R でプログラミング：データの一括処理とグラフ描き

6. グラフの重ね描き

グラフの重ね描きのしかたを覚えると，複数の線を引いたり， 散布図のうえに回帰直線を引いたりできるようになります． 思いどおりに重ね描きするために注意することをまとめてみます．

※なんだかめんどうだなあ，マウスでこちょこちょやるグラフ作製ソフトのほうが 簡単だなあ，という気がしてきたら，もういちど第４章の 最初のところ，「プログラムでグラフを描くメリット」を読んでみてください． もしかしたら，ちょっとやる気が戻るかもしれません．

重ねるための予備知識と基本的な手順

高水準作図関数と低水準作図関数

すでに前に書いたように，作図関数には高水準作図関数と 低水準作図関数がありました．この二つのグループの関数の特徴のうち， とくにグラフを重ね描きするときに重要になる点を表にしてみます．

作図関数の種類	関数の例	座標系	既存画面の消去
高水準	plot, hist, curve 他	• データから自動設定 • xlim, ylim で指定も可	• 描画前に消去する． • 消去させないためには par(new=T)
低水準	lines, points, text, segments, legends, rug 他	• すでに用意された座標系上に描画	• 消去せずに上書き

座標系の設定というのは，横軸と縦軸それぞれの最小値，最大値を いくつにするかを決めるということです． 高水準作図関数は，与えられたデータの値の範囲から，それらがうまく おさまり，かつ軸の両端が中途半端な値にならないように，座標系を 設定してくれます．

高水準作図関数は，ふつうは前のグラフを消して新しい図を書きます． けれども，描画前に par(new=T) として，グラフィックパラメータ new にT(真) を設定すると，まっさらな作図デバイスが用意されたものと思って，消去作業をしません． ただし，そこに何が描かれているかはいっさい関知しない（なにせ白紙だと思っている）ので，前の作図のときの座標系も知りません． ですから，同じ座標系でプロットを重ねたいなら，座標系を明示的に指定してやる必要があります．

＜練習＞

前のページで， t <- read.table('temperature.txt', header = T) で読み込んだデータを使って，以下のことを試してみる これで，plot で重ね描きするには工夫が必要だということを実感する．

- plot(tday, tSite01, type = 'l') としたあと，plot(tday, tSite03, type = 'l') として， 最初に描いたグラフは消えてしまうこと，ふたつのグラフで縦軸のスケールが異なることを 確認する．
- plot(tday, tSite01, type = 'l') のあと，par(new=T) としてから plot(tday, tSite03, type = 'l') と入力して， スケールがずれたまま重ね描きされることを確かめる．

なお，一部の高水準作図関数（curve など）では，add という引数を設定可能です． これを add =T と真に設定して作図関数を呼び出すと，画面の消去をせず，座標系の再設定も せずに重ね描きしてくれます．curve は，データをプロットしたあと，これにあてはめた 式のグラフを重ね描きするような場合に便利です． くわしくはあとで説明します．

重ね描きの基本的な方針

うえで整理した作図関数の特性を踏まえると，いくつかの基本的な方針がたてられます． ここでは３つの方針を紹介します．

方針１ 毎回スケール指定をしながら高水準作図関数で重ね描き

- 高水準作図関数を繰り返し使う．
- ２回め以降は，関数呼び出し前に，par (new =T)として， 前の絵を消さないようにしながら重ねていく．
- 同じ座標系で描けるように，すべての作図関数でスケールを指定する (xlim = c( ), ylim = c( )) ．
- 軸の名前（ラベル）だのタイトルだのをなんども上書きしないように， 一度だけ書いたらあとは空の文字列 "" を設定する．
- あとから文字列だの矢印だの凡例だのを低水準関数で加筆する．

方針２ 高水準作図関数で座標系を設定してから低水準関数でプロット

- まずは高水準作図関数で描画する.
- ワクだけ書いてプロットはしなくてもよい (plot(..., type = 'n')) し, プロットしてもよい. やりやすいように.
- 低水準描画関数で要素を書き加えていく. 高水準作図関数の描画時に設定された座標系がそのまま生きている.

方針３ add が可能な高水準作図関数で重ね描き

- まずは高水準作図関数で描画する.
- 引数として add が設定可能な高水準作図関数で, add = T として重ね描きする. この場合は par(new=T)は不要. また, 座標系はその前に設定されたものがそのまま 生きている.

以下では, 方法１から方法３の例をそれぞれ紹介します.

繰り返し高水準作図関数を使う

方針１のやりかたで, 'temperature.txt' のデータを使って, ３地点 (Site01, Site02, Site03) の温度データの一年間の変化を, 一枚のグラフに書いていみます. 以下のコードを実行して みましょう.

なお, 描画画面を表示したままいろいろパラメータ設定を変えて試していると, 前のパラメータ設定が残っていて思わぬ結果になったりします. まず dev.off() で画面を閉じてからはじめると, こうした混乱を避けられます.

```
t <- read.table ('temperature.txt', header = T)  # データの読み込み

#  縦軸の範囲を指定して, 折れ線グラフを描く.
plot(t$day, t$Site01, type = 'l', ylim = c(-10, 30),
     col = 'black', xlab = 'Day', ylab = 'Temperature')

par(new = T)   #  上書き

plot(t$day, t$Site02, type = 'l', ylim = c(-10, 30),  #  縦軸の範囲は前の図と同じ.
     col = 'blue', xlab = '', ylab = '')   # 軸のラベルが重ならないように '' を設定

par(new = T)   #  上書き

plot(t$day, t$Site03, type = 'l', ylim = c(-10, 30),  #  縦軸の範囲は前の図と同じ.
     col = 'green', xlab = '', ylab = '')   # 軸のラベルが重ならないように '' を設定
```

この例で, xlim の設定をしていないのは, ３回の描画でいずれも t$day を横軸にしている おり, plot が自動的に設定する軸の最小値, 最大値も共通になると考えられるからです. そのほかは, とくに新しく説明するべきことはありません.

縦軸の範囲を c(-10, 30) のように具体的な値で与えていますが, これもデータによって プログラム中で決められればそのほうが便利ですね. それでこそプログラムが生き

ます． そういう方法は，あとからいろいろ出てきます．

＜練習＞

以前に使った len_width.txt のデータで，一枚のグラフに2種類の点をプロットする．

- d <- read.table('len_width.txt', header = T) でデータフレームにデータを読み込む．
- d.sp1 <- d[d$sp == 'Sp1',] として，sp の列の値が 'Sp1' のデータのみからなる データフレーム d.sp1 を作る． （ 第２章 の，「条件を指定して行を選ぶ」の節を参照）
- 同様にして，'Sp2' のデータからなるデータフレーム d.sp2 を作る．
- 上の例を参考にして，pch ないしは col に別の値を指定して plot で 重ね描きし，Sp1 と Sp2 それぞれの d$len と d$width の関係をしめすグラフを描く．
- xlim は c(0, 80)，ylim は c(0, 30)とする．

＞ できあがり参考例

＞ プログラム例 （まずは見ないで書いてみること）

高水準作図関数で座標系を設定してから低水準作図関数で重ね描き

最初の表で整理したように，低水準作図関数は自分では軸のスケールを設定しません．すでに設定された座標系のなかで作図します． 上の練習で使った d.sp1 と d.sp2 を，低水準作図関数 **points** を使って プロットしてみます． **points** は，長さがおなじ二つのベクトルをうけとって， それぞれを縦軸，横軸の値とみなして点を配置する関数です． なんだか**plot**と同じみたいですが， **plot**はすでに説明したように総称的関数で，与えるデータしだいで いろんなグラフを描きますし，高水準作図関数ですので，データの値の範囲にもとづいて 自分で軸をきめます．さらに，軸を描いたり，軸ラベルを描いたりしてくれます． それに対して**points**は，点を配置することしかしません．

points と plot それぞれの特徴を生かして，組み合わせてみます．

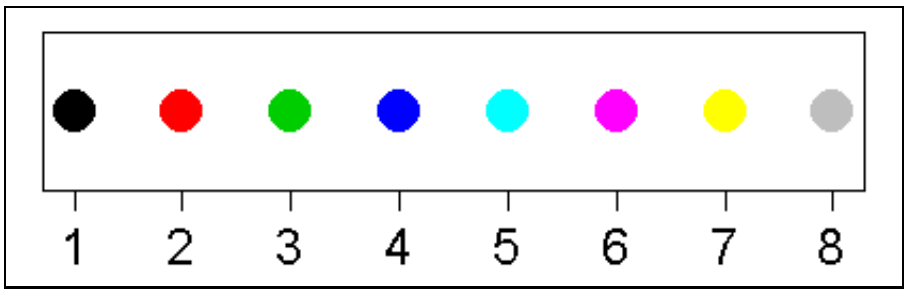
```
d <- read.table('len_width.txt', header = T)
d.sp1 <- d[d$sp == 'Sp1',]
d.sp2 <- d[d$sp == 'Sp2',]

#type = 'n' で，プロットはせず軸やラベルだけ描く．
plot(d$len, d$width, type = 'n', xlab = "Length", ylab = "Width")
```

```
# それぞれの種の点をプロット．色は番号で指定．cex で大きめの点に．
points(d.sp1$len, d.sp1$width, col = 1, cex = 1.5)
points(d.sp2$len, d.sp2$width, col = 2, cex = 1.5)
```

＞ できあがり参考例

plotには，データに応じた座標系を決めて軸やラベルを描くという 仕事だけさせています．種ごとに分けずにすべてのデータを渡しているので， 全部のデータがおさまりきるように座標系を決めてくれます．

plot が用意してくれた座標系のなかに， **points** でそれぞれの種の点をプロットしていきます． シンボルは同じ（ディフォルトの〇）ですが，色（col）をそれぞれ番号で指定しています． 色にはこのような指定も使えます．１番から８番まであって，色との対応は下の図のとおり です．



この例では，Sp1 と Sp2 という２種類の種のデータがあるという前提の書き方を していますが，何種類あって，それぞれなんという名前かをデータから判断するように 書ければもっとプログラムらしくなりますね．そういう技は次のページで出てきます．

また，このように複数種の点や線をプロットした図では，凡例も書き加えたく なります．凡例は，**legend** という低水準作図関数で書込みますが， これについても次のページで

plot で座標系だけ決めて，あとはすべて低水準作図関数を駆使して 作図したいというような場合，**plot** に渡したいデータはなにもありません． でも，**plot**はなにもデータを渡さないと文句を言って仕事をして くれないので，ダミーのデータを渡します．たとえば，

```
plot(0, 0, type = 'n', xlim = c(0, 100), ylim = c(0, 100), xlab = '', ylab = '')
```

とすれば，プロットするべきデータとしては (0, 0)の一点だけを渡す，でも type = 'n' なのでそれも実際にはプロットされず， x 軸， y 軸とも 0から 100 の座標系と枠の線だけが 用意される，ということになります． 枠の線も不要なら， axes という引数に F (FALSE, 偽) を設定します．

高水準作図関数で，add = T を指定して重ね描き

一般化線形モデルのあてはめ

一部の高水準作図関数は，add という引数を設定可能です． add が T(TRUE) だと，画像の消去や座標系の設定はしません． 直前に設定せれた座標系のうえで描画します．

その例として，二次元の散布図に，回帰直線を重ねてみます． 回帰直線を求めるのには，**glm** を使ってみます． **glm** は，一般化線形モデル（generalized linear model）のあてはめを行う関数です． 一般化線形モデルは，まだなじみがないかた，名前だけは聞いたことがあるけど というかたも多いかと思います． 包括的かつ初心者むけの説明がなかなかないのですが， 使用例は増えつつあります．

もっとも単純な場合には，ふつうの直線回帰と同じことをしてくれますが， 直線回帰をおこなうに当たっての仮定や制約を可変にして， より一般性の高い統計モデルを求めることができます．

このページでは統計解析手法の解説はしません（できません）から，詳しい説明は ほかにゆずって，まずはもっとも簡単に直線回帰モデルのあてはめをしてみます．

あてはめる統計モデルには独立変数ひとつの一次式を書いて， あとは特別になにも指定しなければ，ふつうの直線回帰の条件 （誤差は正規分布に従い，その分散の大きさは一定）で統計モデルをあてはめてくれます．

```
d <- read.table('len_width.txt', header = T)
lf <- glm(d$width ~ 1 + d$len)
```

glmには，どの変数を，どの変数で説明したいかを 表現した式をわたします． y = b + ax という形で，x を使って y を説明（予測，推定）したいなら， **y ~ 1 + x** のように書きます． ~ の左側の変数を，~ の右側の形をした式で説明したい，という意味です． 1 は定数項があるよという意味（ 0 にすれば定数項なし）， **x** は x の一次の項も 欲しいという意味．全体で，y を ax + b の形の一次式で推定する統計モデルを 作ってくれ，という意味になります．

上のプログラムでは，**d$width ~ 1 + d$len** という式をわたしていますが， これは d$width を d$len の一次式（定数項あり）で推定する式を求めたい，という意味です． このあてはめの結果がしまわれたオブジェクトを lf に代入しています． このlfの内容を見てみましょう．入力画面で単に lf と入力すると， このようなものが表示されます．

```
Call:  glm(formula = d$width ~ d$len + 1)

Coefficients:
(Intercept)        d$len
   -0.06156      0.24132

Degrees of Freedom: 49 Total (i.e. Null);  48 Residual
Null Deviance:      1113
Residual Deviance: 848.8        AIC: 289.5
```

最後の AIC は赤池の情報量基準で，モデルがどのぐらいよくデータに 当てはまっているかを表すものです． （参考： 群馬大・青木さんの統計学用語辞典中の AIC）． ここでは詳しく説明しません．

Coefficients: の下， (Intercpet) と d$len の ところの数値が，それぞれあてはめた直線の切片と傾きの値です． 第３章の，相関関係の計算結果を見たところを思い出しつつ， lf に含まれる内容を取り出してみましょう．まず，どんな名前のデータがあるかを， **names**で調べてみます．

```
names(lf)
```

とすると，

```
 [1] "coefficients"      "residuals"         "fitted.values"
 [4] "effects"           "R"                 "rank"
 [7] "qr"                "family"            "linear.predictors"
[10] "deviance"          "aic"               "null.deviance"
[13] "iter"              "weights"           "prior.weights"
[16] "df.residual"       "df.null"           "y"
[19] "converged"         "boundary"          "model"
[22] "call"              "formula"           "terms"
[25] "data"              "offset"            "control"
[28] "method"            "contrasts"         "xlevels"
```

のように，30種類ものデータを含んでいることがわかります．このなかで，あてはめたモデル の傾きと切片は，最初の lf$coefficients に入っていそうです．はたして，

```
lf$coefficients
```

と入力すると，

```
(Intercept)       d$len
 -0.0615620   0.2413217
```

と表示されます．名前のついた２つの数値データからなるベクトルですね． ふたつの要素それぞれは，d$coefficients[1]，d$coefficients[2] のように指定します．

式で表される線をグラフに描く

モデルをあてはめることができたら，それを表す線をグラフに書込みます． これには **curve** という高水準作図関数を使います． 引数に式を与えると，その式のグラフを描いてくれる関数です． たとえば，

```
curve(x^2, from = -10, to = 10)
```

と入力すると，x が -10 から 10 の範囲で x^2 （x の二乗）のグラフを描いてくれます （＞作図例）．from と to が，x の範囲を設定する引数です．

縦軸のスケールは勝手に決めてくれるし，軸も描いてくれるし，たしかに高水準作図関数です． この関数を add という引数に T(TRUE) を設定して呼び出すと， 画面の消去をせず，すでに存在する座標系のなかにグラフの線だけが描かれます． まるで低水準作図関数のような振る舞いです （といっても，式からグラフを描くのはやはり高水準な仕事ですが）．

これを使って，上で求めた直線回帰モデルを表す線を，散布図の上に重ねて描いてみます． x の範囲は，関数 **min**と**max**で求めています．

```
d <- read.table('len_width.txt', header = T)
plot(d$len, d$width)                # 散布図を描く
lf <- glm(d$width ~ 1 + d$len)      # モデルのあてはめ結果を lf にしまう．
cf <- lf$coefficients               # あてはめたモデルの係数を変数 cf にしまう．
curve(cf[1] + cf[2] * x, from = min(d$len), to = max(d$len), add = T)
```

＞ できあがり参考例

グラフ中に式を書き込む

せっかく求めた式の切片と傾きも，グラフ中に書き込むことにしましょう． これには，**text** という低水準作図関数を使います． 文字列を配置する場所のx座標とy座標，そして文字列の内容をわたします． （そのほかにもいろいろな設定ができます．くわしくは help(text) で調べてください）．

さて，ここでちょっと困るのが座標の指定です．たとえばグラフの上のほうのまん中に書きたいとして，そこの xy 座標を知らないと指定のしようがありません． 点をプロットしてから眺めればおおよそ分かることですが，それではプログラムで 自動化するのには不都合です． となると，今，座標系はどうなってるのか， x 軸， y 軸の両端の値を調べてから， 適当な座標を計算してやればよさそうです．軸の両端の値は，グラフィックパラメータ usr にしまわれてきます．

```
par('usr')
```

とすると，たとえば

```
[1] 24.580 73.720  2.876 25.124
```

のように４つの値が表示されます．はじめから順に x軸の最小値，最大値， y軸の最小値，最大値です．この， ４つの数値データが並んだベクトルを一度 変数にしまってから，その値を使って適切な配置場所の座標を計算することに しましょう．たとえば，

```
par('usr') -> usr
x <- (usr[1] + usr[2]) / 2             #  x軸のまん中
y <- (usr[4] - usr[3]) * 0.9 + usr[3]  #  y軸の下から 90% のところ.
```

のようにして，適当な座標を決めます． glm で求めた式の内容は， 第３章のなかの「計算結果をファイルに記録する」のところで紹介した **sprintf**をつかって文字列にします.

```
eq <- sprintf('y = %.3f * x + %.3f', cf[2], cf[1])
```

%.3f というのは，実数として，小数点以下３ケタまで表示せよ，という書式指定子です． これを実行すると，変数 eq には "y = 0.241 * x + -0.062" というような文字列が記録されます.

ここまで準備ができれば，あとは **text** を呼ぶだけです． ついでに，グラフ全体のタイトルも書きこみましょう． これは，plot 関数で main という引数に設定します.

```
plot(d$len, d$width, main = "Model fitting using glm")                  #  散布図を描く
```

ここまでのプログラムをひとつにまとめてみます.

```
d <- read.table('len_width.txt', header = T)
plot(d$len, d$width, main = "Model fitting using glm")  #  散布図を描く
lf <- glm(d$width ~ 1 + d$len)      #  モデルのあてはめ結果を lf にしまう.
cf <- lf$coefficients               #  あてはめたモデルの係数を変数 cf にしまう.
curve(cf[1] + cf[2] * x, from = min(d$len), to = max(d$len), add = T)
par('usr') -> usr                       #  現在の座標系を取得
x <- (usr[1] + usr[2]) / 2              #  x軸のまん中
y <- (usr[4] - usr[3]) * 0.9 + usr[3]   #  y軸の，下から 90% のところ.
eq <- sprintf('y = %.3f * x + %.3f', cf[2], cf[1])
text(x, y, eq)
```

＞ できあがり参考例

だいぶ盛り沢山の内容でしたが，なかなかりっぱなグラフが描けました． これまでの知識を使ったり，さらにいろいろ調べたりすれば，さらに 見栄えのよいグラフが描けるでしょう.

＜練習＞

- d$sp が Sp1 のデータと Sp2 のデータとを色分けしてプロットし， それぞれにあてはめたモデルのグラフを重ねて描く.
- それぞれのモデルの式も書き込む.

＞ できあがり参考例

＞ プログラム例 （まずは見ないで書いてみること）.

これまでの知識を総動員する練習です． ぜひ試してみてください．

このように複数の種類のデータがプロットされているグラフでは，凡例も 表示させたいところです．凡例の描画には専用の関数が用意されています． それについては次のページで紹介します．

グラフを並べる

この章の最後の節は，グラフを重ねるのではなく，並べる話です．

前の章で， デバイス領域，作図領域，プロット領域について簡単に紹介しました． デバイス領域をいくつかに区切って，その区画をじゅんに作図領域にして グラフを書いていけば，一枚の'紙'にいくつものグラフを並べることができます． 同じ形式のグラフをたくさん並べるのにも使えるし， 関連がある複数のグラムを見比べるために使える方法です．

作図領域を **par**(fig = c(....)) で指定しながらグラフを描いていくこともできますが， もっと簡単に，デバイス領域を ２×２に区分して使いますよ，とか， ３×４にして使いますよ，と設定すると，新しいグラフを描くごとに自動的に 作図領域をずらしてくれる機能があります．

そのためには，**par**で mfrow あるいは mfcol というパラメータに， たてに何区画，よこに何区画という２つの数値からなるベクトルを設定します． mfrow を使うと，左上から右へ描き進め，一行目が一杯になったら二行目へ，と進みます． また， mfcol を使うと，左上から下へ描き進め，一列目が一杯になったらニ列目へと 進みます． たとえば３行４列にグラフを並べる，描き進む順序は１行目，２行目…なら，
par(mfrow = c(3, 4)) と設定します．

＜練習＞

- par(mfrow = c(3,2)) としてから，plot で適当なグラフを何度も繰り返し描いて， 描画画面上に順にグラフが並んでいくのを確かめる．
- mfrow のかわりに mfcol を設定して，同様のことを試みる．

ところで，この方法でグラフを並べていって，一枚の'紙'がいっぱいになった場合，なにが起こるかはデバイスによります． 描画画面や，png，jpg などのビットマップ画像の場合は， 次のグラフを描くときに一度すべてが消去されてしまいます．

一方，pdf や postscript のように，ページという概念があるデバイスでは， 新たなページが足されて，そこに描きこまれていきます．

※ wmf （ウィンドウズメタファイル）では，もう描けない状態になると ファイルを作れませんとメッセージが表示されます．さらに， R のプログラム自体が停止してしまったりするようです．

したがって，全部で何個のグラフがあるのか，一枚の ’紙’ に入りきるか どうか分からずに画像ファイルを作る場合には，デバイスによっては それなりの処理が必要です． 複数枚のページにわたって描いてくれない デバイスの場合には，’紙’ の容量一杯（あるいはその前でも きりのいいところ）までグラフを描いたら一度 **dev.off()** し， あらたに画像ファイルのデバイスドライバを起動します． 当然，このファイルは前に作ったファイルと別の名前にしないといけません． 同じ名前だと，先に作ったファイルの内容は上書きされて消えてします．

| Top Page | プログラミング | R 自動化 目次 | 索引 | 前へ | 次へ |