＞ `dir()`

▲ `JPN001.txt` を例に
```
scan("JPN001.txt" , what="char", sep="¥n")
```

これで、`JPN001.txt` 専用のプログラムを作ってみる。

```
myTTJPN001 <- function(){
     lines.tmp <- scan("JPN001.txt" , what="char", sep="¥n")
     data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
     body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
     body.tmp <- body.tmp[body.tmp != ""] #空行を削除。
     body.tmp.lower <- tolower(body.tmp)                # 小文字に
     words.tmp <- unlist(strsplit(body.tmp.lower, "¥¥W")) # 単語に
     words.tmp = words.tmp[words.tmp !=""]              # あるのだけに
     c(length(unique(words.tmp)), length(words.tmp))     # 二つの要
素を配列に
}
```

後は実行：

```
＞ myTTJPN001()
```

▲ファイル名の一覧表示
```
＞ list.files
```

▲繰り返して実行（制御：プログラムの流れをコントロールする）

```
for (条件) {
     すること
     すること
     }
```

▲ファイル一覧のファイルを順に処理すればよい

```
myTT <- function() {
     output.file = choose.files()
     type <- 0
     token <- 0
     files <- list.files()
     for (i in files) {
       lines.tmp <- scan(i , what="char", sep="¥n")
       data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
       body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
       body.tmp <- body.tmp[body.tmp != ""]
       body.tmp.lower <- tolower(body.tmp)
       words.tmp <- unlist(strsplit(body.tmp.lower, "¥¥W"))
       words.tmp <- words.tmp[words.tmp !=""]
       type <- length(unique(words.tmp))
       token <- length(words.tmp)
       cat(type, token, "¥n", file=output.file, append=T)
     }
}
```

ファイルに保存された。

```
tat <- read.table(choose.files())
```

中身を見てみる

```
tat
```

見出しがない。

```
names(tat)
```

見出しを付けるには、

```
names(tat) <- c("Type", "Token")
```

基本統計量を見る

```
summary(tat)
```

```
      Type          Token
 Min.   : 57.0   Min.   :106.0   （最小値）
 1st Qu.:110.0   1st Qu.:243.2   （四分位点の 1/4 値）
 Median :135.0   Median :324.0   （中央値）
 Mean   :142.4   Mean   :339.6   （平均）
 3rd Qu.:170.0   3rd Qu.:420.8   （四分位点の 3/4 値）
 Max.   :302.0   Max.   :898.0   （最大値）
```

```
tat$Type
```

```
tat$Token

boxplot(tat)

bar(tat)

plot(tat)
plot(tat$Token)
plot(sort(tat$Token))
plot(tat$Type)
plot(sort(tat$Type))

hist(tat$Token)
hist(tat$Type)
```

１２月１５日

■母語話者のデータを処理。

１）作業ディレクトリーを、データのあるディレクトリーに変更
　メニューの「ファイル」から「ディレクトリの変更」

２）「カレントディレクトリ」の確認
```
getwd()
```

３）myTT の実行
```
--------------------
myTT <- function() {
     output.file = choose.files()
     type <- 0
     token <- 0
     files <- list.files()
     for (i in files) {
       lines.tmp <- scan(i , what="char", sep="¥n")
       data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
       body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
       body.tmp <- body.tmp[body.tmp != ""]
       body.tmp.lower <- tolower(body.tmp)
       words.tmp <- unlist(strsplit(body.tmp.lower, "¥¥W"))
       words.tmp <- words.tmp[words.tmp !=""]
       type <- length(unique(words.tmp))
       token <- length(words.tmp)
       cat(type, token, "¥n", file=output.file, append=T)
     }
}
--------------------

myTT()
```

　保存ファイル名は、例えば「 tatns.txt 」

４）保存ファイルから R に読み込む
```
tatns <- read.table(choose.files())
```

５）見出しを付ける
```
names(tatns) <- c("Type", "Token")
```

６）基本統計量を見る
```
summary(tatns)
--------------------
     Type          Token
 Min.   :196.0   Min.   : 386.0
 1st Qu.:247.8   1st Qu.: 521.0
 Median :265.0   Median : 557.0
 Mean   :272.0   Mean   : 595.1
 3rd Qu.:287.2   3rd Qu.: 612.0
 Max.   :493.0   Max.   :1555.0
--------------------
```

７）ヒストグラム

```
hist(tatns$Token)

hist(tat$Token)
```

重ねて表示させるには、オプション　add=T

```
hist(tatns$Token)

hist(tat$Token, add=T)
```

重なって見にくい。色を付けよう。

描き直し。　前のを消すには　plot.new()

```
hist(tat$Token, col="blue")
hist(tatns$Token, add=T, col="red")
```

▲hist()のオプション
　棒の本数を設定する breaks=本数
　ｘ軸の範囲を指定する xlim=c(0,1500)

●箱ひげ図

```
boxplot(tat$Token)
boxplot(tatns$Token, add=T)

boxplot(tat$Token, tatns$Token)
```

★宿題だったこと：学習者と母語話者で、Type の比較、Token の比較

＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝■新しいこと■＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

■すべてのファイルに対して TTR を出してみる。

```
tat$Type

tat$Type[1]

tat$Token[1]

tat$Type[1]/tat$Token[1]
```

行列[行番号,列番号]

tat[1,]　1行目の要素
tat[,1]　1列目の要素（＝Type）

★三列目にTTRの値を入れたい

```
tat[1,3] <- tat[1,1]/tat[1,2]
```

tat で見てみよう

★じゃ、これを順番にやっていけばよい。

順番に繰り返す　for　を使う。

行の数は　nrow()

```
for (i in 1:nrow(tat)){
     tat[i,3] <- tat[i,1]/tat[i,2]
}
```

■じゃ、すべてのファイルに対してGIを出してみる。

```
for (i in 1:nrow(tat)){
     tat[i,4] <- tat[i,1]/sqrt(tat[i,2])
}
```

```
   names(tmp) <- c("Type", "Token", "TTR", "GI")
```

★「TTRはテキストの長さに影響を受ける」といわれている。

```
 plot(tmp$Token, tmp$TTR)
 grid()

lines(lowess(tmp$TTR~tmp$Token))
```

lines() で線を引く
lowess()で平滑化をする（ｙ軸～ｘ軸）

▲相関があるか
（１）ピアソンの相関係数（パラメトリック）
（２）スピアマンの相関係数（ノンパラメトリック）
（３）ケンドールの相関係数（ノンパラメトリック）

★パラ？

http://aoki2.si.gunma-u.ac.jp/lecture/Kentei/nonpara.html

＝＞　正規分布をしているかを検定してみればよい。

★正規分布の検定
shapiro.test() で Shapiro-Wilk 検定
　（p 値が .05 より大きいと、正規分布しているとみなせる）

相関係数を出して、かつ有意かどうかも確かめる
http://oku.edu.mie-u.ac.jp/~okumura/stat/correlation.php

```
cor.test(x, y, method="spearman")

cor.test(tmp$Token, tmp$TTR, method="spearman")
```

♪♪　四列目に Guiraud Index を付け足してみよう！！（宿題）

　（Guiraud Index はテキストの長さの影響を受けないだろうか？）

＝＝＝＝＝＝＝＝＝＝＝＝■語彙頻度順上位語リスト＝＝＝＝＝＝＝＝＝＝＝＝

（myTT の応用）

■上位１０位まで

```
--------------------
myTop10 <- function(a){
    lines.tmp <- scan(choose.files(), what="char", sep="¥n")
    data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
    body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
    body.tmp <- body.tmp[body.tmp != ""] #空行を削除。

    body.tmp.lower <- tolower(body.tmp)                # 小文字に
    words.tmp <- unlist(strsplit(body.tmp.lower, "¥¥W")) # 単語に
    words.tmp = words.tmp[words.tmp !=""]              # あるのだけに

    #c(length(unique(words.tmp)), length(words.tmp))      # 二つの要素
を配列に

    head(sort(table(words.tmp), decreasing=T), 10)       # ★語彙リス
ト　降順並べ替え　上位１０

}
--------------------
```

■上位何位までも決められるように　（関数の目的語を二つにする）

```
--------------------
myTop <- function(a,b){
    lines.tmp <- scan(choose.files(), what="char", sep="¥n")
    data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
    body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
    body.tmp <- body.tmp[body.tmp != ""] #空行を削除。

    body.tmp.lower <- tolower(body.tmp)                # 小文字に
    words.tmp <- unlist(strsplit(body.tmp.lower, "¥¥W")) # 単語に
    words.tmp = words.tmp[words.tmp !=""]              # あるのだけに

    head(sort(table(words.tmp), decreasing=T), b)       # ★順位を b
で指定

}
--------------------

myTop(,20)
```

＝＝＝＝＝＝＝＝＝＝＝＝■連語表現の抽出＝＝＝＝＝＝＝＝＝＝＝＝

（概念をホワイトボードで説明）

```
--------------------
my2gram <- function(a){
    lines.tmp <- scan(choose.files(), what="char", sep="¥n")
    data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
    body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
    body.tmp <- body.tmp[body.tmp != ""] #空行を削除。

     word.tmp <- unlist(strsplit(body.tmp, "¥¥W"))   # 英数字以外で
切って「単語」バラバラに

     for (i in 1:length(word.tmp)-1) {
          cat(word.tmp[i], word.tmp[i+1], "¥n")
     }
}
--------------------
```

my2gram 実行

＝＞　1語だけの行がある。
＝＞　複数のスペースをひとまとめで切る！　¥¥W+

```
--------------------
my2gram <- function(a){
    lines.tmp <- scan(choose.files(), what="char", sep="¥n")
    data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
    body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
    body.tmp <- body.tmp[body.tmp != ""] #空行を削除。
```

```
     word.tmp <- unlist(strsplit(body.tmp, "¥¥W+"))   # 1文字以上の
英数字以外で切って「単語」バラバラに

     for (i in 1:length(word.tmp)-1) {
          cat(word.tmp[i], word.tmp[i+1], "¥n")
     }
}
--------------------
```

★ cat にオプションを付けて、ファイルに保存。

file="ファイル名"

上書きではなく、追記するオプションも付ける。

append=T

★ 保存するファイルを固定ではなく、最初に指定できるようにする。

```
     output.file <- choose.files()

--------------------
my2gram <- function(a,b){                                    # 目的語二
つ！
     output.file <- choose.files()
    lines.tmp <- scan(choose.files(), what="char", sep="¥n")
    data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value=T)
    body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
    body.tmp <- body.tmp[body.tmp != ""] #空行を削除。

     word.tmp <- unlist(strsplit(body.tmp, "¥¥W+"))   # 1文字以上の
英数字以外で切って「単語」バラバラに

     for (i in 1:length(word.tmp)-1) {
          cat(word.tmp[i], word.tmp[i+1], "¥n", file=output.file,
append=T)
     }
}
--------------------
```

my2gram() 実行

最初に保存ファイル名を入力（例　2gram.txt）

次に、分析対象ファイルを選択

結果はファイルとして保存される

保存されたファイルを読み込んでみる

```
table(scan(choose.files(), what="char", sep="¥n"))
```

R の中に入れる

```
bigramNS002 <- table(scan(choose.files(), what="char", sep="¥n"))
```

```
head(sort(bigramNS002, decreasing = T), 10)

--------------------
myTopBigram <- function(a) {
     tmp <- table(scan(choose.files(), what="char", sep="¥n"))
     head(sort(tmp, decreasing=T), a)
}
--------------------


myTopBigram(20)
```

１２月２２日

■学習者のレベルを選ぶ（TOEIC 600 点を例に）

考え方
１）属性情報のなかで TOEIC が６００点代の人をえらぶ

２）どう表記されているか？

```
@Begin
@Participants:  JPN002
@Age:  20
@Sex:  M
@YearInSchool:  U1
@Major:    medicine
@StudyHistory:  8
@OtherLanguage:  Spanish=0.6;none=
@Qualification:  TOEIC=620(2005);none=;none=@Begin
```

３）@Qualification の行に着目　＝　その行を取り出して調べる

「@Qualification」という文字列を検索　＝　grep

```
qualification.tmp <- grep("@Qualification", lines.tmp, value = T)
```

４）その行に６００点代と書いてあるかどうかで条件判断する。

TOEIC は、３ケタ　＝　６００点代は、６で始まる。

TOEIC=6　と書いてあるかを判断。

```
toeic <- (grep("TOEIC=6", qualification.tmp))
```

５）★条件判断の書き方★

```
if (判断の条件) {
   該当する場合に実行する命令
   }
```

６）grep で検索した結果に「値」が入っていればよい。

```
if (length(toeic) != 0){
   実行する命令
}
```

■TOEIC６００点代の人の Type と Token を出す。
（myTT の応用）

```
--------------------
function(){
    output.file = choose.files()

    files <- list.files()

    for (i in files) {
        lines.tmp <- scan(i, what = "char", sep = "¥n")

        qualification.tmp <- grep("@Qualification", lines.tmp, value
= T)
        toeic <- (grep("TOEIC=6", qualification.tmp))
         if (length(toeic) != 0){

         data.tmp <- grep("¥¥*(JPN|NS)...:¥t", lines.tmp, value = T)
         body.tmp <- gsub("¥¥*(JPN|NS)...:¥t", "", data.tmp)
         body.tmp <- body.tmp[body.tmp != ""]
         body.tmp.lower <- tolower(body.tmp)
         words.tmp <- unlist(strsplit(body.tmp.lower, "¥¥W"))
         words.tmp <- words.tmp[words.tmp != ""]
         type <- length(unique(words.tmp))
         token <- length(words.tmp)
         cat(type, token, "¥n", file = output.file, append = T)

         }

    }
}
--------------------
```

■頻度の検定　カイ自乗検定

例：　副詞の生起位置

```
a. However, such men don't make good husbands.
b. Such men, however, don't make good husbands.
c. Such men don't, however, make good husbands.
d. Such men don't make good husbands, however.
                      (Halliday, 1985: 82)
```

文頭・文中・文末のどこにでも置くことができる。

実際は、どうなのか、コーパスで観察する。

#LOB コーパス中の however の生起位置を調べたところ、次のような結果になった。

文頭 109
文中 347
文末 8

はて、本当にどこに置いてもよいのか、それとも、「偏り」があるのか。

頻度の差が、偶然による誤差のうちなのか、誤差とは言えないのか、カイ自乗検定で調べる。

```
--------------------
however.data <- c(109, 347, 8)

chisq.test(however.data)
--------------------
Chi-squared test for given probabilities
data: however.data
X-squared = 391.7371, df = 2, p-value < 2.2e-16
--------------------
#2.2e-16 意味は、2.2×10-6 乗 （つまりほとんどゼロ）
```

このバラつきは、偶然の誤差とは言えない。

★文末は、少ないのでほとんど使わないとして、文頭と文中の使い分けに何か原則があるのだろうか？

Sugiura, M. (1991：平成 3 年 3 月 1 日). The Distribution Environment of the Connective "However" and the Principle of Its Position ---Based on the LOB Corpus---. 『中部大学女子短期大学紀要・言語文化研究』, 2, 47-63.
http://oscar.gsid.nagoya-u.ac.jp/paper/sugiura1991distribution.pdf

■二次元の頻度差を考える。

イギリス英語とアメリカ英語で、 therefore の生起位置に違いがあるか。

位置	英	米
文頭	15	38
文中	96	53

これもカイ自乗検定。

```
--------------------
therefore.data <- matrix(c(15,96,38,53), nrow=2, ncol=2)

#matrix で行例作成。nrow は行数、ncol は列数。この場合は 2×2
--------------------

   [,1] [,2]
[1,]  15  38
[2,]  96  53

--------------------

chisq.test(therefore.data)
```

```
-------------------------結果の出力-----------------------
Pearson's Chi-squared test with Yates' continuity correction
data: therefore.data
X-squared = 19.1788, df = 1, p-value = 1.190e-05
----------------------------------------------------------------
```
（結果の解釈例）
イギリス英語とアメリカ英語で、therefore の生起位置に有意な偏りがある。

★どこに差があるかは「残差分析」をする。

★ところが、標準のパッケージには残差分析の関数は含まれていない。

他の人が作ったものを使わせてもらう。

1）ファイルをダウンロードして、手元に保存して、それを利用。
関数をテキストファイルで保存して、拡張子を「.R」にする。

source(choose.files()) で、ファイルを読み込む。

2）オンラインのファイルをそのまま URL で指定して読み込む。

群馬大学の青木先生のサイト
http://aoki2.si.gunma-u.ac.jp/R/

度数に関する検定
カイ二乗分布を使用する独立性の検定と残差分析

```
source("http://aoki2.si.gunma-u.ac.jp/R/src/my-chisq-test.R",
encoding="euc-jp")
```

my-chisq-test.R

分析したい数値を関数に送り、結果を変数 a に代入。

```
a <- my.chisq.test(matrix(c(15,96,38,53), nrow=2, ncol=2))
```

```
-------------------------結果の出力-----------------------
カイ二乗分布を用いる独立性の検定（残差分析）
data: matrix(c(15, 96, 38, 53), nrow = 2, ncol = 2)
X-squared = 20.6124, df = 1, p-value = 5.623e-06
----------------------------------------------------------
```

```
# 自動的には残差分析まで出力してくれない。以下を実行。
> summary(a)
```

```
-------------------------結果の出力----------------------
調整された残差
     [,1]   [,2]
[1,] -4.5401  4.5401
[2,]  4.5401 -4.5401

p 値
     [,1]       [,2]
[1,] 5.6231e-06 5.6231e-06
[2,] 5.6231e-06 5.6231e-06
-------------------------------------------------------------
```

（結果の解釈例）
カイ二乗検定の結果より、イギリス英語とアメリカ英語で、therefore の生起位置に有意な偏りがある。残差分析の結果より、イギリス英語では、アメリカ英語に比べて、therefore を文中で用いる頻度が有意に高く、文頭で用いる頻度が有意に低いことが分かる。

■対数尤度比（G スコア）
#言語データは偏りがあり、カイ二乗分布に近似させることには無理があるため、対数尤度比を利用する方が望ましいという指摘がある。

```
source("http://aoki2.si.gunma-u.ac.jp/R/src/G2.R", encoding="euc-jp")
```

４．分析したい数値を関数に送り、結果を変数 a に代入

```
> G2(matrix(c(15,96,38,53), nrow=2, ncol=2))
-------------------------結果の出力----------------------
対数尤度比に基づく独立性の検定（G-squared test）
data: matrix(c(15, 96, 38, 53), nrow = 2, ncol = 2)
G-squared = 20.9249, df = 1, p-value = 4.776e-06
-------------------------------------------------------------
```

（結果の解釈例）
p< .001 より、イギリス英語とアメリカ英語で、therefore の生起位置に有意な偏りがある。

以上