

R の操作と R を用いた統計解析

飯島勇人^{*}

2024 年 12 月 19 日

この資料について

本資料は、R の操作や R による統計解析を初学者が習得するためのものです。できるだけ実際に R で作業を行い、実体験を通して学ぶ内容となっています。本資料は飯島が作成したのですが、様々な文献や講義等から学んだものの集合体です。ただし、できるだけ努力はしていますが、内容に不正確な部分があるかもしれません。あくまで自己責任での利用をお願いします。

目次

第 I 部 基礎	4
1 はじめに	4
1.1 データファイルを置く場所と場所の指定	4
1.2 使うデータ	8
2 まずは触ってみよう！	8
2.1 パッケージ	8
2.2 R にデータを読み込ませる方法	9
2.3 まず体験してみる～R と tidyverse で楽々データ操作～	11
2.4 まず体験してみる 2～R と ggplot で楽々作図～	15
3 データ操作	17
3.1 部分的な選択や削除	17
3.2 演算関数	19
3.3 列の追加	20
3.4 カテゴリーごとの操作	20
3.5 文字列操作	22
3.6 パイプライン処理の利点	29
3.7 繰り返し動作	29
3.8 ティブル同士の結合	31

* (国研) 森林総合研究所野生動物研究領域

† 連絡先: hayato.ijiima@gmail.com または hayatoi@affrc.go.jp

3.9	複数列に対する一括処理	34
4	作図	35
4.1	基本的な記法	35
4.2	複数の図	42
4.3	装飾	44
4.4	図の保存方法	48
第 II 部 R における地理情報データの扱い		49
5	はじめに	49
5.1	II 部の事前準備	49
6	データの入手	49
7	データの読み込みと操作	50
7.1	データの読み込み	50
7.2	データの操作	54
8	作図	59
8.1	データ単独の作図	59
8.2	背景地図の活用	60
8.3	動的な地図	61
第 III 部 統計解析		63
9	はじめに	63
9.1	III 部の事前準備	63
9.2	ベイズ統計とは?	63
10	確率分布	65
10.1	正規分布	65
10.2	二項分布	66
10.3	ポアソン分布	68
11	尤度	71
11.1	対数尤度	72
12	GLM (一般化線形モデル)	74
12.1	確率論的モデル	74
12.2	リンク関数	74
12.3	データの読み込み	76

12.4	R の関数による GLM	77
12.5	MCMC 法	79
12.6	BUGS 言語によるモデルの記述	82
12.7	データの準備	84
12.8	初期値の設定	85
12.9	監視対象パラメータの設定	85
12.10	MCMC 法の条件設定	85
12.11	MCMC 法の実行	86
12.12	結果の検証	87
13	GLMM (一般化線形混合モデル)	90
13.1	変量効果とは何か？	90
13.2	R の関数による解析	91
13.3	BUGS 言語によるモデルの記述	92
13.4	データの準備	93
13.5	初期値の設定	93
13.6	監視対象パラメータの設定	93
13.7	MCMC 法の条件設定	93
13.8	MCMC 法の実行	94
13.9	結果の検証	94
13.10	収束しないときの対処法	95
第 IV 部 Rmarkdown による簡易レポート作成		98
14	はじめに	98
15	Rmarkdown による文書の基本	98
15.1	文書作成の手順	98
15.2	マークダウン記法	98
15.3	チャンク	100
15.4	yaml ヘッダ	101
15.5	文書の出力方法	101
第 V 部 演習の答え		103
索引		106

第Ⅰ部

基礎

1 はじめに

I部は、初学者がRを使ってある程度のデータ解析ができるようになることを目的としています。そのために、Rの基本的な使い方と、基礎的な統計解析について説明します。

Rは、基本的にマウスで操作するのではなく、ファイルを読み込んだり図を書いたりといったあらゆる作業を、コマンドで入力します。マウス操作に慣れた方は、コマンドを覚えたりコマンドによる操作が苦痛に感じられると思います。ですが、後述するようにコマンドによる操作は利点もあります。

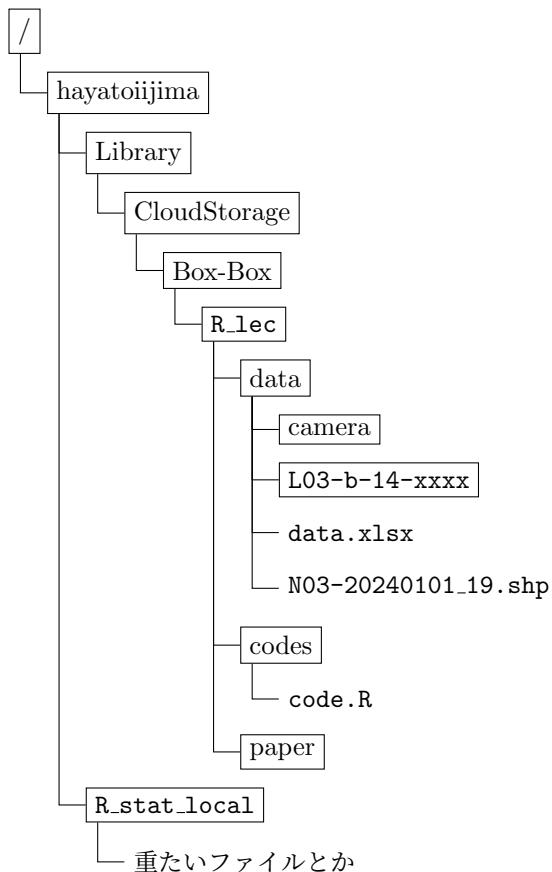
1.1 データファイルを置く場所と場所の指定

1.1.1 ファイルを置く場所

基本的には、好きな場所で構いません。ただし、個人的な流儀の問題になりますが、飯島は以下のようなフォルダ構造を構築した上で、データを「data」というフォルダに格納することをお勧めしています。

- 自分が好きな場所に、あるプロジェクトの起点となるフォルダを作成する。プロジェクトとは、1つのまとまった仕事（例：論文）が終了するまで使う一連のファイルが置かれているフォルダとする。今回のセミナーを一連のプロジェクトと見立て、まずR_lecというフォルダを作る。
- その中に、dataというフォルダを作る。同じ階層に、飯島はcodesというフォルダ（のちに説明する、Rに出す命令を書いたファイル）と、paperというフォルダ（論文の原稿や論文中で使う図表などを置くフォルダ）を作ることが多いです。
- ファイルサイズが大きいデータを扱う場合、上記の基本フォルダがクラウド（Dropboxなど）にある場合は、別途自分のPCのハードディスクやNAS（R_lec_localのような名称とする）に置く。

飯島の場合、さまざまな場所、PCで作業することが多いため、以下のようなクラウド環境に置いています。



1.1.2 データファイルの場所の指定

上記で、データ（とそれを扱うプロジェクトフォルダ）を好きな場所に配置しました。次は、データがある場所（フォルダ、ディレクトリ）を R に教えてあげる必要があります。教える方法にはいくつか選択肢があり、以下のとおりです。これらについて、説明します。

- GUI 上でデータがある場所を毎回指示する
- コマンドでデータがある場所を毎回指示する（絶対パス）
- コマンドでデータがある場所を毎回指示する（相対パス）

R では、以下のように GUI 上で使ってデータがある場所を指示することが可能です（図 1）。以下の図は macOS の場合ですが、Windows の場合は画面の左上の「ファイル」から「ディレクトリの変更」を選択します。

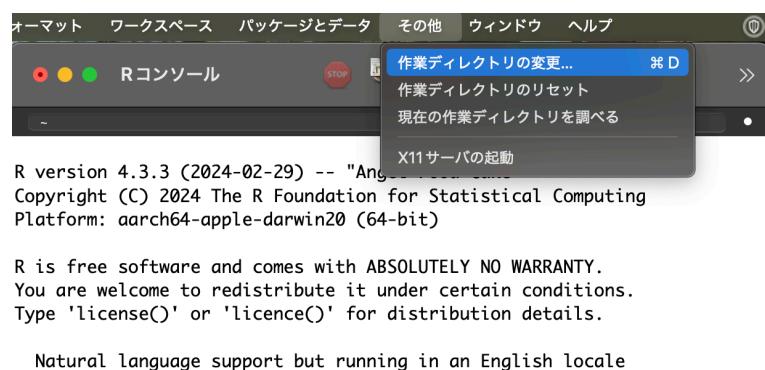


図1 マウスによる作業ディレクトリの指定

データの位置がほぼ固定の場合はこの方法でもいいと思いますが、データフォルダ内にもフォルダがある場合や、解析結果をファイルとして出力する場所が違う場合、その都度 GUI で作業ディレクトリを指定することは煩雑です。一方、コマンドで指定できれば、あらかじめコマンドを書いておくことで効率的に指示ができます。R に作業ディレクトリを教える関数として、`setwd` 関数があります。

`setwd` による最も単純な指定は、作業ディレクトリまでのパスを以下のように全て書き下すことです。

絶対パスによる作業ディレクトリの指定

```
> setwd("/Users/hayatoijima/Library/CloudStorage/Box-Box/R_lec/")
```

この方法の場合、毎回パスを書く必要があり、やや煩雑です。そのためにも、コマンドを入力する際、いきなり R に文字を打ち込むのではなく、必ずメモ帳などでテキストの形でコマンドを下書きして、それをコピペすることで行ってください。コードを残しておくと、間違っていた際の修正に便利です。また、#という記号の後は R の中に無視されます（コメントアウト）。そのため、コード中に日本語で何をする操作なのか書き込んで、コメントアウトして残しておくことができます。早速、コマンドを打ち込むためのテキストファイルを、各自で用意しましょう。

仮に作業ディレクトリの指定を GUI で行うつもりの人でも、データの読み込みや作図などで多数のコマンドを使うことになりますので、コードファイルは必要です。さらに、コードファイルを作成することは、結果の再現性を担保する上で非常に重要な役割を果たします。近年、論文として発表した結果の再現可能性を担保することが、不正防止という観点から求められています。自身で不正をする気がなくても、論文を投稿する際に、生データと生データから結果を得るまでの過程を開示するように求められることがあります。このような点からも、コードとして作業過程を残しておくことは重要です。

絶対パスによる指定は、自分以外の人とコードを共有する場合に、PC のディレクトリ構造が違っていたり、そもそも OS が違っていると不都合が生じます。

これらの点を解消する記法が、相対パスです。相対パスとは、ある起点となる場所を定め、そこから「～階層下のこのディレクトリ」のような指定をする方法です。そのため、まず「起点となるディレクトリ」を指定する必要があります。起点となるディレクトリを指定する方法は、いくつかあります。また、相対パスで指定する上では、`here` パッケージの `here()` 関数が便利です。

起点となるフォルダの設定 1

```
> setwd("/Users/hayatoijima/Library/CloudStorage/Box-Box/R_lec/")
> library(here)
```

この方法の場合、最初の 1 回だけ絶対パスを指定すれば、あとは `here` パッケージによる相対記法を使うことができます。ですが、コードを異なる OS の人と共有する場合、区切り文字が OS 間で異なるという問題があります。それを回避するためには、以下のようないすすめをします。

起点となるフォルダの設定 2

```
> setwd(file.path(path.expand("~/"), "Library", "CloudStorage", "Box-Box", "R_lec"))
> library(here)
```

`path.expand` は、作業している PC のフォルダ構造を認識し、ホームフォルダ（ユーザー名直下）までのパスを取得してくれる関数です。もし自身の OS でどこが参照されるのか確認したいときは、`> path.expand("~/")` として確認して下さい。また、`file.path` は、OS を自動で認識し、ディレクトリ間の区切り文字を OS に適

した形で与えてくれる関数です。

どの方法にも一長一短がありますが、飯島は一番最後の記法を用います。これ以降のコードでは、皆様も上記の構造で作業しているとみなしてパスの指定などを行います。

1.2 使うデータ

ここで使うデータについては、すでに配布しています。後ほど皆さん的手元でも読み込んでもらいますが、ひとまずどのようなデータか説明します。

調査者は森林の研究者で、近年日本各地で個体数が増加しているニホンジカが森林動態に与える影響について調べようとしています。そこで、ニホンジカ密度が異なると思われる 20 の地域で、毎木調査と稚樹調査を行いました。1 地域で、毎木調査は 2 箇所（1 箇所のサイズは 20×20m）、毎木調査の箇所ごとに稚樹調査は 5 箇所（1 箇所のサイズは 2×2m）で実施しました。仮説として、ニホンジカが多い地域では、ニホンジカによる樹皮剥ぎが多く、また稚樹の本数は少ないということを考えています。また、ニホンジカの個体数を調べるために、各地域に 5 台ずつ自動撮影カメラを設置しました。

毎木調査の項目。

`Region` 地域の ID

`Stand` 每木調査箇所の ID

`Species` 調査した個体の樹種

`DBH` 調査した個体の胸高直径 (cm)

`Debark` 剥皮されていれば 1、されていなければ 0

稚樹調査の項目。

`Region` 地域の ID

`Stand` 每木調査箇所の ID

`Quadrat` 稚樹調査箇所の ID

`H` 調査した個体の地上高 (cm)

`Species` 調査した個体の樹種

2 まずは触ってみよう！

この章では R にデータを読み込んで、整形・加工する方法を学びます。R での作業の全ての基礎となります。

2.1 パッケージ

R はそのままでも多くの機能を有していますが、追加的に必要な機能は「パッケージ」としてまとめられており、必要に応じて読み込む必要があります。

これ以降で使うパッケージについて、先に読み込んでおきます。では、これが出ている状態で、パッケージを読み込んでみましょう。ここでは、データの操作性を高める `tidyverse` パッケージ、および Excel 形式 (`.xls` および `.xlsx`) ファイルを読み込む関数を含んでいる `readexl` パッケージを読み込みます。

パッケージの読み込み

```
> library(tidyverse)
> library(readxl)
```

以上のコマンドを打ち込んでエンターキーを押せば、パッケージが読み込まれます。

2.2 R にデータを読み込ませる方法

2.2.1 データファイルの作り方

- Excel などにデータを打ち込む。
- 各列に個体番号、環境条件などを入れる。
- 1行目にはデータのラベルを入力する。
- 基本的に空のセルを作らない（欠損値がある場合は空欄でもよいが、NA を入力しておいた方がいい）。
- 違った形式（並べ方が異なる）のデータを同一ファイル内に混在させない。

2.2.2 R ではオブジェクトにデータを与える

R でデータを操作する基本単位は、オブジェクトです。オブジェクトとは、「適当な文字」です。自分で適当につけた文字にデータをくっつけて、データをいじります。では、オブジェクトがどういうものか、試してみましょう。

データの読み込み例（GUI で作業ディレクトリを指定）

```
# ここで GUI で「data.xlsx」が保存されているディレクトリを指定
> treedf <- read_excel("data.xlsx", sheet="Trees")
# read_excel("ファイル名", sheet="読み込むシート名") という書き方
```

データの読み込み例（コマンドで絶対パスを指定）

```
# 絶対パスで作業ディレクトリを指定
> setwd("/Users/hayatoijima/Library/CloudStorage/Box-Box/R_lec/data/")
> treedf <- read_excel("data.xlsx", sheet="Trees")
# read_excel("ファイル名", sheet="読み込むシート名") という書き方
```

データの読み込み例（コマンドで相対パスを指定）

```
# 相対パスで作業ディレクトリを指定
> setwd(file.path(path.expand("~/"), "Library", "CloudStorage", "Box-Box", "R_lec"))
> treedf <- read_excel(here("data", "data.xlsx"), sheet="Trees")
# read_excel("ファイル名", sheet="読み込むシート名") という書き方
```

- 以上のように打ち込むことで、treedf という文字に毎木データ.csv のデータをくっつけることができます。`read_excel` は.xls および.xlsx 形式のファイルを読み込むための関数です。
- treedf がオブジェクトです。
- 本当に読み始めたのか、

```
> treedf
```

で確認します。以下のようにデータが表示されるはずです。

```
# A tibble: 1,122 × 5
  Region Stand   DBH Species      Debark
  <chr>   <chr> <dbl> <chr>        <dbl>
1 A       ST1    35.5 Genus_speciesM     0
2 A       ST1    40.1 Genus_speciesO     0
3 A       ST1    42.2 Genus_speciesD     0
4 A       ST1    41.9 Genus_speciesG     0
5 A       ST1    39.5 Genus_speciesA     0
6 A       ST1    25.0 Genus_speciesQ     1
7 A       ST1    43.7 Genus_speciesB     0
8 A       ST1    45.2 Genus_speciesO     0
9 A       ST1    35.2 Genus_speciesS     0
10 A      ST1   46.7 Genus_speciesS    0
# \UTF{2139} 1,112 more rows
# \UTF{2139} Use 'print(n = ...)' to see more rows
>
```

- オブジェクトは、数字のみ、あるいは数字が先頭に来る文字以外なら何でも構いません。
- 詳細は後述しますが、`read_excel()` 関数で読み込んだデータは `tibble` という形式です。`tibble` は標準で、データの頭数行のみを表示する仕様となっています。もし、もっと多くのデータを表示させたい場合（例えば先頭から 50 行分）は`> print(treedf, n=50)` と打ってみて下さい。
- これ以降のコードでは、行頭の`>`は省略します。

2.2.3 作業・データの保存

作業内容（読み込んだデータや生成したオブジェクトなど）やコマンドの入力履歴をそのまま保存しておくことが出来ます。作業内容は、「ファイル」→「作業スペースの保存」で`.RData` という拡張子のファイルとして保存されます。`.RData` には、読み込んだデータそのものが含まれています。`.Rhistory` には、入力したコマンドの履歴だけが含まれています。

`.RData` は読み込んだデータそのものや計算結果も保存できるので、一連の解析が終わった後に`.RData` として保存しておくと、結果を振り返るのに便利です。特に、ベイズ統計を使うようになると、モデルやデータ量によっては非常に計算に時間がかかるため、結果を保存しておく重要性が高くなります。ただし、パッケージはその都度読み込む必要があります。

2.2.4 小技

- キーボードの上矢印を押すと一つ前に打ったコマンドが、下矢印を打つと先のコマンドが表示されます。
- `1:10` というように数字の間に`:`をはさむと、左の数字から右の数字まで 1 ずつ変化する数列を生成で

きます（等差数列）。

2.3 まず体験してみる～Rとtidyverseで楽々データ操作～

系統だった使い方の説明の前に、ひとまずRで簡単にできることを紹介したいと思います。上記のようなデータを取った後、みなさんだったら何を知りたいと思いますか？

ちょっと考えただけでも、知りたいことがたくさんあると思います。例を挙げると、以下のような項目です。

- 每木プロットごとに、何個体調査したのか？
- 每木プロットごとに、何種出現したのか？
- 每木プロットごとに各種が何個体出現したのか？
- 每木プロットごとの胸高断面積合計は？

では、これらをRで調べてみましょう。まず、一番最初の「プロットごとの調査個体数」は、以下のように打ち込むことで得られます。

```
treedf %>%
  group_by(Region, Stand) %>%
  reframe(本数=n()) %>%
  print(., n=40)
```

…早速、意味不明な文字列がたくさん出てきて、心が折れてしまった方もいるかも知れません（笑）。でも大丈夫です。1つずつ解説していきます。

2.3.1 パイプライン処理

tidyverseパッケージでは、あるオブジェクトに対する一連の処理を`%>%`という記号でつなげて記述します。これはパイプ演算子と呼ばれます。ある処理を行い、次の処理に渡すことができます。パイプ演算子によって、コードを直感的に記述することが可能になります。

例えば、上記のコードの1行目は、毎木データを付与したオブジェクトに対して処理を行いますよ、ということを宣言していることになります。

2.3.2 カテゴリーごとにデータを分割

2行目は、`group_by`という関数が使われています。これは、中に書かれた変数の「カテゴリーごとにデータを（仮想的に）分割する」ということを意味しています（仮想的、というところがポイントです）。ここでは、RegionとStandごとにデータを分割しました。Rの組み込み関数である`split`と似ています。これだけでは何が起きたのか分かりにくいかも知れません。次の処理に進んでみましょう。

2.3.3 カテゴリーごとに任意の処理を行う

3行目は、`reframe`という関数が使われています。これは、カテゴリーごとに分割されたデータについて、`reframe`中に書かれた処理を行ってその単位にデータを集約する関数です。Rの組み込み関数である`tapply`と似ています。

さて、では処理として何が行われているでしょうか。本数=n()と書かれています。n()は行数を合計するという記法です。それを、「本数」という列として結果に付与するという意味になります。

最後の列は、それを40行表示させるという関数です。これは、先ほど述べたように、tibbleの標準では先頭の数行しか表示しないためです。

2.3.4 プロット毎の出現種数

では、次に出現種数を調べてみましょう。それを実行するコードは、以下のとおりです。

プロット毎の出現種数

```
treedf %>%
  group_by(Region, Stand) %>%
  reframe(種数=length(unique(Species))) %>%
  print(., n=40)
```

先ほどと異なるのは、3行目ですね。reframeでデータを集約する部分が異なっています。Speciesは、データに含まれている種を示す列です。uniqueはその中に入れたデータの中で固有の要素を調べる関数で、lengthはデータ数（ベクトルの長さ）を調べる関数です。プロットごとに、出現した種の一覧をuniqueで作成し、その一覧の長さ（数といつてもいいと思います）をlengthで評価して、プロットごとの種数を得ることができました。

Rでは、このように多種多様な関数が出てきます。その全てを覚えることは困難ですし、その必要もありません。よく使う関数だけ覚えておいて、それ以外は「こういった処理ができる関数は？」とChatGPTに聞けば、答えが返ってきます。

2.3.5 プロットかつ種ごとの出現個体数

直前の例では「種数」のみを調べました。つまり、種の単位でみると「在不在」だけを調べたことになります。しかし、種ごとの個体数を知りたいこともあります。

上で、データを（仮想的に）分割する関数としてgroup_byを学びました。これまでの例ではRegionかつStand単位でしたが、ここに「種（Species）」を加えれば良さそうです。では、以下のコードの空白部分に何に入るか、考えてみましょう。

演習 1: プロットかつ種ごとの出現個体数の計算

```
treedf %>%
  group_by(Region, Stand, _____) %>%
  reframe(個体数=___)
```

正しく指定ができていれば、以下のように表示されるはずです。

```
# A tibble: 594 × 4
  Region Stand Species      個体数
  <chr>   <chr>  <chr>        <int>
  1 A       ST1    Genus_speciesA     2
```

```
2 A      ST1   Genus_speciesB      3  
(以下、省略)
```

さて、目的は達成されましたが、人間にとっては視覚的にわかりにくいです。感覚的な言葉で言うと、データが縦方向に並んでいます。このデータを、行が Species、列が Region ごとの Stand という行列になっていたらもう少しあかりやすくなると思います（このことを、横展開すると言います）。

R では、このような「データの並び方の変換」も簡単です。以下で、上記の結果を行列の形に変換してみましょう。

```
treedf %>%  
  group_by(Region, Stand, Species) %>%  
  reframe(個体数=n()) %>%  
  pivot_wider(., id_cols="Species", names_from=c("Region", "Stand"),  
             values_from="個体数", values_fill=0)  
# A tibble: 20 × 41  
  Species     A_ST1 A_ST2 B_ST1 B_ST2 C_ST1 C_ST2 D_ST1 D_ST2 E_ST1 E_ST2 F_ST1 F_ST2 G_ST1 G_ST2  
  <chr>      <int>  
1 Genus_spec…     2     1     4     1     0     1     2     0     3     1     1     0     1     0  
2 Genus_spec…     3     2     2     1     2     0     4     2     3     3     2     1     1     0  
(以下、省略)
```

行列に展開することができました。ここで新しく、pivot_wider という関数が出てきました。名前の通り、データを横方向に展開するための関数です。引数（ひきすう）の指定を説明します。

id_cols 横展開した時の「行」に該当する要素。複数の行で同じ値があってはいけない。

names_from 横展開した時に「列」に該当する要素。複数指定が可能。

values_from 横展開した行列に入る「値」。

values_fill 横展開した場合に、行と列の組み合わせが存在しないデータについて埋める値。

2.3.6 結果の出力

上記の例では、結果が R 上に表示されました。ですが、これを論文や報告書などに貼りたいこともあると思います。R で操作した結果は、何かしら新しいオブジェクトとして、それを様々な形式で出力できます。ここでは、.xlsx 形式のファイルとして出力します。

```
# sptable というオブジェクトに結果を付与する  
sptable <- treedf %>%  
  group_by(Region, Stand, Species) %>%  
  reframe(個体数=n()) %>%  
  pivot_wider(., id_cols="Species", names_from=c("Region", "Stand"),  
             values_from="個体数", values_fill=0)  
# .xlsx 形式で出力するためのパッケージの読み込み  
library(writexl)
```

```
# 出力  
write_xlsx(sptable, "種_林分行列.xlsx")
```

出力したファイルは、起点ディレクトリに保存されています（`here` 関数を使えば起点フォルダ内の任意の場所に保存できます）。他には、csv ファイル形式で出力する `write.csv()` という関数もあります。

2.3.7 每木プロットごとの胸高断面積合計

胸高断面積合計は、林分の蓄積量を表現する上でよく使われる指標です。毎木で出現した個体ごとに、胸高断面積を計算し、林分単位で合計することで計算できます。ただし、毎木の調査面積で値の意味が違ってきますので、1 ha あたりの平方メートル (m^2) で表現されることが多いです。上で説明したように、毎木調査の1箇所あたりの面積は $20m \times 20m$ という設定です。また、個体の DBH (胸高直径) は、cm 単位で記録されています。

これらの点に留意し、プロットごとの胸高断面積合計を計算してみましょう。これまでと異なる部分もありますので、段階を踏んで説明します。まず、個体ごとに断面積（ただし、平方メートル単位）を計算してみましょう。

演習 2: 個体ごとの断面積の計算

```
treedf %>%  
  mutate(断面積=3.14*(_____/100/2)^2)
```

また新しい関数が出てきました。`mutate` は、tibble に「新しい列」を加える関数で、多用します。樹木の断面積を円と仮定すると、円の面積の公式に従って面積を計算することができます。ただし、のちに平方メートル単位の断面積合計を得るので、ここで平方メートル単位に変換してしまいます。上記のコードの欠けている部分を埋めて、計算しましょう。

個体ごとの断面積が計算できたら、それを Region かつ Stand 単位で合計し、ha あたりに変換すれば、断面積合計 (m^2/ha) が計算できます。計算するコードは、以下のとおりです。

演習 3: 每木プロットごとの胸高断面積合計の計算

```
treedf %>%  
  mutate(断面積=3.14*(DBH/100/2)^2) %>%  
  group_by(_____, _____) %>%  
  _____(断面積合計=sum(_____)/(20*20)*100*100)
```

ここで、`sum` という新しい関数が出てきました。これは名前から推察できるかもしれません、合計値を計算してくれる関数です。正しく指定できれば、以下のような出力が得られます。

```
# A tibble: 40 × 3  
  Region Stand 断面積合計  
    <chr>   <chr>     <dbl>  
1 A       ST1      75.8  
2 A       ST2      63.2
```

(以下、省略)

調査データの概要を数字で掴むための操作を、Rでやってみました。これらのこととは、Excelでもできるしそちらの方が簡単だと言うご意見もあるかもしれません。しかし、Rで実行すると、調査地が幾つあろうとも同じ書き方で表現できること、この後行う作図とも連携することができます。コマンドを書くことに慣れていないと非常に面倒なことをしているように感じるかもしれませんのが、慣れるととても便利です。

2.4 まず体験してみる 2～R と ggplot で楽々作図～

上記のように数字で情報を取り出すことも重要ですが、視覚的にデータを確認することも重要です。毎木データを収集した時に、どんな図を描きたいでしょうか？

Rで作図する利点として、同じような図をコマンドで多数生成できることが挙げられます。以下では、プロット後のサイズ(DBH)構造を図にしてみます。個別の関数の説明は後述しますので、まずは以下のコードを打ち込んでみてください。

プロット毎のサイズ構造

```
p1 <- ggplot(data=treedf, aes(x=DBH))+  
  geom_histogram() +  
  facet_wrap(~ Region + Stand,  
             labeller = labeller(.cols = label_value, .multi_line = FALSE), ncol=6)+  
  theme_bw()  
  
p1
```

複数の同じ形式の図を、一瞬で描画することができました。しかも、x軸やy軸のラベルが、外側に位置する図にだけ描画されています（もちろん細かいことを言い出すとまだまだ直すところはあるのですが、それは後ほど）。少し、Rが便利だと思えてきたのではないでしょうか？

作成した図は、様々な形式で保存することが可能です。ここでは、PDFで出力する例を示します。

```
ggsave("DBH_hist.pdf", device=cairo_pdf)
```

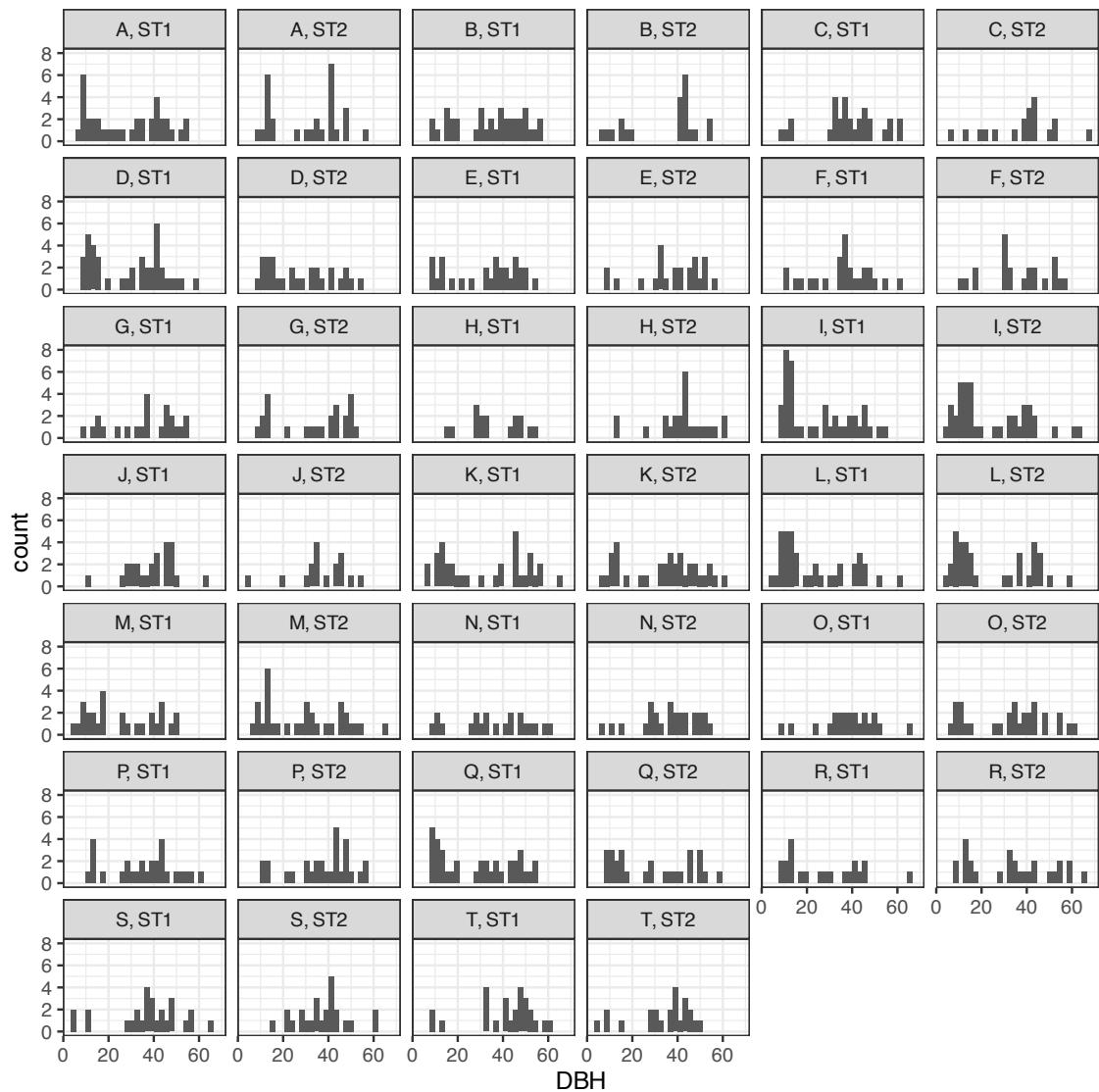


図2 プロットごとのサイズ構造

3 データ操作

以下では、ティブル形式を中心にデータの操作方法について解説します。

3.1 部分的な選択や削除

ティブルはすでに説明したように、行列という2次元の形式のデータです。この一部を取り出したり、逆に削除したりすることはよくあります。そのためのいくつかの方法を紹介します。

3.1.1 ティブルの特定の部分を選択

必要な列を指定

```
# Region、Stand、Species 列を指定  
treedf %>%  
  select(Region, Stand, Species)  
# 左からの列の位置を数字で指定することもできる  
treedf %>%  
  select(c(1:2, 4))
```

c()は、その中に書いた要素をつなげる（ベクトル化する）ことができます。

なお、この操作ではあくまでティブルという行列形式を維持して部分的にデータを抽出しましたが、そうではなく構造のないデータの塊（ベクトル）として取り出したい時は、Rの基本的な記法を使います。

特定の列の抽出

```
# オブジェクト$列名という記法  
treedf$DBH  
[1] 35.469068 40.142572 42.157646 41.888702 39.498515 25.045804 43.678378  
[8] 45.171440 35.156645 46.748556 32.375514 43.860738 33.359967 22.756558  
(以下、省略)
```

不要な列を指定

```
# Region、Stand、Species 列を残すために  
# DBH と Debark 列を不要とする  
treedf %>%  
  select(-c("DBH", "Debark"))  
# 数字でも同じことができる  
treedf %>%  
  select(-c(3,5))
```

必要な行を指定

```
# DBH が 30 より大きいデータだけを指定  
treedf %>%  
  filter(DBH > 30)  
# Species が Genus_speciesA だけを指定  
treedf %>%  
  filter(Species=="Genus_speciesA")  
# 複数条件も指定可能  
treedf %>%  
  filter(Debark==1&Species=="Genus_speciesG")  
# 文字列の複数条件を指定する場合は str_detect 関数を使うと便利  
treedf %>%  
  filter(str_detect(Species, "B|F")) # 部分一致
```

不要な行を指定

```
# Species が Genus_speciesC 以外のデータを得る  
treedf %>%  
  filter(Species!="Genus_speciesC")  
# 複数条件ももちろん可能  
# Species が B および F を除く  
treedf %>%  
  filter(!str_detect(Species, "B|F"))
```

このように部分的な選択や削除を行う際、データの種類や型を意識する必要があります。また、条件を指定するための二項演算子も知っておく必要があります。以下にまとめます。

3.1.2 R におけるデータの種類

単一の値の種類

名称	R での表記	変更方法	例
空値	NA		NA
論理値	logical	as.logical()	TRUE
整数	numeric	as.numeric()	5.3
文字列	character	as.character()	"Tekito"
要因	factor	as.factor()	

NA はしばしば他の関数を無効化してしまうので注意が必要です。また、要因は外見は文字列とほぼ同じですが、要因は明確なカテゴリーとして扱われます。外見が数字でも文字列として扱われているケースがあるので注意が必要です。

値の集まりの構造

名称	R での表記	変更方法
ベクトル	c	c()
行列	matrix	matrix() または as.matrix()
データフレーム	data.frame	data.frame() または as.data.frame()
ティブル	tibble	tibble() または as_tibble()
リスト	list	list() または as.list()

ベクトル 最も基本の単位。一つながりのデータ。あらゆるデータを含めるが、同一ベクトル内で異なった形式の値を混在させることはできない。異なった形式の値を混在させると、一番可塑性が高い形式に変換されてしまう。例えば、

```
test <- 1:4
test[1] + test[2] #test の 1 番目と 2 番目の要素を足しなさい
[1] 3
test2 <- c(1, 2, "Tekito", TRUE)
test2
[1] "1"      "2"      "Tekito" "TRUE"
test2[1] + test2[2] #test2 の 1 番目と 2 番目の要素を足しなさい
Error in test2[1] + test2[2] : non-numeric argument to binary operator
```

となり、文字列が入っている場合は数字も文字列扱いになってしまう。

行列 ベクトルがいくつも集まつたもの。numeric 以外は含めない。

データフレーム ベクトルの集まり。R にデータを読ませるときによく使われる形式だが、最近 tibble への移行が進んでいる。行や列のラベルあるいは番号で行や列を取り出せる。行数の違う列が混在することはできない。行列でいろんなデータを扱えるようにしたもの、と考えられる。

ティブル 上記のように、データフレームとほぼ同じだが、最近はティブルが使われることが多い。標準ではデータの先頭だけを表示させる、group が扱えるなどの特徴がある。

リスト 行数の違うデータだろうがなんだろうが無理やり 1 つのオブジェクトにしてしまう。データの長さや形式がごちゃごちゃなものをとりあえず一つのオブジェクトにまとめたいときに有用。

とりあえず Excel などに打ち込んだデータはデータフレームまたはティブルとして扱うことがほとんどで、これが基本形です。

3.1.3 二項演算子

3.2 演算関数

R では、演算のための関数がいくつか紹介されています。

mean 平均値を計算する。例：mean(treedf\$DBH)

sd 標準偏差を計算する

sum 合計値を計算する。

意味	R での表記
等しい	<code>==</code>
等しくない	<code>!=</code>
a より大きい	<code>> a</code>
a 以上	<code>>= a</code>
a 未満	<code>< a</code>
a 以下	<code><= a</code>
a または b	<code>a b</code>
aかつb	<code>a & b</code>

`max` 最大値を計算する。
`min` 最小値を計算する。
`length` 要素数（データの数）を計算する。例：`length(treedf$DBH)`
`unique` データ中の固有の要素を表示する。例：`unique(treedf$Species)`
`summary` ティブルやデータフレームに適用し、列毎の要約統計量を計算する。例：`summary(treedf)`

3.3 列の追加

これは非常によく使います。すでに説明したように、ティブルに新しい列を追加するときには `mutate` を使います。以下では、DBH から周囲長を計算して、新しい列として加えます。

新しい列の追加

```
treedf <- treedf %>%
  mutate(GBH=DBH*pi)
```

なお、R の基本関数による列の追加は、`treedf$GBH <- treedf$DBH*pi` のように、**オブジェクト\$列名 <-** という形で実行できます。

では、DBH から断面積を計算してみましょう。断面積は、DBH の半分（半径）を 2 乗し、円周率を乗じることで計算できます。以下の空白部分を埋めてみましょう。

演習 4: 断面積を計算して新しい列として付与する

```
treedf <- treedf %>%
  _____(断面積=(_____/2)^{2}*pi)
```

3.4 カテゴリーごとの操作

データに含まれる何かしらのカテゴリー毎にデータを操作する場合、すでに紹介した `group_by` 関数を使います。カテゴリー毎に擬似的にデータを分割し、`reframe` 関数または `mutate` 関数で処理を行います。これも非常に多用します。

処理を行う関数ですが、`group_by` で指定する単位でデータを集約する（元のデータと形が変わる）場合は `reframe`、`group_by` 単位で何かしらの処理を行うがそれを元々のデータに付与する（元のデータと形は変わらず、列が追加されるだけ）場合は `mutate` を使います。と、言葉で書いても分かりにくいと思うので、実例を通して学びます。まず、`reframe` を使う場合です。種ごとの剥皮率を計算します。

種毎の剥皮率の計算

```
treedf %>%
  group_by(Species) %>%
  reframe(剥皮率=mean(Debark))
```

今回は結果を表示させるだけですが、もしこの計算結果を保存したければ、上記のコードの先頭に **保存するオブジェクトの名前 <- treedf** と記述することで保存できます。

使い方に慣れるために、似たような例題をやってみましょう。以下の空白を埋めて下さい。

演習 5: 種ごとの平均 DBH を計算する

```
treedf %>%
  group_by(Species) %>%
  ____(平均 DBH=____(DBH))
```

次は、`mutate` を使う場合です。毎木プロットごとに、プロットの断面積合計に対する各個体の断面積の相対的な優占度を計算してみましょう。

プロットごとの各個体の相対優占度

```
treedf %>%
  group_by(Region, Stand) %>%
  mutate(Relative_dominance=断面積/sum(断面積)) %>%
  # 以下は、単に表示のため、本来は不要
  select(断面積, Relative_dominance)
```

結果を見ていただくとわかるように、`mutate` を用いた場合は元のデータに `mutate` 内で計算した結果が付与されています。ただし、上記の例では `sum(Area)` によって Region かつ Stand の合計が計算されており、`group_by` が作用していることがわかります。

このような `group_by` 関数の性質を利用することで、例えば「プロット内で自分より DBH が大きい個体の DBH の合計を計算する」ということも可能です。

自分より大きい個体の DBH を合計する

```
treedf2 <- treedf %>%
  group_by(Region, Stand) %>%
  mutate(Supression_index=map_dbl(DBH, ~sum(DBH[DBH>.]))) %>%
  # 以下は、単に表示のため、本来は不要
  select(断面積, Supression_index)
```

`map_dbl` は行ごとに何かしらの関数を適用し、その結果を数字のベクタとして返す関数です。一般に、~以下

で書かれた部分が関数とみなされます。また、関数中で. は自分、つまり与えるデータを示す記号です。

3.5 文字列操作

データとして含まれている文字列を、データ処理や作図の過程で変更したいことが生じると思います。ここでは、文字列の操作法について説明します。

3.5.1 文字列の結合

以下では、Region と Stand を結合して Plotid という新しい文字列を作ります。str_c 関数を使います。

文字列の結合

```
Plotid <- str_c(treedf$Region, treedf$Stand)
# 結合する文字列の間に、任意の文字列を挿入することも可能
Plotid2 <- str_c(treedf$Region, treedf$Stand, sep="_")
```

では、使い方に慣れるために、Region と Stand を、_で結合した Plotid という列を、treedf オブジェクトに新しい列として付与する以下の演習を行なってください。

演習 6: Region と Stand の結合

```
treedf %>%
----(Plotid=----(Region, Stand, sep="_"))
```

3.5.2 文字列の置換

今回のデータで種名は Genus_species* という表記にしてあります。ですが、論文の表などで記載する場合、_は不要でしょう。図の場合は、最後の一文字だけがあれば十分かも知れません。

このように、特定のパターンの文字列を探して置換する関数として、str_replace があります。

文字列の置換

```
treedf %>%
  mutate(Sp2=str_replace(Species, "_", " "))
# A tibble: 1,122 × 6
  Region Stand    DBH Species        De bark Sp2
  <chr>   <chr> <dbl> <chr>        <dbl> <chr>
  1 A       ST1     35.5 Genus_speciesM     0 Genus speciesM
  2 A       ST1     40.1 Genus_species0     0 Genus species0
  (以下、省略)
```

str_replace(検索する列, 検索する文字列, 置換する文字列) という書き方をします。では、str_replace に慣るために、以下の演習を行なってください。先ほどは Species 列の各データから_を除きました。今度は、Genus_species 部分を除き、Sp2 という新しい列として treedf オブジェクトに付与してください。

演習 7: 文字列の置換

```
treedf %>%
  ----(Sp2=str_replace(Species, "-----", ""))
# 以下は、単に表示のためで、本来は不要
select(Species, Sp2)
```

これまで、固定された文字列を置換しました。もう少し柔軟な文字列の抽出も可能です。以下では、正規表現を使って、「先頭から_以外の部分」を抽出します。

曖昧な条件指定による抽出

```
treedf %>%
  mutate(Sp2=str_extract(Species, "^[^_]+"))
# 以下は、単に表示のためで、本来は不要
select(Species, Sp2)
```

3.5.3 正規表現

効率よく検索や抽出を行うためには、正規表現を使いこなす必要があります。正規表現とは、完全一致とは異なり「～のような条件に従う」という、より曖昧な表現です。

正規表現を学ぶために、`char1 <- "山梨森林管理事務所, やまなししんりんかんりじむしょ, yamanashi, 1530_は-01"`という素敵な文字列から必要な部分を抽出してみましょう。

文字の抽出

```
str_extract(char1, "\\w")
[1] "山"
```

文字列の抽出においては、必ず「左から見て最初にマッチした部分」が結果として返されるという原則があることに注意して下さい。`\w` は、特殊記号でない文字を意味します。そのため、一番最初の文字である「山」が抽出されました。

では次に、文字が続いている部分を抽出してみましょう。

一連の文字の抽出

```
str_extract(char1, "\\w+")
[1] "山梨森林管理事務所"
```

先ほどと異なるのは、末尾に `+` が付いただけです。これは、一つ以上の要素が続くことを意味します。このため、文字が続いている範囲全てがマッチするので「山梨森林管理事務所」という文字列を抽出することができました。`,` は特殊文字なので、結果に含まれていません。

上記では種類を問わず文字を検索しましたが、特定の種類の文字を検索することも可能です。以下では、ひらがなを抽出してみましょう。

一連の文字の抽出

```
str_extract(char1, "\\p{Script=Hiragana}")  
[1] "や"
```

\\p{Script=Hiragana}は、ひらがなを意味します。では、一連のひらがなを抽出してみましょう。

演習 8: 一連のひらがなの抽出

```
str_extract(char1, "\\p{Script=Hiragana}__")
```

では、もう少し高度な抽出を行ってみましょう。上記の文字列は、森林の小班を表しています（値は架空のものです）。この文字列から、例えば林班の番号だけ、小班の名前だけ、小班の枝番だけを抽出したいことがあると思います。しかし、上記の文字列では、区切り文字に一貫性がなく、左から見て一番最初にマッチするという単純な検索では抽出が難しいです。

区切り文字を認識させれば良さそうですが、ここで問題となるのが、「区切り文字は検索する目印としては使いたいが、抽出結果には含めたくない」ということがあります。これを可能にする正規表現があります。以下では例として、「_の前の数字」を検索します。

肯定の先読み

```
str_extract(char1, "\\d{1,9}(?=\\_)")  
[1] "1530"
```

(?=)は、「肯定の先読み」と呼ばれる正規表現です。この表現が書かれる直前から見て後ろ（右側）に、=以下に書かれた文字列についてその文字列があることを認識するがその文字列は結果に含まれないという処理を行えます。上記の文字列では_は「1530_」の箇所でしか出てきません。このため、_の直前に指定した\\d{1,9}(1~9桁の数字)の直後にある_を認識するが、結果には含まれていません。

今度は、後ろに続く文字列の直前に任意の文字列があるかを確認する、「肯定の後読み」を行ってみます。

肯定の後読み

```
str_extract(char1, "(?<=\\_)\\\p{Script=Hiragana}")  
[1] "は"
```

上記の例で見た通り、正規表現は何かしらの条件を記号で指定します。すべての正規表現をここで紹介することはできませんが、いくつか代表的な表現を紹介します。正規表現は最初はとっつきにくいかもしれませんがあ、使いこなせると文字列を非常に柔軟に取り出すことが可能になります。

\\w (特殊文字以外の) 文字

[a-zA-Z] 半角アルファベット文字

\\d 数字

\\d{1,4} 1桁から4桁までの数字

\\p{Script=Han} 漢字

\\p{Script=Hiragana} ひらがな

\\p{Script=Katakana} カタカナ

\\s 半角スペース

\b 文字列の区切り部分（半角スペースまたは_）
^ (文字列を左から見た時の) 先頭
\ エスケープ記号（特殊文字を検索する時に、この記号と組み合わせて表記すると検索可能になる）
[^...] ...以外の要素
+ 直前の要素が 1 つ以上続く
(?=...) 特定のパターンの後に...の条件が存在することを確認するが、その条件はマッチング結果に含まれない（肯定の先読み）。
(?<=...) 特定のパターンの前に...の条件が存在することを確認するが、その条件はマッチング結果に含まれない（肯定の後読み）。

3.5.4 大文字と小文字の変換

R は大文字と小文字を区別します。これは有用な時もありますが、人間側があまり意識せず大文字と小文字を混在させてしまい、データを結合するときなどに混乱が生じることがあります。そんなときは、以下で紹介するように R 上で大文字や小文字に変換することができます。

全て大文字にする —

```
treedf %>%  
  mutate(Sp4=str_to_upper(Species)) %>%  
  # 以下は、単に表示のためで、本来は不要  
  select(Species, Sp4)
```

全て小文字にする —

```
treedf %>%  
  mutate(Sp4=str_to_lower(Species))
```

先頭だけ大文字にする —

```
treedf %>%  
  mutate(Sp4=str_to_title(Species))
```

3.5.5 日本語の扱い

昔から R を使っている人は、R ではデータに日本語が含まれるとうまく扱えないというイメージを持っているかもしれません。しかし、read_excel 関数で Excel ファイルを直接読めるようになり、文字化けの問題はかなり解消しました。また、日本語を扱える関数も充実してきました。そのため現在では、R で日本語が含まれるファイルを積極的に編集することが可能です。

以下では、全角と半角文字の変換、日本語文字のローマ字化について説明します。自分でデータを打ち込む場合に両者が混在することはないと想いますが、行政データの婆は全角と半角で作成していることがあり、くらくらした経験がある方もいるのではないでしょうか？なお、以下で扱う関数は stringi パッケージに含まれていますので、最初にパッケージを読み込みます。

半角から全角

```
J_text <- c("ニホンジカ", "ニホンジカ", "ハウチワカエデ", "ハウチワカエデ")  
library(stringi)  
stri_trans_general(J_text, "Halfwidth-Fullwidth")
```

`stri_trans_general` 関数を使い、引数に `Halfwidth-Fullwidth` を指定することで、半角文字を全角文字に変換できます。これで原理はおおよそわかったと思いますので、全角から半角文字への変換は演習問題にしたいと思います。

演習 9: 全角から半角

```
stri_trans_general(J_text, "----width----width")
```

このように日本語も使えますが、関数によってどうしても英数字しか扱えない状況もあると思います。そのような場合、日本語を半角のアルファベット（いわゆるローマ字）に変換することも可能です。

日本語をローマ字にする

```
stri_trans_general(J_text, "Latin")
```

3.5.6 条件分岐

与えられた文字列について、～なら～に変換するといった操作も可能です。例えば、DBH が 50 より大きい個体を林冠を構成する個体という意味で Canopy、それ以下のサイズの個体をその他（Other）と分類してみます。`if_else` 関数を使います。

1 条件の条件分岐

```
treedf %>%  
  mutate(Layer=if_else(DBH>50, "Canopy", "Others")) %>%  
  print(., n=100)  
# if_else(条件式, TRUE の場合, FALSE の場合)
```

条件分岐については、条件が 2 つ以上でも可能です。ここでは、なぜかニホンジカによる植物の嗜好性がわかっているとして、「嗜好性種」、「普通種」、「不嗜好性種」に再区分した列を追加します。`case_when` 関数を使います。

2 条件以上の条件分岐

```
treedf %>%
  mutate(Sptype=case_when(Species=="Genus_speciesA" ~ "Palatable",
                          Species=="Genus_speciesB" ~ "Unpalatable",
                          .default="Normal")) %>%
  print(n=100)
# case_when(条件式 1 ~ 条件にマッチした場合,
#           条件式 2 ~ 条件にマッチした場合,
#           .default ~ 条件式を設定しない場合
#           ....)
```

一般に文字列を条件分岐させる場合、指定する条件が多くなりがちです。`case_when` 関数ではそのような状況も想定し、`.default` という表現があります。上記の例のように、個別に指定しない場合は全て`.default`で指定した結果が返されます。

3.5.7 NA（欠損値）の扱い方

空欄（欠損値）のことを、R では **NA** と表記します。こいつが少々曲者です。NA を含むティブルやベクトルに上記の関数を適用しても、ほとんどは **NA** が返ってきます（つまり実行できない！）。しかし、欠測値が生じることは、実データでは避けられません。そのため、NA の扱い方を知ることは重要です。

NA を含むデータを作って、挙動を見てみましょう。最初の方で、プロットかつ種単位の出現個体数の行節を生成した際、`pivot_wider` の引数で存在しない組み合わせの箇所を 0 で埋める `fill=0` を指定しましたが、それを指定しなければ NA を含んだデータができます。

```
treedfNA <- treedf %>%
  group_by(Region, Stand, Species) %>%
  reframe(個体数=n()) %>%
  pivot_wider(., id_cols="Species", names_from=c("Region", "Stand"),
             values_from="個体数")
summary(treedfNA)
# NA データが含まれている
  Species          A_ST1          A_ST2
Length:20      Min.   :1.000    Min.   :1.00
Class :character 1st Qu.:1.000   1st Qu.:1.00
Mode  :character Median :2.000   Median :1.50
                  Mean   :2.176   Mean   :1.75
                  3rd Qu.:3.000   3rd Qu.:2.00
                  Max.   :4.000   Max.   :4.00
                  NA's    :3       NA's    :4
```

(以下、省略)

```
mean(treedfNA$A_ST1)
[1] NA
```

このように、NA が返ってきてしまいます。NAへの対処法としては、以下の方法があります。

- 関数の引数で対応
- NA をデータから除く
- NA を 0 に変換する（問題ない場合は）

関数によっては、引数で NA を除いて動作してくれる関数があります。先ほどの `mean()` も実はそうです。

```
mean(treedfNA$A_ST1, na.rm=TRUE)
[1] 2.176471
```

ただし、関数によってこのような引数が使える場合と使えない場合がありますので、ご自身で使いたい関数の動作を調べてください。

NA は関数の動作を妨げるだけでなく、二項演算子などの動作も妨げます。NA を指定して除くためには、NA かどうかを判定する専門の関数 `is.na()` を使う必要があります。先ほど生成した NA を含むデータは多くの列に NA が含まれていますが、仮に A_ST1 列について NA がない部分に限定するコードを示します。

```
treedfNA2 <- treedfNA %>%
  filter(!is.na(A_ST1))
summary(treedfNA2)

  Species          A_ST1          A_ST2
Length:17      Min.   :1.000    Min.   :1.000
Class :character 1st Qu.:1.000  1st Qu.:1.000
Mode  :character Median :2.000  Median :2.000
                  Mean   :2.176  Mean   :1.857
                  3rd Qu.:3.000  3rd Qu.:2.000
                  Max.   :4.000  Max.   :4.000
                  NA's    :3
```

(A_ST1 列には NA が含まれていない)

欠測であることが 0 と等価と考えて差し支えない場合（例えば、今回は生存しているかどうかに欠測があるが、これは発見できなかった場合は死亡 0 と考えて差し支えない）は、0 に変換するのも手です。

```
treedfNA3 <- treedfNA %>%
  mutate(A_ST1 = if_else(is.na(A_ST1), 0, A_ST1))
# ifelse(条件式, TRUE の場合, FALSE の場合)
summary(treedfNA3)

  Species          A_ST1          A_ST2          B_ST1
Length:20      Min.   :0.00    Min.   :1.00    Min.   :1.0
Class :character 1st Qu.:1.00  1st Qu.:1.00  1st Qu.:1.5
Mode  :character Median :2.00  Median :1.50  Median :2.0
                  Mean   :1.85  Mean   :1.75  Mean   :2.4
```

```
3rd Qu.:2.25   3rd Qu.:2.00   3rd Qu.:3.0  
Max.     :4.00   Max.     :4.00   Max.     :5.0  
NA's      :4       NA's      :5
```

(A_ST1 から NA が消えている。かつ、treedfNA2 と異なり
NA を 0 にしているので、平均値などが異なっている点に注意)

例で示したように、`if_else()` 関数は、最初に条件式を記述し、次に条件式が TRUE の場合、最後に条件式が FALSE の場合の動作を記述します。

3.6 パイプライン処理の利点

ここまで、基本的には tidyverse によるパイプライン処理でコードを記述してきました。これに対し、R の基本関数による記法も存在します。両者は対立するものではなく、どちらの記法も知っておく必要があります。ただし、一般にパイプライン処理の方が、コードの可読性が高いため、飯島は可能な限りパイプライン処理をすることをお勧めしています。

両者の違いを明確にするため、同じ処理をそれぞれの記法で記述します。題材として、毎木データから Region かつ Stand ごとの胸高断面積合計を計算し、それを毎木データに付与するという作業を行います。

—パイプライン処理—

```
treedf1 <- treedf %>%  
  group_by(Region, Stand) %>%  
  mutate(plotBA=sum(3.14*(DBH/100/2)^2)/(20*20)*100*100)
```

同じ内容を、R の基本関数で書くとこうなります（基本関数の説明はしないので、これまでに説明していない関数を使っています）。

—基本関数—

```
plotBA <- as.data.frame(ftable(tapply(3.14*(treedf$DBH/100/2)^2/(20*20)*100*100,  
  treedf[, c("Region", "Stand")], sum)))  
colnames(plotBA) <- c("Region", "Stand", "plotBA")  
treedf2 <- merge(treedf, plotBA, by=c("Region", "Stand"))
```

行数としては同じ 3 行ですが、基本関数はデータの形を整えるために様々な関数を組み合わせる必要があります。また、1 行 1 行ごとに異なる処理をしており、元々のデータとの関係が見えにくいです。

3.7 繰り返し動作

繰り返し命令は、様々な場面で使える関数です。R で繰り返し命令は、`for` 関数を使います。

3.7.1 `for()` の基本

`for()` は、基本的に以下のように記述します。

for() の使い方

```
for (i in 1:N) {  
  i  
}
```

1:N はすでに学んだように、等差数列 (1, 2, ..., N) です。この数字が、i という文字列に逐次代入されます。つまり、まず i に 1 が代入され、括弧内で指定された動作が終了すると、次に i に 2 が代入され、括弧内で指定された動作が行われます。そして、i に代入する文字列がなくなるまでこの動作が続けられます。代入するのは数字である必要はなく、また代入位置を示す文字列が i である必要もありません。

3.7.2 多数のファイルの読み込み

for() が力を発揮するのは、同じような動作を何度も繰り返すときです。今回用いるデータの一つに、自動撮影カメラのデータがあります。自動撮影カメラによる撮影枚数が多いほど、対象とする動物が多いと考えられます。ですが、自動撮影カメラのデータは、通常カメラの SD カードごとに管理されます。そのため、カメラが複数ある場合、データのファイルも複数あることが普通です。このように、同じような形式のデータファイルが多数ある場合に、それらをまとめて処理するのに便利なのが for です。

ですが、いくら繰り返し命令が使えるとしても、ファイル名を個別に指定することは煩雑です。もし自動撮影カメラのファイルが特定のフォルダに格納されていれば、そのフォルダ以下にあるファイル名を、ファイル名の特徴（例えば拡張子）でまとめて取得することができます。

ファイル名による検索

```
cam_files <- list.files("camera フォルダまでのパス",  
  recursive=TRUE, pattern="*.xlsx")  
  
# 飯島の場合  
cam_files <- list.files(here("data", "camera"),  
  recursive=TRUE, pattern="*.xlsx")
```

本当にファイル名を取得できたのか、cam_files と打ち込んで確認してみましょう。

```
cam_files  
[1] "A_1.xlsx" "A_2.xlsx" "A_3.xlsx" "A_4.xlsx" "A_5.xlsx"  
[6] "B_1.xlsx" "B_2.xlsx" "B_3.xlsx" "B_4.xlsx" "B_5.xlsx"  
(以下、省略)
```

どうやら、ファイルは 100 個あるようです。これらを、繰り返し命令を使わないので読み込もうとすると、以下のようになります。

```
cam_df <- read_excel(here("data", "camera", cam_files[1]), sheet="Sheet 1")  
cam_df <- cam_df %>%  
  bind_rows(., read_excel(here("data", "camera", cam_files[2]), sheet="Sheet 1"))  
cam_df <- cam_df %>%  
  bind_rows(., read_excel(here("data", "camera", cam_files[3]), sheet="Sheet 1"))  
...  
30
```

```
cam_df <- cam_df %>%
  bind_rows(., read_excel("data", "camera", cam_files[100]), sheet="Sheet 1"))
```

上記の例では、`bind_rows` という新しい関数が出てきました。これは、ティブル同士を縦方向で結合する関数です。似たような関数として `rbind` がありますが、`rbind` は結合するティブルの列名が完全に一致している必要があるのに対し、`bind_rows` は一致していないなくても構いません。

上記のように、100 行もコマンドを書くのは煩雑だと思います。そこで、`for` の出番です。

複数ファイルの読み込みと結合

```
cam_df <- read_excel(here("data", "camera", cam_files[1]),
  sheet="Sheet 1")
for (i in 2:length(cam_files)) {
  cam_df <- cam_df %>%
    bind_rows(., read_excel(here("data", "camera", cam_files[i]),
      sheet="Sheet 1"))
}
# 警告が出ますが、気にしなくて大丈夫です。
```

データが本当に全て結合できたのか、`cam_df` および `tail(cam_df)` と打ち込んで、データの最初と最後を見てみましょう。

```
cam_df
# A tibble: 892,900 × 4
  cam_id     ymd           count species
  <chr>   <dttm>       <dbl> <chr>
1 A_1     2024-05-09 10:00:00     0 <NA>
2 A_1     2024-05-09 10:05:00     0 <NA>
(途中省略)
```

ご覧いただければわかるように、自動撮影カメラデータは以下の構造となっています。

```
cam_id カメラの ID
ymd 撮影日時
count 撮影個体数
species 撮影された種
```

3.8 ティブル同士の結合

3.8.1 縦または横方向で結合

これは先ほどの繰り返し命令のところで出てきたように、`bind_rows` や `bind_cols` を使います。

3.8.2 2つのデータに共通のデータを目印に結合

複数のデータを結合する場合、両者に共通な情報をもとに結合することも可能です。そのための関数として、`*_join` という一連の関数が用意されています。これも多用します。

例えば、すでに読み込んでいる毎木データは測定した1本1本の木が基本単位です。一方、Region や Stand 単位で得られる情報もあると思います。以下では、データとして持っている Region の情報を毎木データと結合することで、`*_join` 関数の使い方を学びます。まず、Region 単位のデータを読み込みます。

```
regioninfo <- read_excel(here("data", "data.xlsx"), sheet="Region")
regioninfo
# A tibble: 20 × 2
  Region 地域
  <chr>  <chr>
1 A       白州
2 B       八ヶ岳
# ... (以下、省略)
```

どうやら、Region の日本語地名が入っているようです。では、これを毎木データと結合します。

2つの異なるデータを共通の列を目印に結合する

```
treedf <- treedf %>%
  inner_join(., regioninfo, by=c("Region"))
treedf
# A tibble: 1,122 × 8
  Region Stand   DBH Species      Debark    GBH 断面積 地域
  <chr>  <chr> <dbl> <chr>        <dbl> <dbl> <dbl> <chr>
1 A       ST1     35.5 Genus_speciesM     0 111.    988. 白州
2 A       ST1     40.1 Genus_speciesO     0 126.    1266. 白州
# ... (以下、省略)
```

ご覧いただければわかるように、確かに「地域」という列が追加できているようです。

ここでは、結合する2つのデータに共通する部分のみ結合しました。`****_join` 関数はいくつか種類があり、以下のような結合も可能です。

- `inner_join(データ1, データ2, by=両者に共通する列)` 結合する2つのデータが一致する箇所のみ結合する
- `left_join(データ1, データ2, by=両者に共通する列)` データ1（左側）のデータは全て残す形で2つのデータを結合する
- `right_join(データ1, データ2, by=両者に共通する列)` データ2（右側）のデータは全て残す形で2つのデータを結合する
- `full_join(データ1, データ2, by=両者に共通する列)` 結合する2つのデータの全てを保持しながら結合する

`inner_join`に慣れるため、今度は Stand 単位の位置情報データを読み込み、結合させましょう。Stand 単位のデータは、以下のようにしてあらかじめ読み込んでおきます。ただし、Stand の番号は Region 間で重複しているので、Region かつ Stand 単位での結合が必要な点に注意が必要です。

```
standinfo <- read_excel(here("data", "data.xlsx"), sheet="Stand")
standinfo
# A tibble: 40 × 4
  Region Stand   Lon   Lat
  <chr>  <chr> <dbl> <dbl>
1 A       ST1    138.  35.8
2 A       ST2    138.  35.9
(以下、省略)
```

演習 10: Stand をキーにしてデータを結合する

```
treedf <- treedf %>%
  inner_join(., ----, by=c("----", "----"))
```

正しく結合できれば、以下のように表示されるはずです。少し分かりにくいけれど、ちゃんと緯度経度が付与されています。

```
treedf
# A tibble: 1,122 × 10
  Region Stand   DBH Species      Debark   GBH 断面積 地域   Lon   Lat
  <chr>  <chr> <dbl> <chr>        <dbl> <dbl> <dbl> <chr> <dbl> <dbl>
1 A       ST1    35.5 Genus_speciesM     0 111.    988. 白州    138.  35.8
2 A       ST1    40.1 Genus_speciesO     0 126.    1266. 白州    138.  35.8
(以下、省略)
```

次に、より応用的な事例に取り組みます。この直前に読み込んだ自動撮影カメラのデータ（オブジェクト名 `cam_df`）を Region 単位に集計し、毎木データに付与します。しかし、カメラデータにはカメラの ID は入っていますが、Region という列がありません。そのため、まずカメラデータに、Region という列を追加します。

```
cam_df <- cam_df %>%
  mutate(Region=str_extract(cam_id, "^[^_]+"))
```

カメラデータに Region 列を追加することができました。では、Region ごとに撮影されていたニホンジカの枚数を合計しましょう。手順としては、species をニホンジカ（deer）に限定し、Region ごとに撮影枚数（count）を合計します。

演習 11: ニホンジカの撮影枚数を Region 単位で集約する

```
deerphoto <- cam_df %>%
  filter(str_detect(species, "_____")) %>%
  group_by(Region) %>%
  _____(Deer=sum(_____))
```

では、シカ密度データと毎木データを結合します。

演習 12: 一致データのみを結合

```
treedf <- treedf %>%
  inner_join(., deerphoto, by="_____")
```

正しく実行できていれば、以下のように表示されるはずです。

```
treedf
# A tibble: 1,122 × 11
  Region Stand   DBH Species      Debark    GBH 断面積 地域     Lon   Lat Deer
  <chr>   <chr> <dbl> <chr>       <dbl> <dbl> <dbl> <chr> <dbl> <dbl> <dbl>
1 A        ST1     35.5 Genus_speciesM     0 111.    988. 白州    138. 35.8  98
2 A        ST1     40.1 Genus_speciesO     0 126.    1266. 白州    138. 35.8  98
(以下、省略)
```

3.9 複数列に対する一括処理

ティブルの、複数の列に同じ関数などを適用したい状況があると思います。ここでは、複数の列に対し「NA を 0 に置換する」という処理を適用します。この例を実行するために、以前も作成した NA を含むデータを、以下の通り作成します。

```
treedfNA <- treedf %>%
  group_by(Region, Stand, Species) %>%
  reframe(個体数=n()) %>%
  pivot_wider(., id_cols="Species", names_from=c("Region", "Stand"),
  values_from="個体数")
```

では、このデータの複数の列に対し、一括で「NA を 0 に置換する」という処理を行ってみます。

複数の列に対する一括処理

```
treedfNA1 <- treedfNA %>%
  mutate(across(contains("_"), ~if_else(is.na(.), 0, .)))
```

既存の列に対して処理をする場合、mutate 関数で加える形をとります。across は引数で指定した列に、同じく引数で指定した関数を適用します。contains は、指定した文字列を含む列を検索します。

特定の文字列ではなく、全ての列に処理を行う場合、contains の代わりに everything を使います。

4 作図

ここでは、R で作図する方法について説明します。最初の例のところでも出てきましたが、もう少し体系立てて説明します。

R での作図は、R の組み込み関数を使うか、ggplot2 パッケージを使うかの 2 種類に大きく分けられます。より効率的に図を作成できるので、ggplot2 パッケージを使った作図について説明します。

4.1 基本的な記法

ggplot2 による作図は、+ で図に必要な要素を次々と足していくことで行なわれますが（パイプ演算子 %>% と若干紛らわしいです）、基本的には以下のような記法となっています。

ggplot による作図 —————

```
ggplot(データ, aes(x 軸や y 軸として使うデータ、色などの指定))+  
  geom_xxx() # 丸、箱ひげなど様々な種類が用意されている
```

上記の通り、ggplot 内に使うデータを指定し、geom_xxx 関数で描画する形に応じた関数を指定する、という流れです（色々例外はあります）。

以降では、具体的な描画例を示します。

4.1.1 散布図

ggplot による散布図 —————

```
p1 <- ggplot(data=treedf, aes(x=DBH, y=Debark))  
p1 <- p1 + geom_point()
```

複数の点を描画する場合、geom_point を使います。

4.1.2 棒グラフ

ggplot による棒グラフ —————

```
p2 <- ggplot(data=treedf, aes(x=Region))  
p2 <- p2 + geom_bar()
```

棒グラフの場合、geom_bar を使います。

4.1.3 ヒストグラム

ggplot によるヒストグラム —————

```
p3 <- ggplot(data=treedf, aes(x=DBH))  
p3 <- p3 + geom_histogram()
```

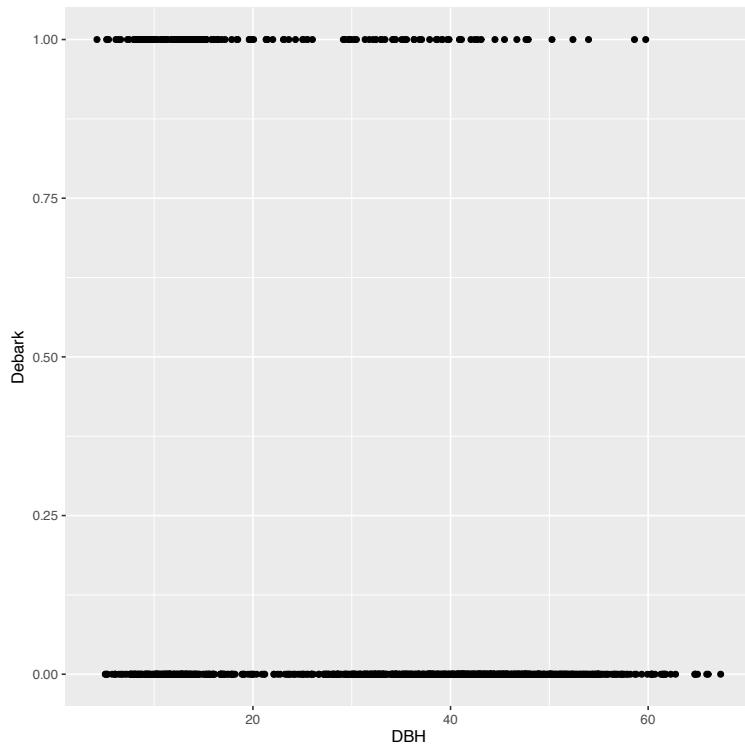


図3 散布図

ヒストグラムの場合、`geom_histogram`を使います。

4.1.4 箱ひげ図

ggplot による箱ひげ図

```
p4 <- ggplot(data=treedf, aes(x=Region, y=DBH))
p4 <- p4 + geom_boxplot()
```

箱ひげ図の場合、`geom_boxplot`を使います。

4.1.5 ヴァイオリンプロット

ggplot によるヴァイオリンプロット

```
p5 <- ggplot(data=treedf, aes(x=Region, y=DBH))
p5 <- p5 + geom_violin()
```

ヴァイオリンプロットの場合、`geom_violin`を使います。

すでに述べたように、ggplot による作図は次々と要素を足していくきます。そのため、ヴァイオリンプロットに箱ひげ図を加えることも可能です。

```
p5 <- p5 + geom_boxplot(width = .2)
```

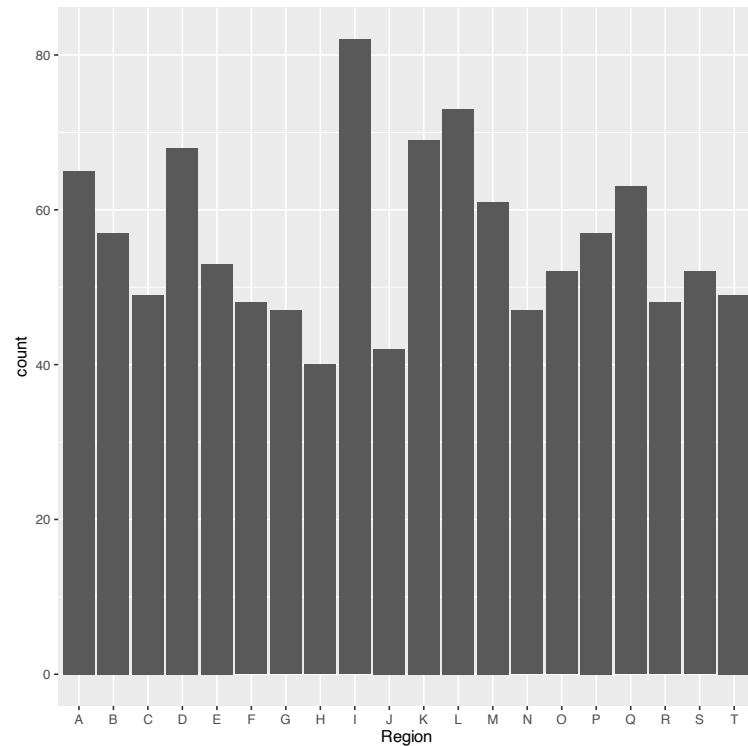


図 4 棒グラフ

こうやって並べてみると、ヴァイオリンプロットの方が箱ひげ図より情報量が多いことがわかります。
geom_xxx は他にもたくさん種類があります。必要に応じて、ご自身で調べて下さい。

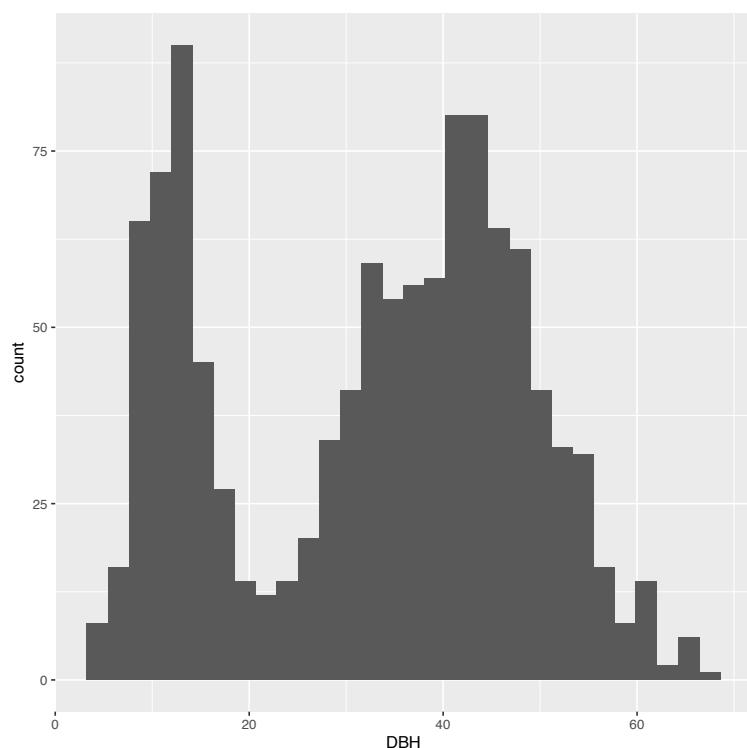


図5 ヒストグラム

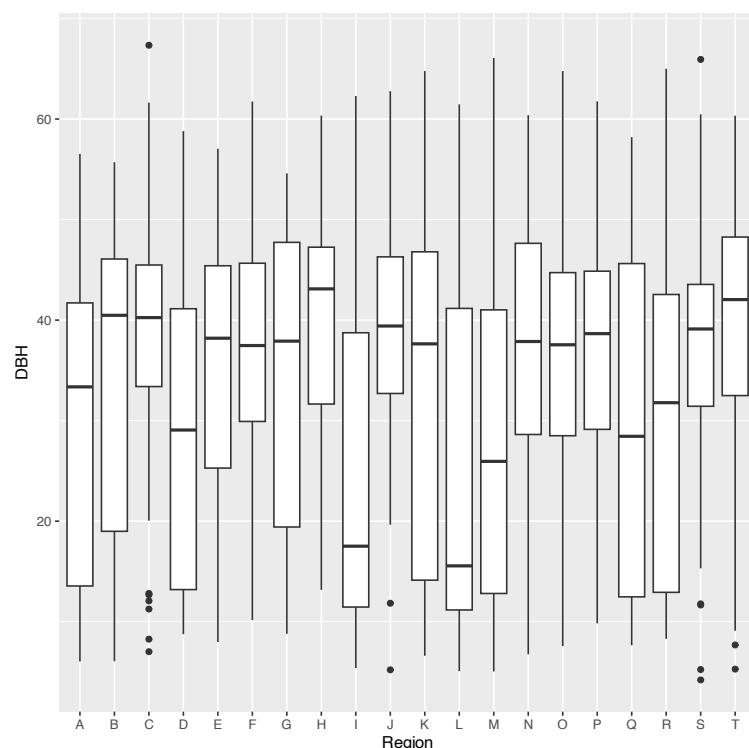


図 6 箱ひげ図

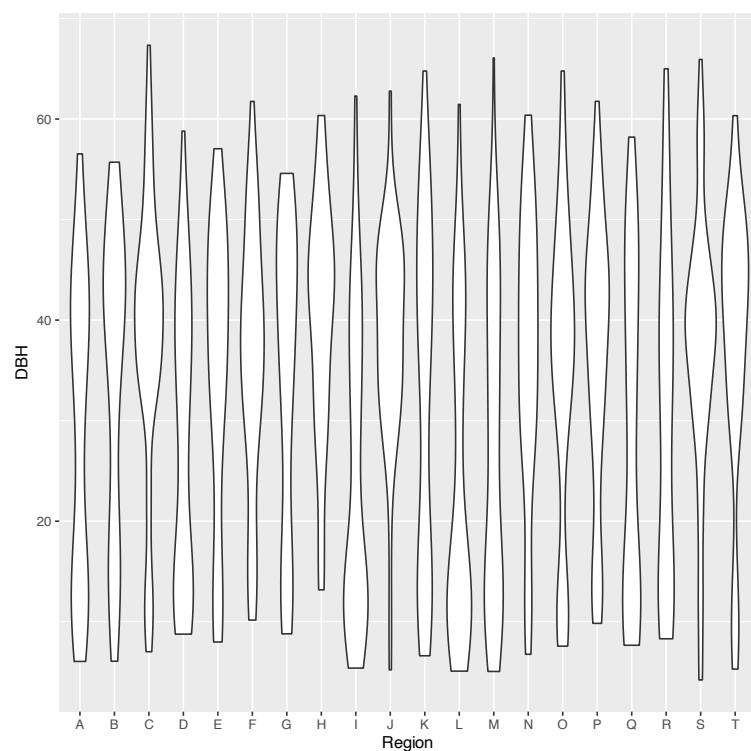


図7 ヴァイオリンプロット

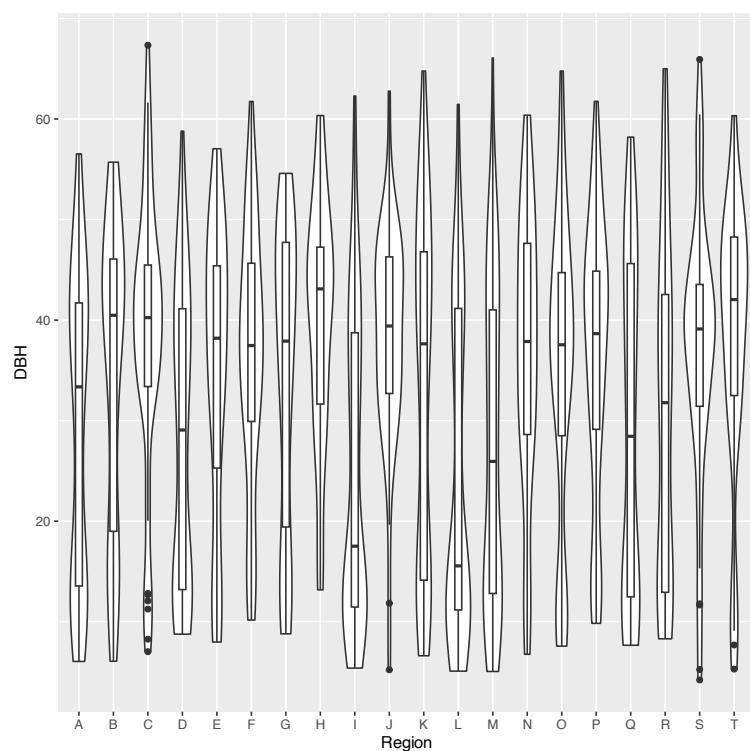


図8 ヴァイオリンプロットと箱ひげ図

4.2 複数の図

ggplot を使うと、複数の図を描画することも容易です。複数の図を作る際、種類の異なる複数の図を描画する場合と、個々の図の要素は同じで用いるデータのカテゴリーだけが異なる複数の図を描画する場合があると思います。それについて、解説します。

4.2.1 異なる複数の図

ggplot で複数の図を並べるパッケージはいくつかありますが、ここでは patchwork パッケージを紹介します。

異なる複数の図の描画

```
library(patchwork)
p1 + p2 + p3 + p5 + plot_layout(ncol=2)
```

ご覧いただければわかるように、ggplot で作成した図のオブジェクトを + で足し合せ、plot_layout で配置

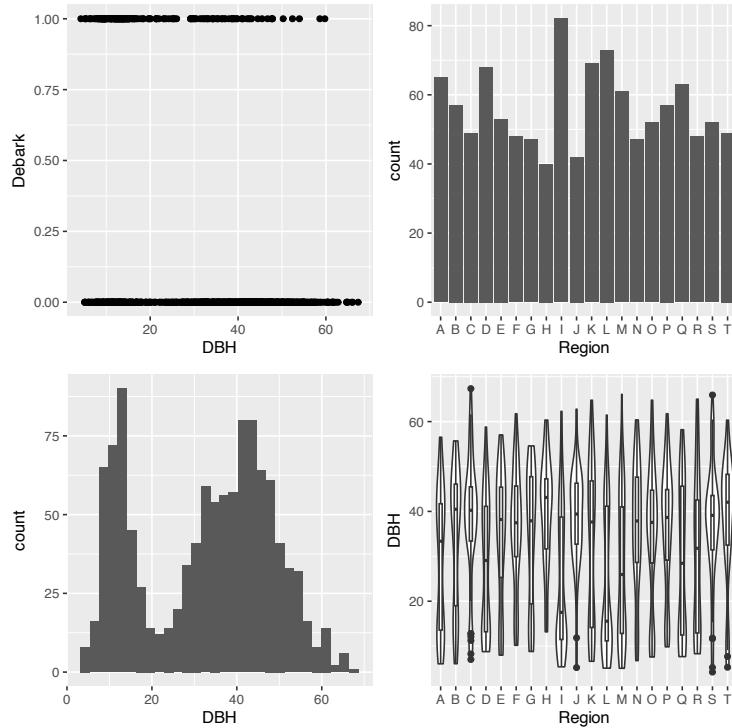


図 9 patchwork パッケージによる複数の図

を指定するだけです。

4.2.2 同じ形式の複数の図

facet_wrap を使います。

同じ形式の複数の図の描画

```
p6 <- ggplot(data=treedf, aes(x=DBH, y=Debark))+  
  geom_point() +  
  facet_wrap(~Species, ncol=5)  
p6
```

描き分けるカテゴリーデータを、`facet_wrap` 内で指定するだけです。

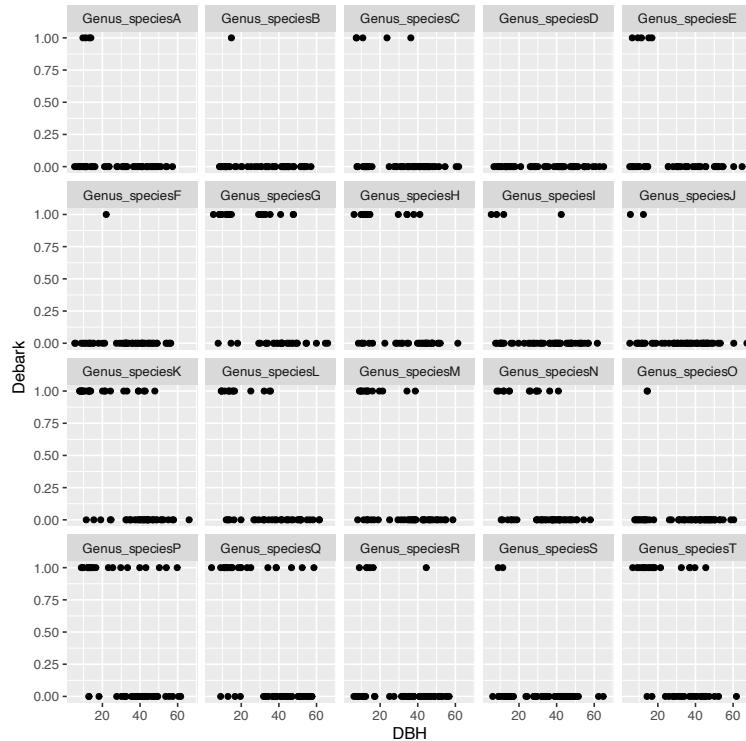


図 10 同じ形式の複数の図

4.3 装飾

では、ここから図の見た目を調整する方法を学びます。先ほど作った散布図に、装飾を加えます。

4.3.1 軸ラベルの変更

`labs`を使います。日本語を使う場合、OSによっては追加の指定が必要です。

軸ラベルの変更 —

```
p1 + labs(x="DBH (cm)", y="Occurrence of debarking")  
### 日本語を使う場合は、mac の場合はフォントの指定が必要  
### Windows の場合は不要  
p1 + labs(x="DBH (cm)", y="剥皮の有無") +  
  theme(text=element_text(family="HiraginoSans-W3"))
```

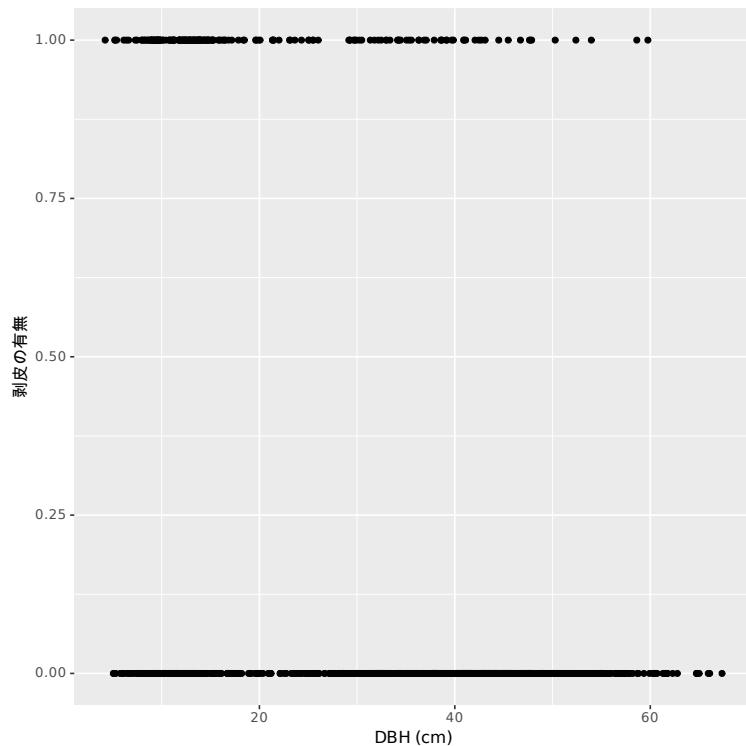


図 11 軸ラベルを指定した図

4.3.2 軸の間隔の調整

`scale_x_continuous`を使います（x 軸の場合。y 軸の場合は x を y に置き換えて下さい）。

軸 m の間隔の調整

```
p1 + scale_x_continuous(breaks=0:6*10)  
# 軸の範囲を指定する場合は、limits=c(xx, xx) のように指定する
```

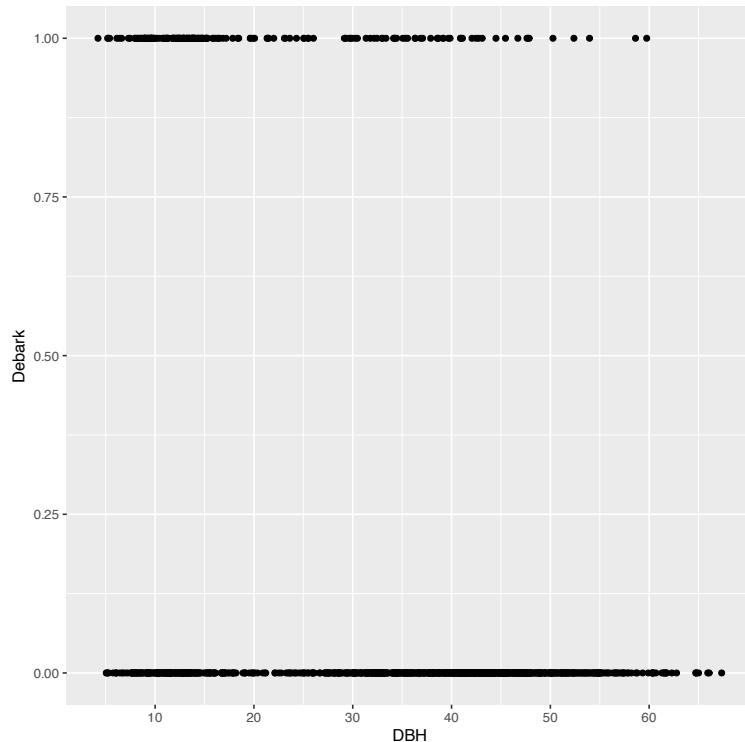


図 12 軸の間隔を変更した図

4.3.3 シンボルのサイズや色の変更

単一のシンボルや色、サイズを指定する場合は aes 外に、データに基づいて指定する場合は aes 内に書きます。

軸ラベルの変更

```
p1 + geom_point(color="red", pch=16, size=4)  
### 透過色も指定できる  
p1 + geom_point(color=rgb(1,0,0,0.1), pch=16, size=4)  
### データに基づいて変更する場合は、aes 内に書く  
p1 + geom_point(aes(color=Species), pch=16, size=4)  
#### シンボルの形を変更したい場合は、shape=で指定
```

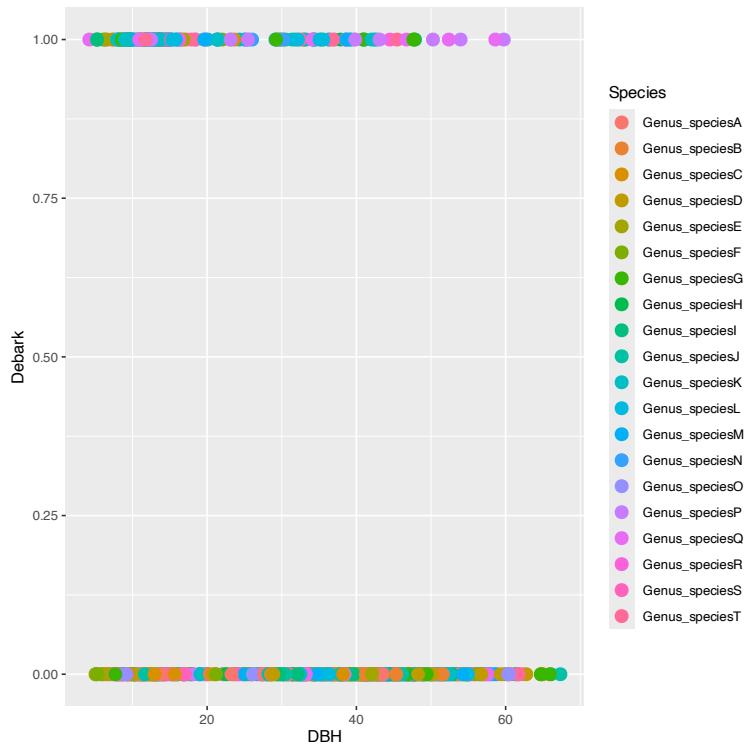


図 13 シンボルの色を変更した図

4.3.4 凡例の変更

凡例は、タイトルのみ変更する場合と、個別の要素まで変更する場合で、書き方が異なります。

凡例の変更

```
### 凡例のタイトルは labs 内で変更
p1 + geom_point(aes(color=Species), pch=16, size=4) +
  labs(color="種") +
  theme(text=element_text(family="HiraginoSans-W3"))

### 凡例の要素ごとに変更する場合
p1 + geom_point(aes(color=Species), pch=16, size=4) +
  scale_color_hue(name="種", labels=str_c("種", 1:length(unique(treedf$Species)))) +
  theme(text=element_text(family="HiraginoSans-W3"))

### 凡例をなくす場合
p1 + geom_point(aes(color=Species), pch=16, size=4) +
  theme(legend.position = "none", text=element_text(family="HiraginoSans-W3"))
```

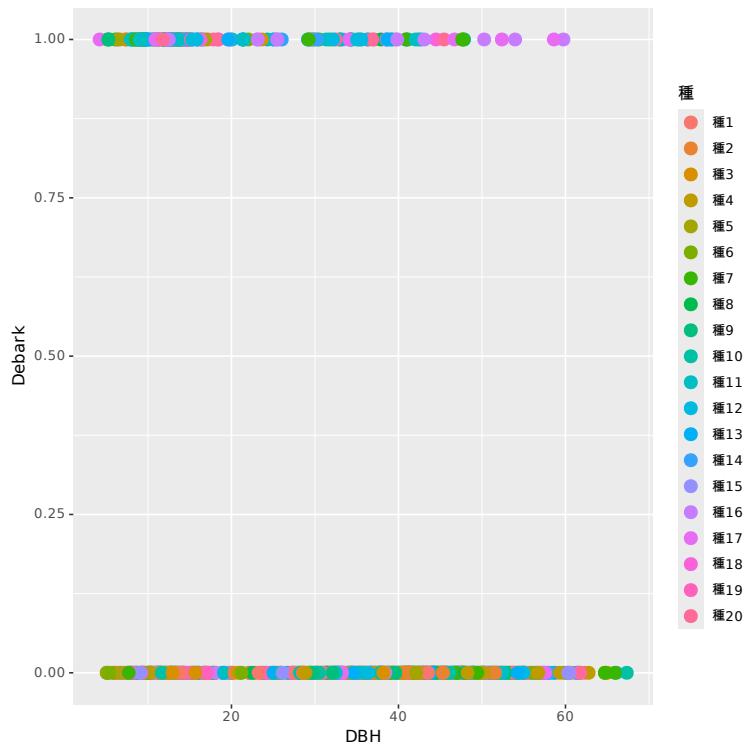


図 14 凡例の要素ごとに変更を行なった図

4.3.5 背景周りの変更

ggplot は、標準だと背景に色がついています。これらを調整する方法です。

背景の色を消す

```
p1 + theme(panel.background = element_blank())
# さすがに外枠の線ぐらい欲しい場合は、以下のように
p1 + theme(panel.background = element_rect(fill = "transparent", colour = "black"))
```

また、ある程度変更が加えられたテーマが用意されており、こちらを使うという手もあります。

組込みのテーマを使う場合

```
### 白黒
p1 + theme_bw()
### 古典的な論文風
p1 + theme_classic()
### これらの用意されているテーマの中では、フォントの指定の仕方が若干異なる
p1 + labs(x="DBH (cm)", y="剥皮の有無")+
  theme_bw(base_family="HiraginoSans-W3")
```

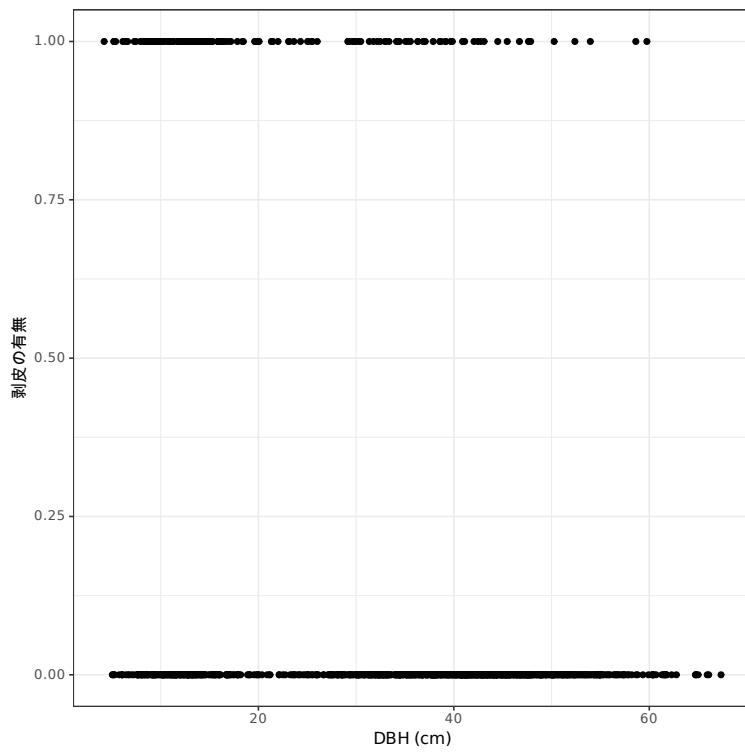


図 15 白黒テーマを適用した図

ggplot は、本当に多彩な図を描画することが可能ですが。ここでは全て紹介しきれませんので、興味のある方はご自身でさらに調べてみてください。

4.4 図の保存方法

ggplot で作成した図は、以下の方法で保存できます。

図の保存方法

```
ggsave(file="xxx.pdf", device=cairo_pdf)
```

第 II 部

R における地理情報データの扱い

5 はじめに

II 部では、地理情報データ（いわゆるベクタデータやラスタデータ）を R で扱う方法について説明します。近年、様々な地理情報データがオープンデータとして公開されており、生態学の分野で活用されています。

地理情報データは地理情報システム（GIS）ソフトで扱うというイメージがあるかも知れませんが、R でも扱うことが可能です。R で扱う利点として、解析のログが残ること、他のデータとの結合が容易であることが挙げられます。

5.1 II 部の事前準備

以下のコマンドを実行しておきましょう。

```
# 作業ディレクトリの設定
## GUI を使う方はここで指定
## コマンドで指定する
### 絶対パス版
setwd("/Users/hayatoijima/Library/CloudStorage/Box-Box/R_lec")
### 相対パス版
setwd(file.path(path.expand("~/"), "Library", "CloudStorage", "Box-Box", "R_lec"))
# パッケージの読み込み
library(here)
library(tidyverse)
library(readxl)
```

6 データの入手

地理情報データは、自分で作成することもありますし、上記のような既存のオープンデータを使うこともあります。自分で作成する場合、位置情報（緯度経度など）が必須です。

一方、代表的な既存データとして、以下のものがあります。

国土数値情報 <https://nlftp.mlit.go.jp/> 国土交通省が管理するサイト。行政区域、土地利用区分、メッシュ気候値など、多くの地理情報が利用可能。

基盤地図情報 <https://www.gsi.go.jp/kiban/> 国土地理院が管理するサイト。詳細な DEM などが利用可能。

自然環境情報 [webGIShttp://gis.biocore.go.jp/webgis/index.html](http://gis.biocore.go.jp/webgis/index.html) 環境省生物多様性センターが管理するサイト。動植物の分布情報などが豊富。

今回は、国土数値情報（図 16）から、山梨県の行政界（N03-20240101_19.shp とその関連ファイル）

と、山梨県周辺の詳細土地利用細分メッシュ（ラスタ版、L03-b-14_5238.tif、L03-b-14_5338.tif、L03-b-14_5339.tif、L03-b-14_5438.tif）を入手しましょう。



図 16 國土數値情報のトップページ

上記ページを下にスクロールすると、利用可能な様々なデータが表示されます。今回はこの中から、行政区域（図 17）と詳細土地利用細分メッシュ（ラスタ版、図 18）をそれぞれ利用します。

図 17 行政区域のページ

なお、ダウンロードに際し、アンケートを求められます。我々利用者がどのように利用しているのかを回答することで運営側の予算獲得や整備の動機付けになりますので、必ず回答しましょう。

7 データの読み込みと操作

7.1 データの読み込み

R で地理情報データを扱うためのパッケージは多数ありますが、ここではベクタデータ用として `sf` パッケージ、ラスタデータ用として `stars` パッケージを紹介します。

先ほどダウンロードしたデータの内、山梨県の行政界はベクタデータ、土地利用データはラスタデータです。基本的にラスタデータの方がファイルサイズを小さくできるので、大面積にわたるデータはラスタデータで扱うことが多いです。



図 18 詳細土地利用細分メッシュ（ラスタ版）のページ

7.1.1 ベクタデータ

ベクタデータは、`read_sf` 関数で読み込みます。まず、先ほど入手した山梨県の行政界を読み込んでみましょう。

ベクタデータの読み込み

```
library(sf)
yama <- read_sf(here("data", "N03-20240101_19.shp"), options = "ENCODING=UTF-8")
yama
Simple feature collection with 36 features and 6 fields
Geometry type: POLYGON
Dimension: XY
Bounding box: xmin: 138.1801 ymin: 35.16838 xmax: 139.1344 ymax: 35.97171
Geodetic CRS: JGD2011
(以下、省略)
```

`sf` 形式の特徴として、先頭にそのデータの種類（ポリゴン、点など）と座標参照系などが表示される点です。

7.1.2 座標参照系とは

座標参照系とは、GIS 上で位置を表す決まりで、地球上の位置と緯度経度を対応させる基準である測地系と、数値による位置（座標）の表し方である座標系を組みわせたものです。用途に応じて様々な座標参照系が存在します。これらの様々な座標参照系は、EPSG コードと呼ばれる数字で識別されます。

地球上の位置を決めるためには、地球の形や基準点などについて定義する必要があります。日本測地系（この中にも何種類かあります）、世界測地系（WGS84 とも呼ばれます）など、いくつかの測地系があります。

地球上の位置を示すのには、一般に緯度経度が用いられます。しかし、緯度経度は地球上のあらゆる場所を指定することができますが、3 次元での角度で表現されるため、その座標値を平面で表現する場合、距離・面積・角度が正確でないという特徴があります。用途によっては、地点間の距離、あるいは面積を知りたいこともあります（困ったことに、生態学では多くの場合こちらの需要が高いです）。そこで、3 次元である地

表1 代表的な座標参照系

EPSG コード	測地系	座標系	備考
4326	世界測地系	緯度経度	Garmin の GPS などで採用
6668	日本測地系 2011	緯度経度	
4612	日本測地系 2000	緯度経度	
3857	世界測地系	擬似メルカトル法	Web のタイル地図などで使われる
6690	日本測地系 2011	UTM53 (東経 138-144)	
3099	日本測地系 2000	UTM53 (東経 138-144)	
6676	日本測地系 2011	平面直角座標 8 系 (新潟県、長野県、山梨県、静岡県)	
2450	日本測地系 2000	平面直角座標 8 系 (新潟県、長野県、山梨県、静岡県)	

球を 2 次元の平面に投影し、メートル法に基づいた XY 座標で表現する方法も存在します。3 次元を 2 次元にしているので、基準点から離れるほどずれが大きくなりますが、ある程度の狭い範囲では実用に耐えうるレベルで正確です。対象とする空間的な広さは、擬似メルカトル法 > UTM 法 > 平面直角座標となっています。

ややこしいことに、同じ「緯度経度」による表現であっても、測地系が違えば異なる緯度経度になってしまいます（基準が違うので）。ですので、測地形と座標系はセットで確認が必要です。世界測地系と日本測地系 2011 および 2000 の誤差は、ほぼないと言われています。ただし、一部のソフトでは、日本測地系 2011 が適切に認識されないことがあります。その場合は、世界測地系か、日本測地系 2000 を使ってください。

いくつか、代表的な座標参照系とその EPSG コードを紹介します。

7.1.3 緯度経度の情報から shp ファイルを作る

自分で shp ファイルを作ることも可能です。緯度経度の情報を持ったファイルを読み込み、`st_as_sf` 関数で shp ファイルに変換します。この際、先ほど説明した座標参照系の EPSG コードを、引数 `crs` に与えてください。

緯度経度情報から shp ファイルを作る —————

```
standloc <- read_excel(here("data", "data.xlsx"), sheet="Stand") %>%
  st_as_sf(., coords=c("Lon", "Lat"), crs=4326)
standloc
Simple feature collection with 40 features and 2 fields
Geometry type: POINT
Dimension:      XY
Bounding box:  xmin: 138.2518 ymin: 35.21183 xmax: 139.0567 ymax: 35.90802
Geodetic CRS:  WGS 84
# A tibble: 40 × 3
  Region Stand      geometry
  * <chr> <chr>    <POINT [°]>
1 A       ST1      (138.2697 35.83841)
2 A       ST2      (138.2518 35.85942)
(以下、省略)
```

7.1.4 ラスタデータ

ラスタデータは、`read_stars` 関数で読み込みます。先ほど入手した、山梨県の詳細土地利用細分メッシュ（ラスタ）の 1 つを読み込んでみましょう。

ラスタデータの読み込み —————

```
library(stars)
e <- read_stars(here("data", "L03-b-14_5238", "L03-b-14_5238.tif"))
stars object with 2 dimensions and 1 attribute
attribute(s):
  L03-b-14_5238.tif
  50      :320805
  0       :100000
  150     : 77458
  70      : 48285
  20      : 41220
  10      : 18839
  (Other): 33393
dimension(s):
  from  to offset      delta  refsys point x/y
  x    1   800    138    0.00125 JGD2000 FALSE [x]
  y    1   800   35.33 -0.0008333 JGD2000 FALSE [y]
```

ラスタデータは、1箇所に1つのデータしか持てません。そのデータの要素ごとの個数や、測地系などに関する情報が表示されます。

今回は、山梨県全体のラスタデータを入手しているので、実際には複数のラスタファイルを読み込み、結合する必要があります。terra パッケージの mosaic 関数でも結合できるのですが、今回のファイルについては結合できなかったため、一度データ部分だけを抽出し、緯度経度情報をもとにベクタファイルとして再構築します。また、ファイル数が多いので、繰り返し命令を使いながら結合します。

```
tifffiles <- list.files(getwd(), pattern="*.tif", recursive = TRUE)
for (i in 1:length(tifffiles)) {
  temp <- read_stars(tifffiles[i]) %>%
    as.data.frame(.) %>%
    rename_at(., 3, ~"lu") %>% # 土地利用を示す列名を「lu」に統一
    filter(!is.na(lu))
  if (i==1) { lu <- temp
  } else { lu <- lu %>% bind_rows(., temp)}
}
# shp ファイルに変換する
lu <- lu %>%
  st_as_sf(., coords=c("x", "y"), crs=4612)
```

最後に shp ファイルに変換する際、日本測地系 2000 の緯度経度形式を指定している点に注意して下さい。数値データから shp ファイルに変換する際は元のデータが作成された際の座標参照系を指定するようにして下さい。

7.2 データの操作

7.2.1 座標参照系の変更

単独のファイルについても状況によって緯度経度がよかったりメートル方が良かつたりしますし、後に行う複数のファイルを結合する際には座標参照系が一致しなければならないので、これは非常に多用します。以下では、山梨県の行政界の座標参照系を、日本測地系 2011 の平面直角座標 8 系に変換しましょう。

座標参照系の変更

```
yama %>%
  st_transform(., crs=6676)
```

7.2.2 バッファを作る

調査地周辺の情報を把握する際、調査地点から任意の範囲の情報を収集することが一般的です。そのようなことをする際に、ポイントデータから任意の半径を持ったバッファを発生させることが有用です。以下では、毎木調査の地点から半径 500m のバッファを発生させましょう。ただし、メートル単位のバッファを適切に発生させるためには、取り扱うファイルの座標参照系がメートル法である必要があります。

任意の半径のバッファの作成

```
standloc %>%
  st_transform(., crs=6676) %>%
  st_buffer(., dist=500)
```

7.2.3 面積の測定

ポリゴンデータの場合、その面積を知りたいことがあると思います。以下では、山梨県の行政界のポリゴンごとの面積を測定します。ただし、標準では平方メートル (m^2) の値が得られるのですが、桁が大きくなりすぎるので、平方キロメートル (km^2) として計算します（ややこしいことに、結果に勝手に [m^2] という単位が表示されるので、誤解しないようにお願いします）。また、こちらも当然ですが、座標参照系がメートル法でなければ正しい値は計算できません。

面積の測定と結果の付与

```
yama %>%
  st_transform(., crs=6676) %>%
  mutate(Area_km=st_area(.) / (1000^2))
```

7.2.4 位置情報の取得

shp ファイルは位置情報を持っていますが、その位置情報（緯度経度など）が数字として必要になる場合もあります。

位置情報の取得

```
st_coordinates(standloc)
```

では、使い方に慣れるために、この数値として取り出した位置情報の数字を、データに結合しましょう。

演習 13: 位置情報の抽出と付与

```
standloc %>%
  ----(lon=st_coordinates(.)[,1], lat=st_coordinates(.)[,2])
```

7.2.5 ポリゴンの重心の取得

ポリゴンは多角形ですが、その代表的な位置の一つとして重心があります。ポリゴンの重心というポイントデータを、以下のようにして作成できます。

ポリゴンの重心の取得

```
yama %>%
  st_centroid(.)
```

7.2.6 ファイル同士の結合

通常のファイル同士の結合については、すでに学びました。`xxxx_join` 関数で結合する際、結合するためには結合するデータに共通の ID が必要でした。一方、地理情報データは自分の位置の情報を持っていますので、このような共通の ID がなくても結合が可能です。ただし、正しく結合するためには、上記の座標参照系が一致している必要があります。

以下では、毎木調査のプロットに、山梨県の市町村情報を結合しましょう。そのために、まず両者の座標参照系を一致させます。毎木調査のプロットは元々は地理情報データではなく後からベクタファイルに変換しており、EPSG コードは 4326（世界測地系、緯度経度）です。一方、山梨県の行政界は、EPSG コード 4612（日本測地系 2011、緯度経度）になっています。ここでは、日本測地系 2011 のメートル法である 6676 に両者を変換しましょう。

演習 14: 座標参照系の変換

```
yama <- yama %>%
  ----(., crs=6676)
standloc <- standloc %>%
  ----(., crs=6676)
```

これで、結合する準備が整いました。ベクタファイル同士の結合は、`st_intersection` を使います。

ファイル同士の結合

```
standloc2 <- standloc %>%
  st_intersection(., yama)

Simple feature collection with 40 features and 8 fields
Geometry type: POINT
Dimension:      XY
Bounding box:   xmin: -22412.75 ymin: -87437.91 xmax: 50459.76 ymax: -10198.36
Projected CRS:  JGD2011 / Japan Plane Rectangular CS VIII
# A tibble: 40 × 9
  Region Stand N03_001 N03_002 N03_003 N03_004
  * <chr>  <chr> <chr>    <chr>    <chr>    <chr>
1 D       ST1   山梨県    <NA>     <NA>     甲府市
2 I       ST1   山梨県    <NA>     <NA>     都留市
  (画面表示の都合上、一部省略しています)
```

ちなみに、上記の「座標参照系の変換」および「ファイル同士の結合」は、わかりやすく説明するために個別に説明しましたが、実際は以下のように一連のパイプライン処理として記述が可能です。

```
yama <- yama %>%
  ----(., crs=6676)
standloc2 <- standloc %>%
```

```
st_transform(., crs=6676) %>%
  st_intersection(., yama)
```

では、ファイル結合を使ってもう少し実用的な解析をやってみましょう。「毎木調査のプロットの半径 500m 以内の、土地利用種ごとの割合」を算出します。この作業手順を整理すると、以下のようになります。

- 每木調査プロットを日本測地系 2011 の平面直角座標 8 系に変換し、半径 500m のバッファを生成する
- 土地利用情報を日本測地系 2011 の平面直角座標 8 系に変換、座標参照系を統一する
- 每木調査プロットのバッファと土地利用情報を結合する
- 地域、プロット、土地利用種ごとに、データ数を計算する
- 地域、プロットごとに、すべての土地利用種の合計データ数に対する土地利用種のデータ数の比率を計算する

これまでに学んだ知識で実行できますので、演習を兼ねて順に実行します。

演習 15: 半径 500m のバッファを作る

```
standloc2 <- standloc %>%
  ----(., crs=6676) %>%
  ----(., dist=____)
```

演習 16: 座標参照系を 6676 に変更

注意！この処理は時間がかかります。

```
lu <- lu %>%
  ----(., crs=____)
```

演習 17: ベクタファイル同士の結合

```
standloc2 <- standloc2 %>%
  ----(., lu)
```

演習 18: 土地利用種ごとのデータ数の計算

```
landuseratio <- standloc2 %>%
  group_by(____, ____, lu) %>%
  ----(Landuse=n())
```

演習 19: プロットごとの土地利用種の割合の計算

```
landuseratio <- landuseratio %>%
  group_by(Region, Stand) %>%
  ----(LanduseRatio=Landuse/sum(Landuse))
  landuseratio
```

```

# A tibble: 73 × 5
# Groups:   Region, Stand [40]
  Region Stand lu    Landuse LanduseRatio
  <chr>  <chr> <fct>  <int>      <dbl>
1 A       ST1    50        75      1
2 A       ST2    50        78      1

```

(以下、省略)

以上で、土地利用種の割合を計算すること自体はできました。ただし、土地利用の種類は数字でコードされており、直感的にわかりにくいです。また、結果を視覚的に把握するためには、データが横方向に展開されていた方が見やすいかもしれません。

土地利用種の割り当てと横方向への展開

```

ludf <- landuseratio %>%
  filter(lu!=0) %>%
  mutate(Type=case_when(lu=="10" ~ "Rice",
                        lu=="20" ~ "Farm",
                        lu=="50" ~ "Forest",
                        lu=="60" ~ "Wasteland",
                        lu=="70" ~ "Buildings",
                        lu=="91" ~ "Road",
                        lu=="92" ~ "Railway",
                        lu=="100" ~ "Humansettlements",
                        lu=="110" ~ "River",
                        lu=="140" ~ "Seashore",
                        lu=="150" ~ "Sea",
                        lu=="160" ~ "Golf")) %>%
  pivot_wider(., id_cols=c("Region", "Stand"), names_from="Type",
             values_from="LanduseRatio", values_fill=0)

```

この一連の処理も、本来であればパイプラインで一度に記述することができます。

```

lu <- lu %>%
  st_transform(., crs=6676)
ludf <- standloc %>%
  st_transform(., crs=6676) %>%
  st_buffer(., dist=500) %>%
  st_intersection(., lu) %>%
  group_by(Region, Stand, lu) %>%
  reframe(Landuse=n()) %>%
  group_by(Region, Stand) %>%

```

```

mutate(LanduseRatio=Landuse/sum(Landuse)) %>%
filter(lu!=0) %>%
mutate(Type=case_when(lu=="10" ~ "Rice",
lu=="20" ~ "Farm",
lu=="50" ~ "Forest",
lu=="60" ~ "Wasteland",
lu=="70" ~ "Buildings",
lu=="91" ~ "Road",
lu=="92" ~ "Railway",
lu=="100" ~ "Humansettlements",
lu=="110" ~ "River",
lu=="140" ~ "Seashore",
lu=="150" ~ "Sea",
lu=="160" ~ "Golf")) %>%
pivot_wider(., id_cols=c("Region", "Stand"), names_from="Type",
values_from="LanduseRatio", values_fill=0)

```

8 作図

地理情報データも、もちろん ggplot で作図することができます。以下では主に、ベクタファイルの作図について扱います。ベクタファイルは geom_sf という専用の関数で扱います。この関数は少し特殊で、渡されたベクタファイルの種類（点、ポリゴンなど）を自動で判別し、描画します。

8.1 データ単独の作図

ベクタファイルの描画

```

p1 <- ggplot()+
  geom_sf(data=yama)+
  theme_bw(base_family="HiraginoSans-W3")
p1

```

ggplot ですので、以前と同じように追加で要素を加えることができます。以下では、市町村ごとに色を変更したり、他のベクタファイルを重ねてみます。

```

p1 + geom_sf(data=yama, aes(fill=N03_004))
p1 + geom_sf(data=standloc, fill="black")

```

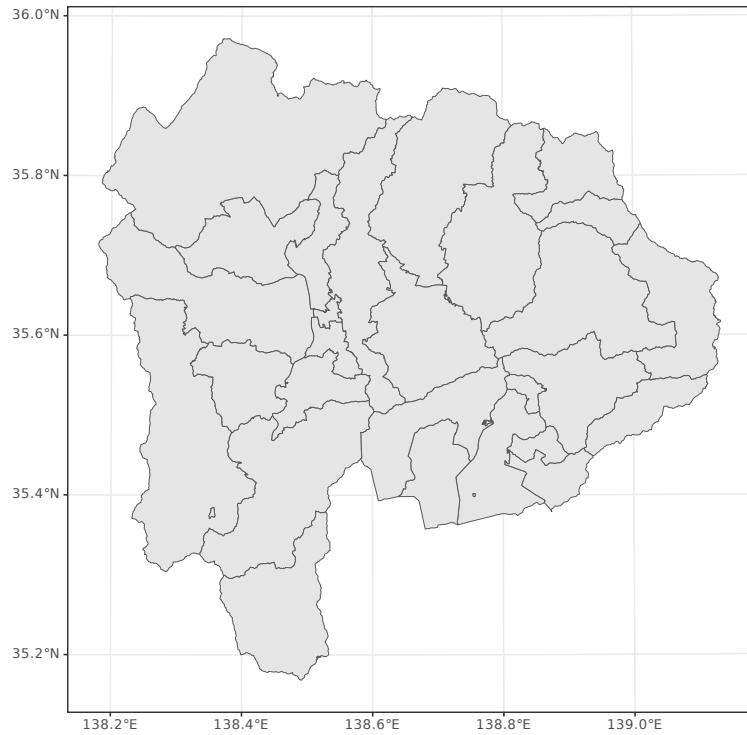


図 19 山梨県の行政界

8.2 背景地図の活用

上記の例では手元にあるベクタファイルを描画しましたが、背景の地図も使うことができます。なお、実行するためにはインターネットに接続する必要があります。

背景地図を伴った作図

```
library(ggspatial)
p2 <- ggplot(standloc) +
  annotation_map_tile(zoomin = -2) +
  geom_sf()
```

`annotation_map_tile` 関数で、背景地図をダウンロードして描画します。引数の `zoomin` の値を大きくするとより解像度の高い地図が得られますが、その分ダウンロードするファイルサイズが大きくなりますので、ほどほどにしておいた方がいいです（作業フォルダに、ひっそりと `roam.cache` というフォルダができていることに注意して下さい！）。

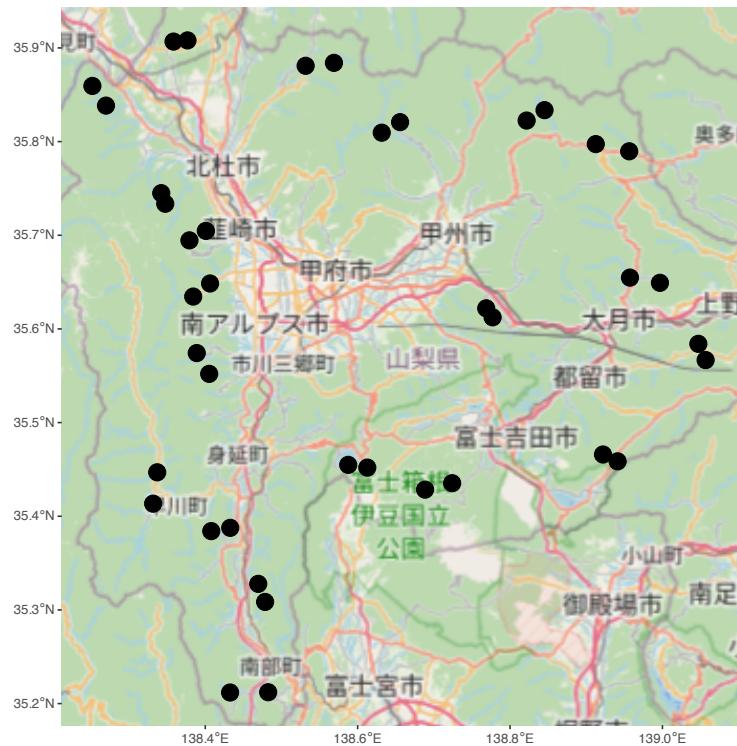


図 20 背景地図を利用した毎木調査プロット位置図

8.3 動的な地図

上記の例では絵としての背景地図を加えましたが、ユーザーの動きに合わせて情報が表示されるような動的な地図も作成可能です。こちらも、実行するためにはインターネットに接続する必要があります。

動的な地図

事前準備として、xy 座標、土地利用割合が入ったデータを作る

```

ludf <- ludf %>%
  left_join(., standloc, by=c("Region", "Stand")) %>%
  # 地図の表示用に、ID 列を作る
  mutate(ID=str_c(Region, Stand, sep=" "))

### leaflet による動的な地図の作成

library(leaflet)
library(leaflet.minicharts)

leaflet(data=ludf) %>%
  addTiles() %>%
  setView(lng=mean(ludf$lon), lat=mean(ludf$lat), zoom=9) %>%
  # 各調査地点
  addCircleMarkers(lng=~lon, lat=~lat, radius=5,
    color="black", weight=2, label=~ID, data=ludf) %>%
  addMinicharts(lng=ludf$lon, lat=ludf$lat,
    type = "bar", chartdata=ludf[, 3:9],
    width = 30, height = 50)

```



図 21 動的な地図

第 III 部

統計解析

9 はじめに

III 部は、R を用いた統計解析、特に一般化線型モデル（Generalized Linear Model, GLM）と一般化線型混合モデル（Generalized Linear Mixed Model, GLMM）について説明します。GLM や GLMM を構築するためには、確率分布と尤度に関する知識が必要不可欠です。そこで、これ以降では、確率分布、尤度、GLM、GLMM を説明します。同時に、パラメータ推定を Markov Chain Monte Carlo (MCMC) 法で推定する方法を学びます。

9.1 III 部の事前準備

以下のコマンドを実行しておきましょう。

```
# 作業ディレクトリの設定
## GUI を使う方はここで指定
## コマンドで指定する
### 絶対パス版
setwd("/Users/hayatoijima/Library/CloudStorage/Box-Box/R_lec")
### 相対パス版
setwd(file.path(path.expand("~/"), "Library", "CloudStorage", "Box-Box", "R_lec"))
# パッケージの読み込み
library(here)
library(tidyverse)
library(readxl)
```

また、MCMC を実行するために NIMBLE を用いますが、os に応じて以下のインストールをお願いします。

Windows ご自身の R のバージョンに合わせた、Rtools (<https://cran.r-project.org/bin/windows/Rtools/>)
macOS Xcode (Apple Store から)

9.2 ベイズ統計とは？

GLM や GLMM は、ある目的変数に影響を与える要因を検討する際に広く用いられているモデルです。また、これら自体も有用ですが、これらを組み合わせてより複雑なモデルを作ることも可能です。そして、モデルが複雑になると、パラメータ推定にベイズ統計が必要になってきます。

ベイズ統計とは、ベイズの定理に基づいてパラメータを推定する統計します。ベイズの定理は、以下の通り

表2 統計学における主要概念の比較

頻度主義	ベイズ主義（ベイズ統計）
パラメータの考え方	唯一無二の正しい値が存在
パラメータの主な推定法	最尤法 MCMC 法、最尤法
事前分布	不要 必要

です。

$$p(A, B) = p(A|B)p(B) = p(B|A)p(A)$$

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

ここで、 $p(A, B)$ は A と B が起こる確率、 $p(A|B)$ は B が起こったという条件で A が起こる確率、 $p(B|A)$ は A が起こったという条件で B が起こる確率、 $p(A)$ は A が起こる確率、 $p(B)$ は B が起こる確率です。ここで、A を推定したいパラメータ (θ)、B をデータ (D) と置き換えると、ベイズの定理は以下のようになります。

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$$

ここで、 $p(D|\theta)$ はパラメータが与えられた場合のデータが得られる尤もらしさであり、のちに説明する尤度です。 $p(\theta)$ は事前情報です。 $p(D)$ はデータの周辺確率で、これを知ることは出来ませんが、 $p(D)$ は事後分布の確率を 1 に収めるための規格化定数ですので、パラメータの推定には影響しません。そのため、上記のベイズの定理は、

$$p(\theta|D) \propto p(D|\theta)p(\theta)$$

のように書かれることもあります。

このベイズの定理に基づけば、ベイズ統計では常に以下のような手続きを取ることになります。

1. 知りたい未知の量を θ とする。
2. データを D とする。
3. $p(\theta|D)$ をベイズの定理に基づいて計算する。

ベイズ統計の利点としては、複雑なモデルを柔軟に構築できること、推定するパラメータの不確実性を推定できることがあると飯島は考えています（異論はあると思います）。生態学は過程が複雑でありそれを表現するモデルも複雑になること、特に応用生態学では推定した結果の不確実性を示すことが求められることから、ベイズ統計に基づいたモデル構築とパラメータ推定方法を学ぶことが有益であると飯島は考えています。

一方、ベイズ統計の利点として、事前に有している知識や情報、あるいは過去の推定結果を事前分布として考慮できる点（ベイズ更新）も挙げられます。しかし、事前に有している知識や情報を事前分布に反映させることは、ややもすると恣意的な推定になりかねないことから、合理的な理由が無い限り行わない方がよいと飯島は考えています。また、計算機速度が向上した現在では、現在使える全てのデータを使って、過去の状態も含めて一度にパラメータを推定する方が合理的だと思います。

10 確率分布

確率分布を、何故理解する必要があるのでしょうか。我々が扱う対象は、実験データの場合もあれば野外の場合もあるでしょう。実験する場合も野外調査をする場合も、扱う対象の条件を完全に制御することはできません。実験の場合、扱う材料や実験器具を完全に同一にすることはできませんし、実験による対象の反応について完全な知識を持っているわけではありません。野外についてはもう絶望的で、同じ種であっても個体差がありますし、野外では様々な環境条件が同時に変動しています。そして、いずれの場合でも人間が実験や調査をする以上、完璧な観測は不可能です。そのため、得られるデータには様々な「誤差」が含まれます（誤差と一口に言っても、様々な誤差があります。自分がどのような誤差を取り扱える、扱えないのか意識することは、非常に重要です）。

このような「誤差」を表現するのに有効なのが、確率分布です。以下では、生態学の分野で特に重要な確率分布について、説明します。

まず、R の乱数生成関数の使い方を説明します。乱数を使いこなせると、擬似的なデータを生成できます。データを自分で生成することが、実は階層モデルの理解に有効です。ここでは、以下の関数を紹介します。

- `rnorm(生成するデータ数, 平均, 標準偏差)` : 正規分布に従う乱数を生成
- `rpois(生成するデータ数, 平均)` : ポアソン分布に従う乱数を生成
- `rbinom(生成するデータ数, 総試行回数, 生起確率)` : 二項分布に従う乱数を生成

ちなみに、R では関数名の前に ? をつけて打ち込む（例えば?`rnorm`）と、ヘルプが表示されます。使いたい関数名が分かっていて関数の使用を調べたいときに有用です。

10.1 正規分布

正規分布は、ある平均周辺にある程度のばらつきを持って分布する $-\infty$ から ∞ までの実数を表現する確率分布で、平均と分散という 2 つのパラメータを持ちます。ある確率分布に由来するデータを取得してその平均値を取ることを繰り返した場合にその平均値の分布が正規分布になるという「中心極限定理」のため、正規分布はこれまで様々な統計的検定に用いられてきました。一般に、長さや重量などは正規分布に従うとされています。正規分布の確率密度関数は、以下のとおりです。

正規分布

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

ここで μ は平均、 σ は標準偏差 (σ^2 が分散)、 x はデータです。正規分布の形は、 μ や σ を変えると変わります（図 22）。

では、この正規分布に従う乱数を自分で作ってみましょう。以下のように R に打ち込んで見て下さい。

```
set.seed(1) #乱数の種の指定（同じ結果を得るために）
rnorm(100, 0, 1)
[1] -0.626453811  0.183643324 -0.835628612
[4]  1.595280802  0.329507772 -0.820468384
```

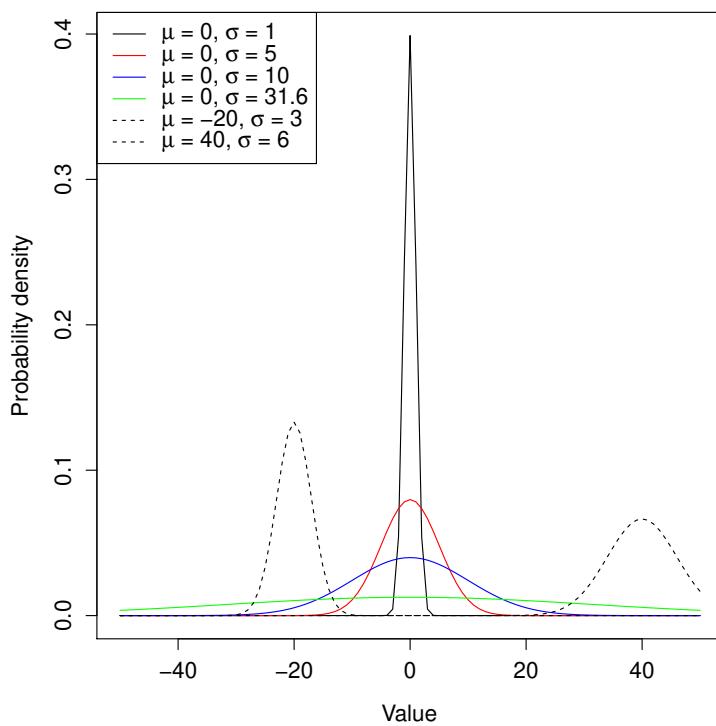


図 22 正規分布

(以下略)

`rnorm(作りたい乱数の数, 平均, 標準偏差)` のように使います。このように、R では種々の乱数を非常に簡単に作ることができます。あまりに簡単すぎて、本当に正規分布の乱数が作れたのか疑問に思うかもしれません。では、得られた値のヒストグラムを書いてみましょう。

```
hist(rnorm(100, 0, 1))
```

確かに、正規分布に従う乱数が生成できているようです。N を大きくしていけば、さらに正規分布のような形になります。

演習 20: 正規乱数の生成

- 平均が-1、標準偏差が 3 の正規乱数を 10000 個生成し、ヒストグラムで描画せよ。ヒストグラムは、`hist()` 関数や、`ggplot2` の `geom_histogram` 関数で作成できる。

10.2 二項分布

二項分布は、総試行回数 n の内、成功する（事象が生起する）回数 k を表現する確率分布で、生起確率 p という 1 つのパラメータを持ちます。コイン投げをして表が出る回数、調査個体の内死亡した個体数などは二項分布に従うと考えられます。二項分布は回数という 0 以上の整数のみを扱うため、得られる値は離散的です。

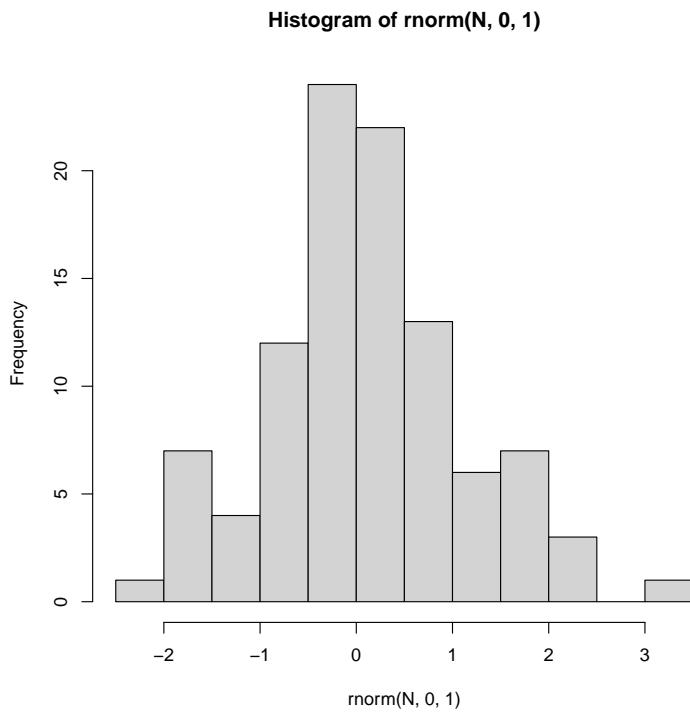


図 23 生成した正規分布のヒストグラム

その確率質量関数は、以下のとおりです。

二項分布

$$P[X = k] = \binom{n}{k} p^k (1-p)^{n-k}$$

ここで p は生起確率です。二項分布の形は p を変えると変わります（図 24）。

二項分布は正規分布と異なり、ばらつきを指定するパラメータがありません。そのため、二項分布は生起確率 p が決まるとばらつきも決まります。また、二項分布は総試行回数 n に応じて取り得る値の範囲が決まってしまうので、生起確率 p の値によってばらつきが異なります。

また、試行回数の違いによって精度が異なる点にも注意が必要です（図 25）。感覚的にも、10 回コイン投げをして 5 回表が出たデータによって「このコインの表が出る確率は 0.5 だ」と結論するよりも、100 回コイン投げをして 50 回表が出たデータによって「このコインの表が出る確率は 0.5 だ」と結論する方が確からしいと思います。

では、この二項分布に従う乱数を作ってみましょう。

```
set.seed(1) #乱数の種の指定（同じ結果を得るために）
# 総試行回数が 1 の場合 (=ベルヌーイ分布)
rbinom(100, 1, 0.5)
[1] 0 0 1 1 0 1 1 1 0 0 0 1 0 1 0 1 1 0 1 1 0 1 0 0
```

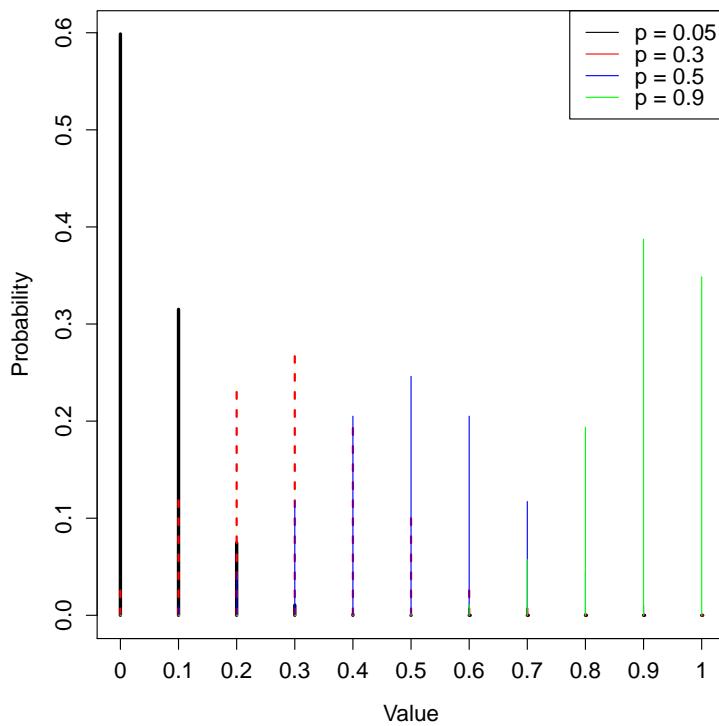


図 24 生起確率の違いと二項分布の形の違い

```
[26] 0 0 0 1 0 0 1 0 0 1 1 1 0 1 0 1 1 1 1 1 1 0 0 1 1
```

(途中省略)

総試行回数が 10 の場合

```
rbinom(100, 10, 0.5)
```

```
[1] 6 4 4 9 6 4 3 5 7 5 8 6 4 5 3 2 6 3 5 6 9 5 5 4 6
```

```
[26] 5 5 4 4 5 5 3 2 6 7 5 5 5 8 5 6 5 4 4 6 5 4 6 3 7
```

(途中省略)

演習 21: 二項乱数の生成

- 100 回コイン投げをした時に得られる表 (1) と裏 (0) のデータを生成せよ。ただし、表が出る確率は 0.5 とする。

10.3 ポアソン分布

ポアソン分布は、一定時間内に事象が生起する回数 k を表現する確率分布で、一定時間内の事象の生起回数に関する λ という 1 つのパラメータを持ちます。一定時間内に電話がかかってくる回数、一定区画内に出現するある種の個体数などはポアソン分布に従うと考えられます。ポアソン分布も二項分布同様に、得られる値は離散的です。その確率質量関数は、以下のとおりです。

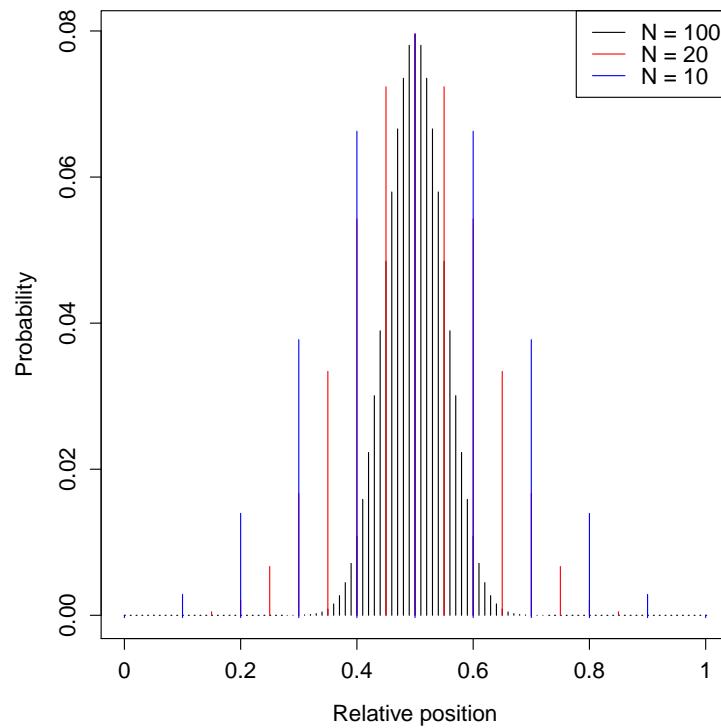


図 25 試行回数の違いと二項分布の形の違い

ポアソン分布

$$P[X = k] = \frac{\lambda^k e^{-\lambda}}{k!}$$

ポアソン分布も二項分布同様、ばらつきを指定するパラメータはありません。そのため、 λ の値が決まるとばらつきも決まります（図 26）。

では、やはり同じように、ポアソン分布に従う乱数を作ってみましょう。

```
set.seed(1) #乱数の種の指定（同じ結果を得るため）
rpois(100, 3)
[1] 2 2 3 5 2 5 6 4 3 1 2 1 4 2 4 3 4 8 2 4 6 2 4 1 2
[26] 2 0 2 5 2 3 3 3 1 5 4 4 1 4 2 5 3 4 3 3 4 0 3 4 4
(途中省略)
```

演習 22: ポアソン乱数の生成

- 同じ大きさの区画 100 箇所において、アカネズミの数を調査したときに得られるアカネズミの計数データを生成せよ。ただし、100 個の区画のアカネズミの計数値のばらつきは、平均 5 のポアソン分布に従うことが分かっている。

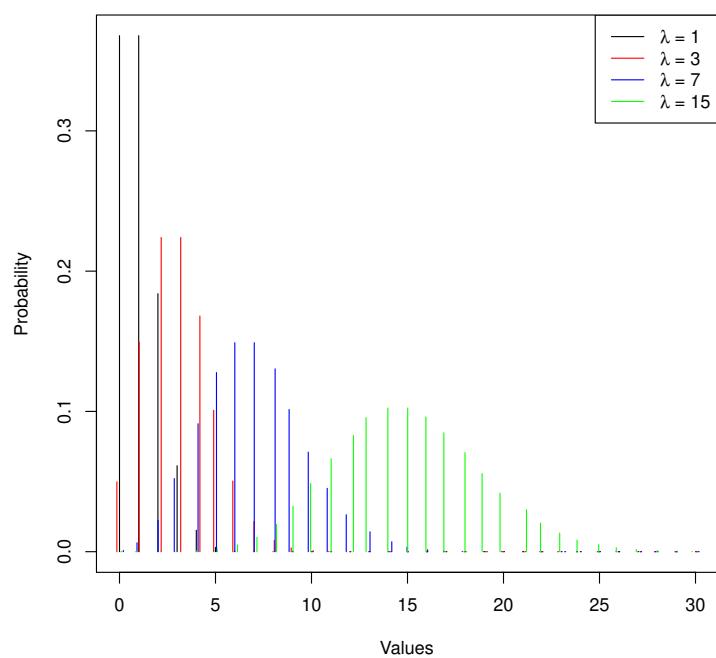


図 26 ポアソン分布

11 尤度

注！

- この文章は粕谷（1997）の最尤法の説明をほぼそのまま使わせてもらっています。

尤度とは、「あるデータが得られたときに、ある確率分布の元でデータをどれだけ尤もしく表現できているかを示すもの」です（確率ではありません）。そして、尤度が最も高くなるようなパラメータの値をパラメータの妥当な推定値と考える考え方を、最尤法といいます。以下では、具体例で尤度と最尤法を説明します。

n 回コイントスをして表か裏かを調べ、 r 回表、 $(n - r)$ 回裏が出たとしたときに、表が出る確率は以下の二項分布で定義できます。この $L(p|r)$ が、尤度です。尤度は、データによく当てはまっているほど値が大きくなります。

$$\begin{aligned} L(p|r) &= \binom{n}{r} p^r (1-p)^{(n-r)} \\ &= {}_n C_r p^r (1-p)^{(n-r)} \end{aligned}$$

例えば3回コイントスをして2回表だったとします。このデータだけから「このコインの表が出る確率は？」と問われたら、ほとんどの人が（表が出た回数）/（全試行回数）で $2/3 = 0.6666\ldots$ と答えるでしょう。「コインの表が出る確率」は、二項分布のパラメータである生起確率 p と同じものです。

では、 $p = 2/3$ にすると、本当に尤度は最大になるのでしょうか？図を描いて見ましょう。この図からわかるように、 $p = 2/3$ が尤度を最も大きくする値のようです。では、3回コイントスして2回表が出た場合に、 p を $2/3$ にすると何故尤度が最も大きくなるのでしょうか？

尤度に関わらず、ある関数の変曲点はその関数を微分することで求めることができます。今回は「3回コイントスして2回表が出た」というデータが得られているので、そのときの尤度を微分して、尤度が最大となる p を求めることができます。

$$\begin{aligned} L(p|r) &= {}_3 C_2 p^2 (1-p)^{(3-2)} \\ &= 3 \times p^2 (1-p) \end{aligned}$$

この尤度を微分すると、

$$\begin{aligned} L(p|r)' &= 6p - 9p^2 = 0 \\ &= p(6 - 9p) = 0 \\ &= p = 2/3 \text{ or } 0 \end{aligned}$$

となり、 $2/3$ が導かれました。この、

- $p = 2/3$ を求める方法が**最尤法**（尤度という基準で確率分布のパラメータをデータに尤も当てはまるよう決める）
- $p = 2/3$ が**最尤推定値**。

です。

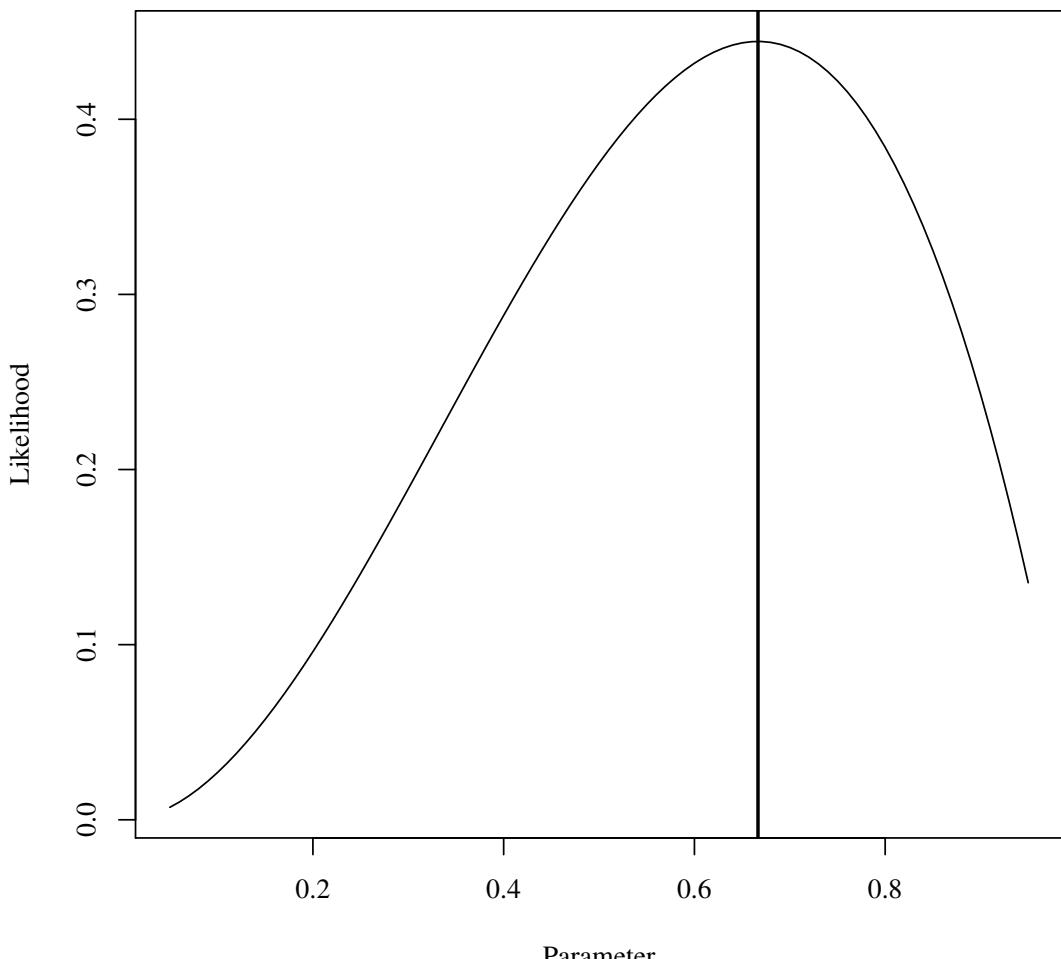


図 27 パラメータを変化させたときの二項分布の尤度

ベイズ統計では、事後確率はこの尤度と事前確率に比例します ($p(\theta|D) \propto p(D|\theta)p(\theta)$)。事前確率に情報をほとんど与えなければ、事後確率は尤度に比例します。そのため、ベイズ統計においても、所与のモデルの元でデータを表現できるようなパラメータを推定しようとすることに、変わりはありません。

11.1 対数尤度

対数尤度とは、尤度の対数をとったものです。実際の線形モデルにおける最尤法は、対数尤度を対象として実行されることがほとんどです。

なぜかというと、使うのは尤度でも対数尤度でもどちらでもいいのですが、計算は対数尤度でないと最尤推定値が求められないことが多いからです。例えば正規分布を例にとって見ましょう。

$y = y_1, y_2, \dots, y_{100}$ という 100 個のデータがあったときに、 y_1 が得られる尤もらしさ（尤度）は、

$$\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(y_1 - \mu)^2\right)$$

と定義できます。同じように y_2, y_{100} なども得られますぐ、これらのデータ y が得られる尤度を最も大きくするためには、 $(y_1 \text{ が得られる尤度}) \times (y_2 \text{ が得られる尤度}) \times \dots \times (y_{100} \text{ が得られる尤度})$ を最も大きくする必要があります。この尤度を $L(\mu, \sigma^2 | y)$ とすると、

$$\begin{aligned} L(\mu, \sigma^2 | y) &= \prod_{i=1}^{100} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(y_i - \mu)^2\right) \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^{100} \exp\left(-\sum_{i=1}^{100} \frac{(y_i - \mu)^2}{2\sigma^2}\right) \end{aligned}$$

となり、解を解析的に求めることが難しくなります。そこで対数をとります。対数をとった先ほどの尤度を $\log L(\mu, \sigma^2 | y)$ とすると、

$$\begin{aligned} \log L(\mu, \sigma^2 | y) &= \log\left(\left(\frac{1}{\sqrt{2\pi}\sigma}\right)^{100} \exp\left(-\sum_{i=1}^{100} \frac{(y_i - \mu)^2}{2\sigma^2}\right)\right) \\ &= -100 \times \log(\sqrt{2\pi}\sigma) - \sum_{i=1}^{100} \frac{(y_i - \mu)^2}{2\sigma^2} \end{aligned}$$

となり、結局のところ $(y - \mu)^2 / 2\sigma^2$ の部分を微分すれば最尤推定値が求まることがあります。対数尤度にすると計算が楽になることが多いので、ほとんどの場合対数尤度を用います。

12 GLM (一般化線形モデル)

GLM とは、Generalized Linear Model の略です。線形モデルとは説明変数が線形に結合（和）したモデルのことで、残差の分布として单一の確率分布ではなく指指数型分布属の確率分布を適用できるように拡張したものなので一般化が付いています。GLM の基本的な構造は、以下のとおりです。

$$\eta = \beta X^T \quad (1)$$

$$\eta = g(\mu) \quad (2)$$

$$\text{Data} \sim \text{Probability distribution}(\mu) \quad (3)$$

式 1 は決定論的モデル (Deterministic model) 、式 3 は確率論的モデル (Stochastic model) と呼ばれます。そして、式 2 の関数 $g()$ はリンク関数と呼ばれます。これらの存在を意識することは、適切な解析を行うために重要です。

- 決定論的モデル: 説明変数と応答変数の関係性を示したもの。自分がどう現象を捉えているかといつてもいい。
- 確率論的モデル: 得られる現象がどのような確率的変動をもって生じるかに関する仮説。
- リンク関数: 決定論的モデルの予測値（線形予測子）を、確率論的モデルで採用する確率分布のパラメータに求められる条件に変換するための関数。

12.1 確率論的モデル

先ほど、自分がどのように現象を捉えているか表現したのが決定論的モデルだと説明しました。しかし、繰り返し述べているように、データの観測には様々な誤差が含まれます。そのため、得られたデータのばらつきを決定論的モデルだけで説明することはできません。

この、観測誤差を確率分布で表現したものが、確率論的モデルです。つまり、観測される現象は決定論的モデルと確率論的モデルの両方が作用して観測されるといえます（図 28）。

12.2 リンク関数

我々が興味があるのは、決定論的モデルによる予測が観測された現象とどのような関係にあるのかということです。これは、決定論的モデルに組み込んだ要因の係数を推定するという問題として考えることができます。

では、係数の推定はどのような基準で行えばよいのでしょうか？観測される現象を表現する確率分布の形は、そのパラメータによって決定されます。そのため、観測される現象に関する確率分布のパラメータを、決定論的モデルによって、観測される現象に最も当てはまるように決定すればいいことになります。決定論的モデルの挙動は、そのモデルに組み込まれた要因の係数や切片によって決まりますから、係数や切片を観測された現象が最もよく再現できるように決定する、と考えることができます。

しかし、決定論的モデルによる予測値（線形予測子）が取り得る値は、そのままでは確率論的モデルのパラメータとして適切でない可能性があります。例えば、二項分布のパラメータ p は、必ず 0 から 1 の間でなくてはなりません。

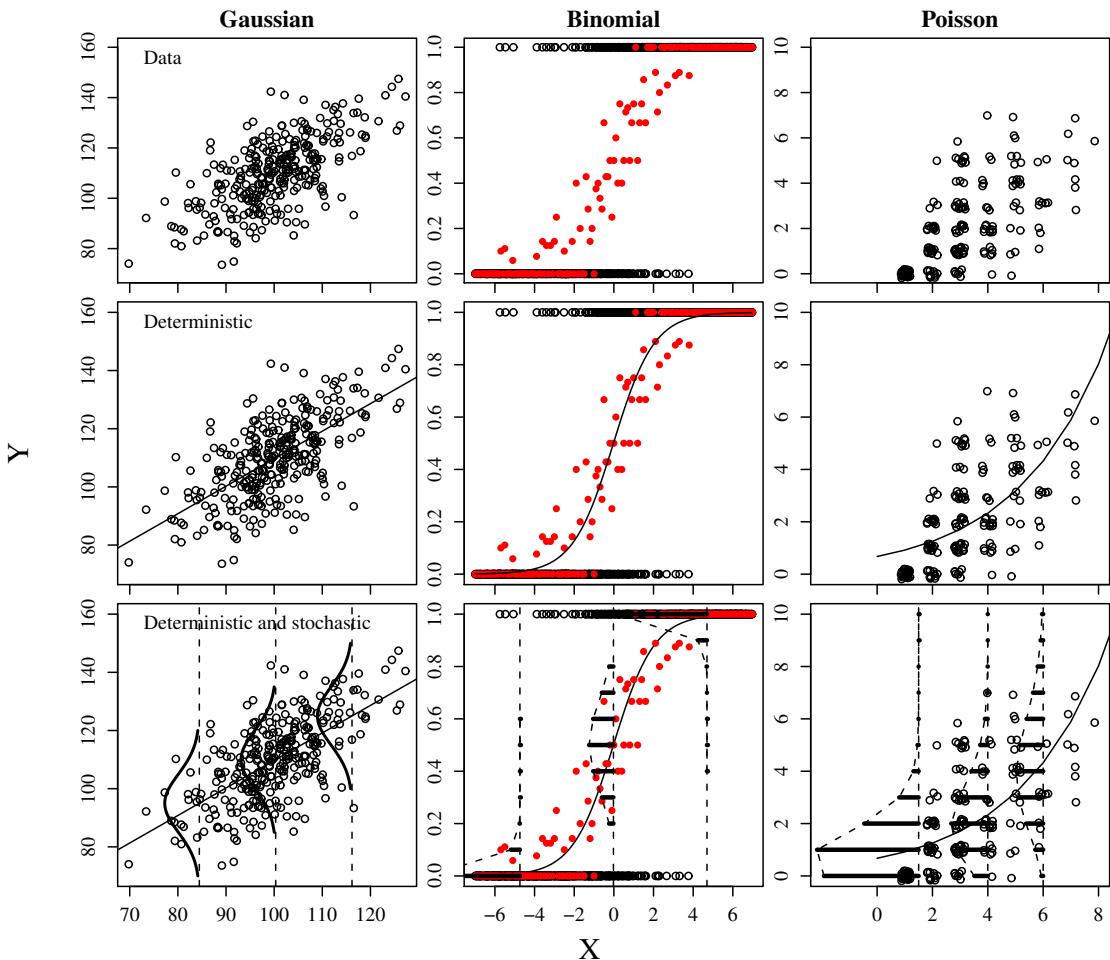


図 28 実データと決定論および確率論的モデルの関係

上段が実データ、中段が決定論的モデルによる予測、下段が確率論的モデルを加えた図。下段では、確率論的モデルによって予測される値の発生 確率が、太線で示されている。二項分布の図では、X 軸の値が 0.1 毎に平均した Y の値を、赤丸で示している。

ここで登場するのが、リンク関数です。リンク関数は、線形予測子を、確率分布のパラメータとして用いることができるよう変換するための関数です。二項分布の例を挙げます。

```

x <- (-10):10      # 決定論的モデルの予測値
p <- 1/(1+exp(-x)) # x を 0 から 1 に収まるように変換
# これは log(p/(1-p)) <- x と等しい
# log(p/(1-p)) はロジットリンク関数
plot(p ~ x)

```

以上を踏まえると、GLM は以下の手順から構築されます。

- 観測される（説明したい）現象が、何らかの確率分布に従って生じると仮定。すなわち、確率論的モデルの決定。

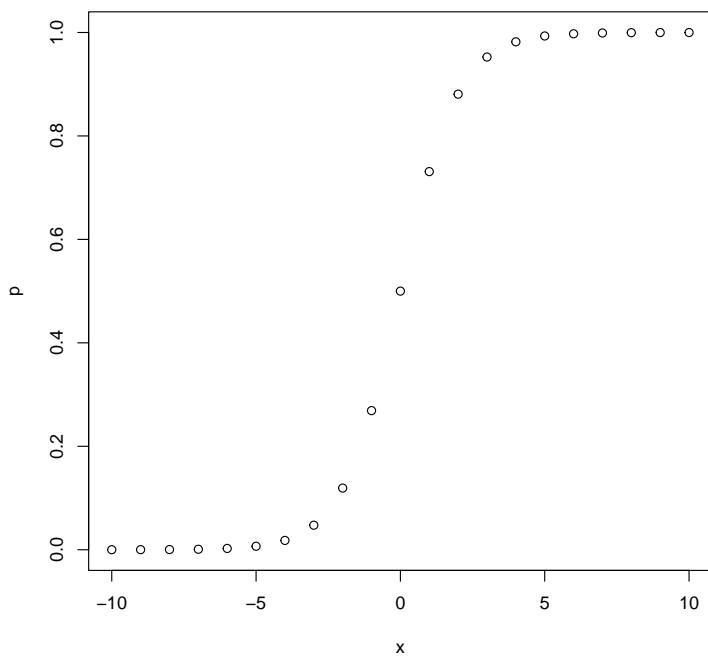


図 29 ロジットリンク関数を適用した結果

表 3 確率分布とリンク関数

リンク関数（標準）		式
正規分布	identity	
二項分布	logit	$\log\left(\frac{p}{1-p}\right)$
ポアソン分布	log	$\log(x)$

- 考えている要因から観測される現象を説明するモデルを作成。すなわち、決定論的モデルの決定。
- 観測される現象を最もよく表現できるように、確率分布のパラメータ、すなわち決定論的モデルの係数を推定する。

「観測される現象を最もよく表現できる」ということを評価する指標として、すでに説明した尤度(*Likelihood*)を用いることができます。

12.3 データの読み込み

では、GLM に関する知識を習得したところで、実際に GLM のパラメータ推定を行ってみます。I 部で用いたデータを、再度読み込みます。

用いるデータの読み込み

```
# 自動撮影カメラのデータ
cam_files <- list.files(here(file.path("data", "camera")),
  recursive=TRUE, pattern="*.xlsx")
cam_df <- read_excel(here(file.path("data", "camera")), cam_files[1]),
  sheet="Sheet 1")
for (i in 2:length(cam_files)) {
  cam_df <- cam_df %>%
    bind_rows(., read_excel(here(file.path("data", "camera")), cam_files[i]),
      sheet="Sheet 1"))
}
deerphoto <- cam_df %>%
  mutate(Region=str_extract(cam_id, "^[^_]+")) %>%
  filter(str_detect(species, "deer")) %>%
  group_by(Region) %>%
  reframe(Deer=sum(count))
# Region 単位のデータ
regioninfo <- read_excel(here("data", "data.xlsx"), sheet="Region")
# Stand 単位のデータ
standinfo <- read_excel(here("data", "data.xlsx"), sheet="Stand")
# 毎木データ
treedf <- read_excel(here("data", "data.xlsx"), sheet="Trees") %>%
  inner_join(., deerphoto, by=c("Region")) %>%
  inner_join(., regioninfo, by=c("Region")) %>%
  inner_join(., standinfo, by=c("Region", "Stand"))
```

I部で述べたように、このデータは、ニホンジカが多い地域では、ニホンジカによる樹皮剥ぎが多く、また稚樹の本数は少ないという仮説を検証するために得られたものです。ひとまず、以下では樹皮剥ぎに関する解析を行います。

樹皮剥ぎは、樹種による差があり、また木の太さも影響することが知られています。

12.4 R の関数による GLM

12.4.1 GLM の構成要素

すでに説明したように、GLM の構成要素は以下のとおりです。

決定論的モデル 解析者が考えた「要因」と「現象」の関係を説明するモデル

確率論的モデル 決定論的モデルで説明できない「ばらつき」を説明するモデル

リンク関数 決定論的モデルによる予測値を確率分布の条件に適合するように変換する関数

今回のデータでは、シカの多さ、樹種、木の太さが要因であり、樹皮剥の有無が現象です。この考えが、決定

論的モデルです。では、実際にこれらのデータはどのような関係になっているでしょうか。

```
library(ggplot2)
library(patchwork)

p1 <- ggplot(data=treedf, aes(x=Deer, y=Debark))+  
  geom_point(pch=16, col=rgb(0,0,0,0.05))

meandb <- treedf %>% group_by(Species) %>% reframe(Debark=mean(Debark))

p2 <- ggplot(data=meandb, aes(x=Species, y=Debark))+  
  geom_bar(stat = "identity")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

p3 <- ggplot(data=treedf, aes(x=DBH, y=Debark))+  
  geom_point(pch=16, col=rgb(0,0,0,0.05))

(p1 | p3) / p2
```

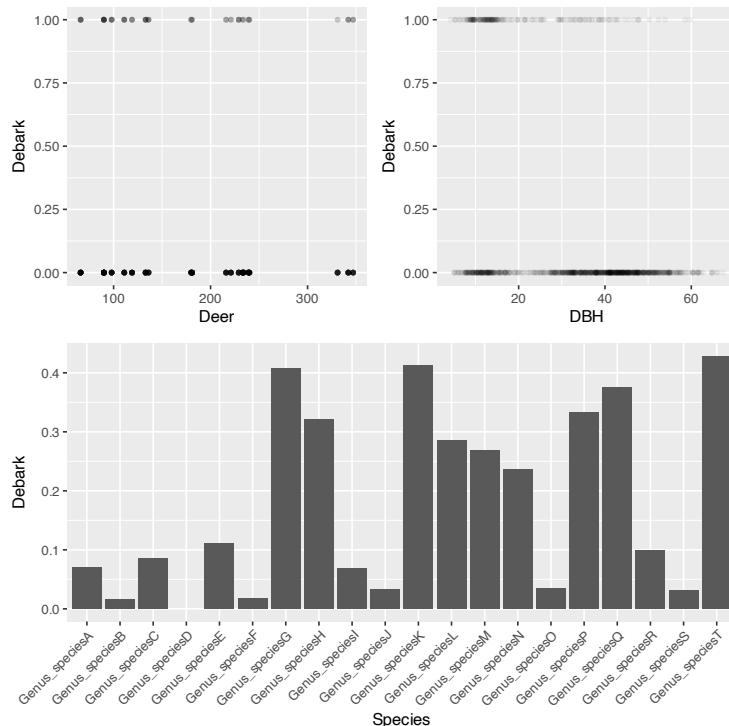


図 30 説明変数と応答変数の関係

シカが多く、木が細いと剥皮が多そうです。また、樹種によって剥皮のされやすさは異なっていそうです。ただし、いずれのデータもばらつきは大きく、要因で全てが説明できるわけではなさそうです。説明変数で説明できない部分は、もちろん決定論的モデルで考慮していない要因も影響していると思われますが、今回はそのようなデータを持っていません。このような関係が本当に存在するかを、GLM で推定します。

R には、`glm()` という便利な関数が用意されています。この関数中で、先程確認した決定論的モデル、確率

論的モデル、リンク関数を指定し、GLM のパラメータを推定します。ただし、都合上、樹種は今回は要因として含めないことにします。

```
res <- glm(Debark ~ Deer + DBH, family=binomial(link="logit"), data = treedf)
summary(res)
```

Call:

```
glm(formula = Debark ~ Deer + DBH, family = binomial(link = "logit"),
     data = treedf)
```

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	0.121656	0.210947	0.577	0.5641
Deer	0.002431	0.001104	2.201	0.0277 *
DBH	-0.076888	0.006795	-11.316	<2e-16 ***

(以下、省略)

glm() の結果は、summary() で取り出すことができます。結果の見方については、ここでは解説しません。

なお、glm() 関数では、標準のリンク関数の場合リンク関数の指定は省略可能です。また、用いるデータを示す data も明示する必要はありません。そのため、上記のコードは以下のように省略可能です。

```
res <- glm(Debark ~ Deer + DBH, family=binomial, treedf)
```

12.4.2 GLM の結果の作図

ggplot を使うと、GLM による予測曲線を簡単に描画することができます。

— GLM による予測曲線 —

```
p1 <- ggplot(data=treedf, aes(x=DBH, y=Debark))+
  geom_point(pch=16, col=rgb(0,0,0,0.1), size=3)+
  geom_smooth(method="glm", se=TRUE,
  method.args=list(family=binomial(link="logit")))
p1
```

なお、geom_smooth で描画できるのは、説明変数が 1 つのモデルです。すでに行ったような、説明変数が複数ある GLM の予測結果を描画したい場合は、ggeffects パッケージを使ってください。

12.5 MCMC 法

R の関数を使えば、このように GLM のパラメータ推定は容易に実行できます。しかし、以下では GLM のパラメータ推定を MCMC 法で行います。

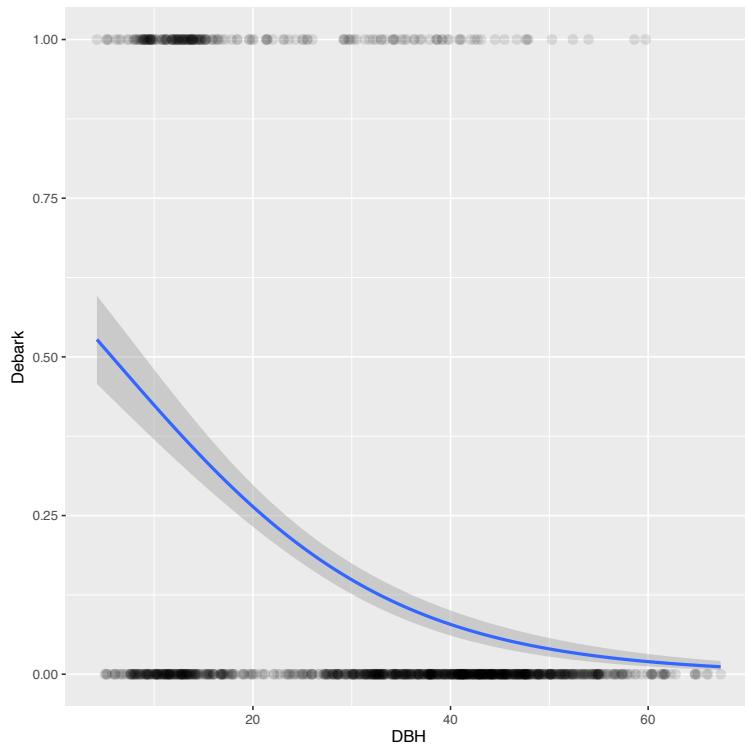


図 31 GLM の予測曲線の描画

12.5.1 MCMC 法の概要

MCMC 法は、確率分布から値をサンプリングするための手法です。求めたい確率分布（ベイズ統計においては事後分布）からの乱数を生成し、得られたサンプルから求めたい確率分布を推測します。

MCMC 法は、適当な初期値を与え、そこから値を変化させ、尤度が高い値を優先的にサンプリングします。そして、MCMC 法では一つ前にサンプリングされた値に依存して次のサンプリング値が決まりますので、サンプリング値はお互いに独立ではありません。このため、サンプリングは同時に複数行います。それぞれのサンプリングを連鎖と呼びます。

MCMC 法を実行し、もしデータに対して適切なモデルを構築すれば、サンプリングされる値は、各連鎖で同じような値を取るようになります（図 32）。同じような値周辺をサンプリングし続けることで、求めたい確率分布の定常分布が得られます。

MCMC 法は、一つ前にサンプリングされた値に依存して次のサンプリング値が決まるところから、モデルによってはサンプリングされる値が変化しにくいことがあります。また、図 32 からわかるように、サンプリング開始からしばらくは初期値の影響を受けています。そのため、MCMC 法では、以下の値を設定する必要があります。

- 連鎖の数。
- 初期値の影響を受けているので破棄すべきサンプリング数 (burn-in)。
- サンプリングする間隔 (thining)。

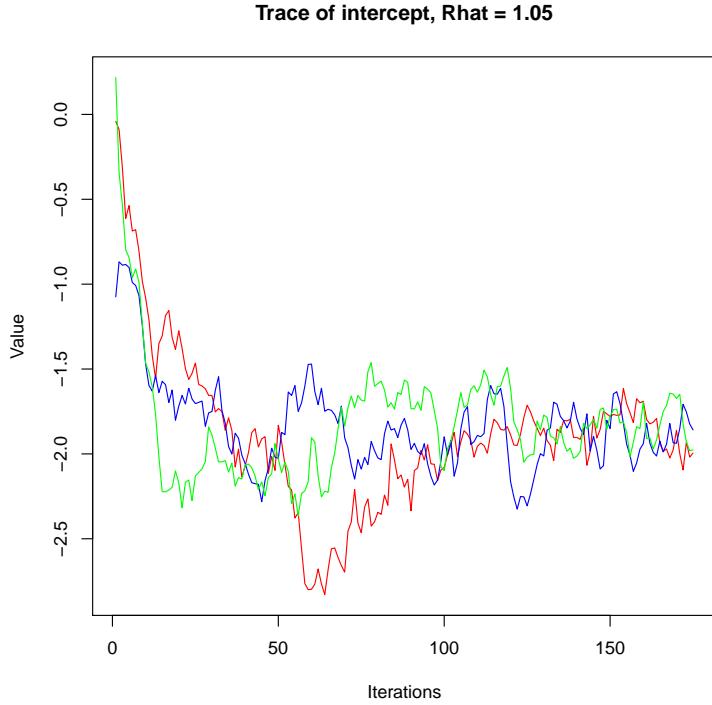


図 32 MCMC 法によるサンプリングの過程

定常分布が得られたのかを判断する指標として、 \hat{R} (R 上では R hat と表記されます) 値があります。 \hat{R} 値は、サンプリング値の連鎖間のばらつきをサンプリング値の連鎖内のはらつきで割ったものです。 \hat{R} が 1.1 未満であれば、パラメータは収束したと判断できます (Gelman et al. 2004)。

- Gelman A, Carlin JB, Stern HS, Rubin DB (2004) Bayesian Data Analysis. Chapman & Hall, New York.

ただし、 \hat{R} 値は上記の通り、連鎖間のサンプリング値の乖離度しか評価していません。このため、サンプリング値が上昇、あるいは減少傾向にないかを判断するため、サンプリングの過程を目視で確認する必要があります。

12.5.2 MCMC 法の実行ソフトウェア

MCMC 法を実行できるソフトウェアには、主に以下のものがあります（表 4）。NIMBLE は、Stan と同様にモデルを内部で C++ 言語に変換するので計算が早い一方、初学者にも理解しやすい BUGS 言語でモデルが記述できるので、今回は NIMBLE を使います。

JAGS と NIMBLE は、BUGS 言語の部分はほぼ完全に互換性があり、かつ NIMBLE の方がより多くの組み込み関数が使えます。データを準備するところで、JAGS はすべてのデータを单一のリスト形式で用意する一方、NIMBLE は後述するように定数とデータを分けて用意するところが異なります。また、MCMC 法の実行関数も異なっています。

表 4 MCMC 法の実行ソフト

	JAGS	NIMBLE	Stan
利用可能 OS	Windows、Mac、Linux	Windows、Mac、Linux	Windows、Mac、Linux
R のパッケージ	<code>rjags</code> など	<code>nimble</code>	<code>rstan</code>
長所	並列化が容易 (<code>jagsUI</code> パッケージ)。	計算が早い。空間相関を推定可能。	収束が早い。エラー箇所の特定が容易。空間相関を推定可能。並列化が容易。
短所	計算が遅い	並列計算は自分で並列関数で定義する必要がある	離散パラメータを推定できない。変数宣言などの規則が少し難しい。

12.5.3 パラメータ推定の手順

NIMBLE によるパラメータ推定を実行する基本的な手順は、以下の通りです。

- BUGS 言語によるモデルの記述
- データの準備
- 初期値の設定
- 監視対象パラメータの設定
- MCMC 法の条件設定
- MCMC 法の実行
- 結果の検証

12.6 BUGS 言語によるモデルの記述

NIMBLE では、BUGS 言語という言語でモデルを記述する必要があります。BUGS 言語は R 言語と非常によく似ており、初学者でも習得しやすい言語だと思います。

では、上記の GLM を BUGS 言語で記述した例を、以下に示します。

```

library(nimble)
modelcode <- nimbleCode(
{
#BUGS 言語でモデルを記述する
for (i in 1:N) {
  # 決定論的モデル (切片 +Deer+DBH)
  # とリンク関数 (log)
  logit(p[i]) <- intercept + bx1*Deer[i] + bx2*DBH[i]
  # p[i] <- 1/(1+exp(-(intercept + bx1*Deer[i] + bx2*DBH[i]))) としても同じ
  # 確率論的モデル (二項分布)
  Debark[i] ~ dbin(p[i], 1)
  # Debark[i] ~ dbern(p[i]) でも同じ
}
# パラメータの事前分布
intercept ~ dnorm(0.0, 1.0E-3) # 切片
bx1 ~ dnorm(0.0, 1.0E-3)      # 説明変数 Deer の係数
bx2 ~ dnorm(0.0, 1.0E-3)      # 説明変数 DBH の係数
}                                # モデルの記述はここまで
)

```

この例をもとに、BUGS 言語の基本を学びます。なお、`nimbleCode` 関数は、その内部に書かれた BUGS 言語によるモデルを NIMBLE が認識できるように翻訳してくれる関数です。

12.6.1 deterministic node と stochastic node

BUGS 言語では、単なる代入部分は deterministic node (`<-`) で記述し、所与の確率分布に従う不確実性を含む部分は stochastic node (`~`) で記述します。例えば、GLM の決定論的モデル（説明変数の線型結合部分）は、以下のように表現されています。

```
logit(p[i]) <- intercept + bx1*Deer[i] + bx2*DBH[i]
```

一方、観測データが得られる過程は確率論的モデルで以下のように表現されています。

```
Debark[i] ~ dbin(p[i], 1)
```

12.6.2 確率分布

BUGS 言語では、以下のように、種々の確率分布を指定できます。先頭に必ず `d` がつきます。

- `dnorm(平均, 分散の逆数)` 正規分布
- `dbern(生起確率)` ベルヌーイ分布
- `dbin(生起確率, 総試行回数)` 二項分布
- `dpois(平均)` ポアソン分布
- `dunif(下限, 上限)` 一様分布

ほとんどは R の乱数生成関数と同じですが、正規分布は分散 ($= \sigma^2$) の「逆数」を指定するところだけ注意が必要です。

ここで紹介した分布以外にも、さまざまな確率分布が用意されています。詳細は、NIMBLE のマニュアルを確認してください。

12.6.3 事前分布

すでに述べたように、ベイズ統計では全てのパラメータに事前分布が必要となります。特に事前の情報がない場合、情報があまりない漠然事前分布を与えることが 1 つの方法です。漠然事前分布の例としては、非常に裾が広い（分散が大きい）正規分布などが挙げられます（図 33）。

先ほども述べたように、BUGS 言語の正規分布を与える関数は `dnorm(平均, 分散の逆数)` という書き方をするので、分散が大きい = 分散の逆数が小さい、という与え方になるところが、少しややこしいです。上記の例では、`dnorm(0.0, 1.0E-3)` と書いていますが、 $1.0E - 3$ とは 10^{-3} と同じことです。

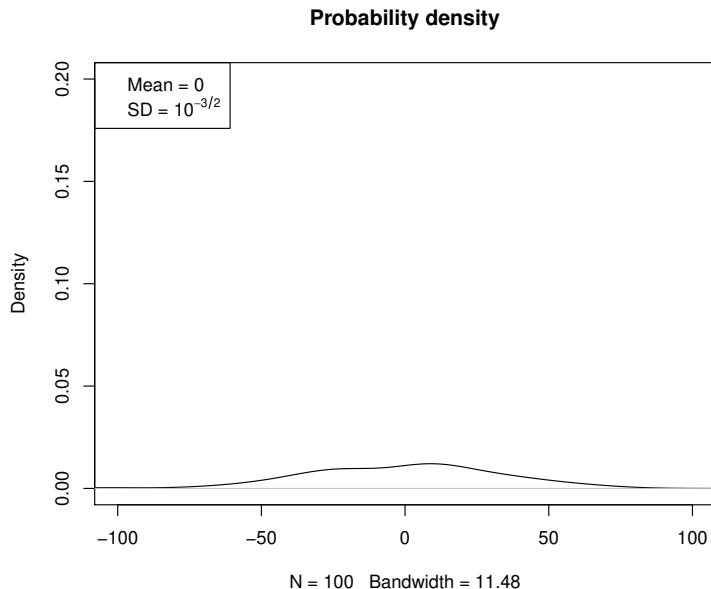


図 33 今回与えた漠然事前分布

今回のモデルでは、切片や環境条件の係数が推定したいパラメータですので、これらのパラメータに事前分布を与えました。事前分布を与えるときは、すでに述べたように stochastic node で与えてください。

12.7 データの準備

NIMBLE では、定数とデータをそれぞれ別のリストとして用意する必要があります。定数とは、データの数や次元数などです。

先ほど BUGS 言語で記述したモデルにおいて定数と言えるのは、データ数を示す N です。そのため、以下のようにして定数とデータのリストを用意します。

```
list_data <- list(Debark=treedf$Debark, Deer=treedf$Deer, DBH=treedf$DBH)
list_cons <- list(N=nrow(treedf))
```

12.8 初期値の設定

パラメータの推定は、MCMC 法で行います。MCMC 法は、すでに説明したように、複数の連鎖を計算します。そのため、推定したい全てのパラメータについて初期値を与える必要があります。そして、複数の独立した計算を行いますので、その数だけ初期値を設定する必要があります。以下は、3 つの独立した計算を行う場合の、初期値の設定例です。

```
set.seed(1)

init1 <- list(intercept=0, bx1=0, bx2=0) # 連鎖 1 の初期値
init2 <- list(intercept=-0.01, bx1=-0.01, bx2=-0.01) # 連鎖 2 の初期値
init3 <- list(intercept=0.01, bx1=0.01, bx2=0.01) # 連鎖 3 の初期値
inits <- list(init1, init2, init3)
```

最初の行の `set.seed()` は、乱数の種を指定する関数です。この数字を固定することで、同じ結果を得ることが可能になります。

12.9 監視対象パラメータの設定

推定したパラメータについて、結果を見たいパラメータ名は、明示的に指定する必要があります。

```
parameters <- c("intercept", "bx1", "bx2")
```

12.10 MCMC 法の条件設定

すでに説明したように、MCMC 法では、複数の計算を独立に行い、事後分布を得ます。そのため、連鎖の数を決める必要があります。また、MCMC 法は 1 つ前に得た値の影響を受けるので、計算開始から一定範囲の計算結果は破棄し（burn-in）、また本計算に入った後でも一定間隔で値を間引く（thin）必要があります。そのため、以下の設定が必要です。

`n.chains` 連鎖の数。通常 3。
`n.update` burn-in の回数。パラメータが動きにくいモデルやデータが多いモデルでは多くの回数が必要。
`n.iter` 本計算の回数。
`thin` 値を間引く間隔。パラメータが動きにくいモデルでは間隔を大きくした方がいい。

では、これらの項目を設定します。

```
nc <- 3
nb <- 2000
ni <- 5000
nt <- 3
```

12.11 MCMC 法の実行

MCMC 法は、`nimbleMCMC` 関数で実行できます。

```
out <- nimbleMCMC(code=modelcode,
                     data=list_data,
                     constants=list_cons,
                     inits=inits,
                     monitors=parameters,
                     niter = ni,
                     nburnin = nb,
                     thin = nt,
                     nchains = nc,
                     progressBar = TRUE,
                     samplesAsCodaMCMC = TRUE,
                     summary = TRUE,
                     WAIC = FALSE
    )
```

引数の意味は、それぞれ以下の通りです。

`data` データを含むリスト
`constants` 定数値を含むリスト
`inits` 初期値を含むリスト
`monitors` 監視対象パラメータ
`niter` 本計算の回数
`nburnin` 計算開始から破棄する計算回数
`thin` 本計算においてサンプリングする間隔
`nchains` 連鎖の数
`progressBar` 計算の進行状況を表示するか
`sampleAsCodaMCMC` coda パッケージで読める形でサンプリング値をまとめるか
`summary` 統計要約量を計算するか
`WAIC` WAIC を計算するか

上記の関数を実行すると、以下のように進行します。

```
Defining model
Building model
Setting data and initial values
Running calculate on model
 [Note] Any error reports that follow may simply reflect missing values in model variables.
```

```

Checking model sizes and dimensions
Checking model calculations
Compiling
  [Note] This may take a minute.
  [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.
Running chain 1 ...
|-----|-----|-----|-----|
|-----|
Running chain 2 ...
|-----|-----|-----|-----|
|-----|
Running chain 3 ...
|-----|-----|-----|-----|
|-----|

```

12.12 結果の検証

さて、MCMC 法によって事後標本が得られました。しかし、事後分布が収束しているかなど、妥当な結果が得られているか確認する必要があります。ひとまず、事後標本から要約統計量を算出しましょう。事後要約量を算出するには、以下のようにします。

```

library(coda)
resdf <- as.data.frame(out$summary$all.chain)
GR.diag <- gelman.diag(out$samples, multivariate = FALSE)
resdf$Rhat <- GR.diag$psrf[, "Point est."]

```

結果のオブジェクトには、以下のようにさまざまな情報が含まれています。

`summary` 事後要約量が、連鎖ごとと全連鎖について計算された結果
`samples` 1つ1つの MCMC 標本が格納されている。`samples[, "パラメータ名"]` とすることで、個別のパラメータの MCMC 標本を抽出することも可能。事後分布を図で示す際に必要となる。

得られた事後要約量を表示すると、平均値、中央値、標準偏差、95% 信用区間、 \hat{R} 値が含まれています。

	Mean	Median	St.Dev.	95%CI_low	95%CI_upp	Rhat
bx1	0.002442375	0.002459484	0.001163498	4.871825e-06	0.004697643	1.005391
bx2	-0.077917868	-0.078168811	0.006790872	-9.120089e-02	-0.064787325	1.001096
intercept	0.140561541	0.135729538	0.210188647	-2.493864e-01	0.549282608	1.001604

収束判定の指標として、 \hat{R} 値が 1.1 未満となることが提案されています (Gelman et al. 2004)。今回の結果を見ると、全てのパラメータが収束しているようです。

次に、サンプリング値を目視でも確認しましょう。以下のように入力することで、図 34 が得られます。ここで出てくる bayesplot パッケージは、名前の通り MCMC の結果の作図を容易にしてくれるパッケージです。

```
library(bayesplot)
mcmc_trace(out$samples) +
  scale_color_discrete()
# 他にもさまざまな図を描画できる
mcmc_hist(out$samples)
mcmc_intervals(out$samples)
mcmc_dens(out$samples)
```

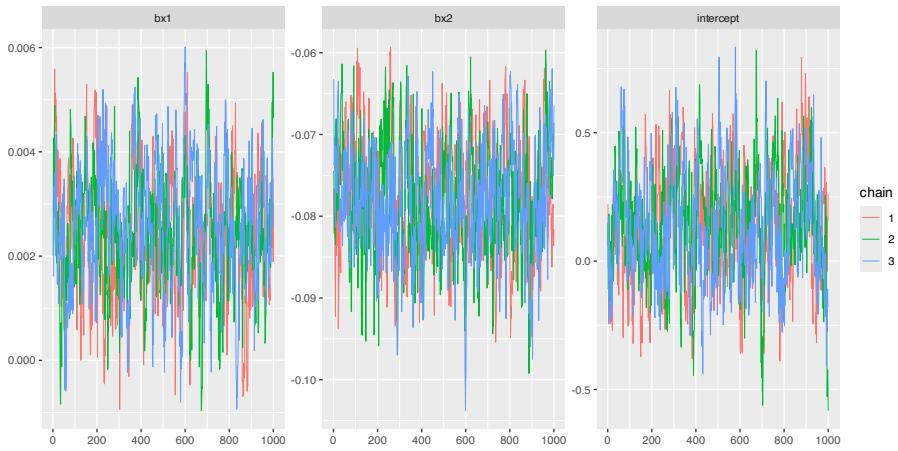


図 34 MCMC の収束状況

MCMC の各連鎖が同じような値をサンプリングしており、かつ増減傾向もありませんので、定常分布に達したと判断できます。

では最後に、`glm()` による推定（最尤法）結果と比較してみましょう。すべてのパラメータが、最尤法とほぼ同じ値となっています（図 35）。漠然事前分布を用いた場合、一般的に MCMC 法による推定結果は最尤法とほぼ一致します。

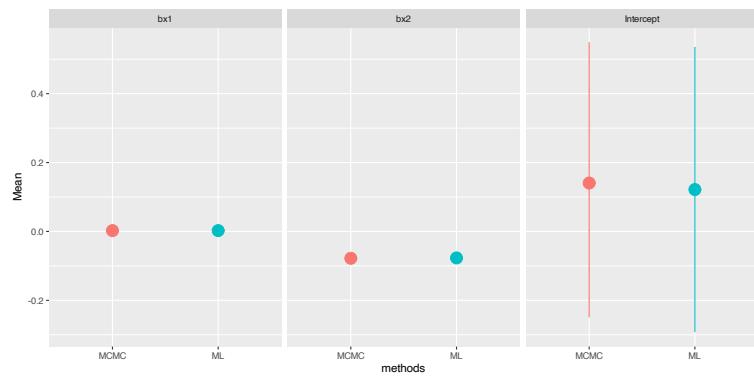


図 35 パラメータの事後分布

13 GLMM（一般化線形混合モデル）

GLMM とは、Generalized Linear Mixed Model の略です。GLM との違いは、変量効果がモデルに含まれると言う意味の Mixed が加わったことです。GLMM の基本的な構造は、以下のとおりです。

$$\eta = \beta X^T + \epsilon \quad (4)$$

$$\epsilon \sim N(0, \sigma^2) \quad (5)$$

$$\eta = g(\mu)$$

$$\text{Data} \sim \text{Probability distribution}(\mu)$$

式 4 に変量効果 ϵ が含まれています。 ϵ の実態は式 5 に示されているとおり、正規分布に従う確率変数です（必ずしも正規分布である必要はありませんが、正規分布が使われることが多いです）。

13.1 変量効果とは何か？

変量効果に用いるのは、カテゴリ変数です。カテゴリ変数を固定効果として扱うこともあると思います。その違いは、以下のとおりです。

$$\beta \sim N(0, 10^3) \quad \text{固定効果: 分散は非常に大きい値}$$

$$\epsilon \sim N(0, \sigma^2) \quad \text{変量効果: 分散は sigma*sigma の値}$$

この違いから分かるように、固定効果はカテゴリごとの係数をカテゴリごとに独立に推定する（漠然事前分布を与える）のに対し、変量効果は σ の値によってはカテゴリ間で類似した値を取る可能性があります。ある要因について固定効果とするか変量効果とするかは、解析者の判断によるところが大きいです。

ただし、例えばある薬品処理を行う区と行わない区といったように明確に区別され、かつそれぞれで十分な標本数が得られている場合は、固定効果とするべきです。一方、水準や処理の区別がそれほど厳密でなく、ある水準と別の水準の効果は類似しているが同一ではない場合は、変量効果とする方が適切と考えられます。

変量効果を用いる利点は複数ありますが、飯島は特に以下の点で有用と考えます。詳細は、Kéry and Schaub (2016) の 4 章を見て下さい。

擬似反復の回避 ある一つの調査区から複数のデータを得ている場合、その調査区から得られたデータは独立ではないと考えられます。このようなデータについて調査区を変量効果としないと、それらのデータの効果を過剰に見積もってしまいます。

標本数が少ない水準の推定 変量効果はある共通の分布からある程度の分散を持った値の集合として推定されますので、標本数が少なく極端なデータとなっている水準の効果を全体の平均に近づけて推定できます（説得力の借用）。

一方、変量効果の集合は分布として推定されるため分布の分散パラメータを推定する必要があります、一般に計算負荷が大きくなります。

13.1.1 今回のデータに潜む観測誤差

I 部で説明したように、1 つの Stand 内で複数の立木の剥皮の調査を行っています。また、1 つの Region に 2 つの Stand を設置しています。そのため、Stand や Region は疑似反復の回避という意味での変量効果に該

当すると考えられます。

一方、毎木データには複数の樹種が含まれますが、樹種によって調査個体数は大きく異なっています。個体数が少ない樹種については、剥皮のされやすさを推定することは困難かもしれません。そのため、Species は表本数が少ない水準の推定という意味での変量効果に該当すると考えられます。

以下では変量効果の考え方と BUGS 言語での実装を習得するという目的のため、Species のみを変量効果とした単純な GLMM のパラメータ推定を行います。

13.2 R の関数による解析

R では、GLMM を実行するための様々な関数が用意されています。ここでは、glmmTMB パッケージの glmmTMB 関数を使います。glmmTMB 関数では、切片の変量効果を (1|変量効果) で指定します。比較のため、Species を固定効果とした GLM と比較してみましょう。

```
res1a <- glm(Debark ~ Deer + DBH + Species, family=binomial, treedf)
summary(res1a)

Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.201300 0.591527 -2.031 0.042270 *
Deer          0.004110 0.001352  3.041 0.002358 **
DBH          -0.108778 0.009164 -11.870 < 2e-16 ***
SpeciesGenus_speciesB -1.129409 1.160774 -0.973 0.330564
SpeciesGenus_speciesC  0.654020 0.742138  0.881 0.378175
SpeciesGenus_speciesD -15.455610 735.296583 -0.021 0.983230
(途中省略)

library(glmmTMB)
res2 <- glmmTMB(Debark ~ Deer + DBH + (1|Species), family=binomial, treedf)
summary(res2)

Conditional model:
Estimate Std. Error z value Pr(>|z|)
(Intercept) 0.047491 0.462120 0.103 0.91815
Deer        0.003935 0.001332 2.955 0.00312 **
DBH        -0.106106 0.008990 -11.802 < 2e-16 ***
(途中省略)

ranef(res2)
$Species
(Intercept)
Genus_speciesA -1.1398675
```

```

Genus_speciesB -1.8663019
Genus_speciesC -0.5668741
Genus_speciesD -2.5862356
(途中省略)

```

Species を固定効果として扱った場合、上記の例では種 D の係数の標準誤差が非常に大きくなっています。一方、変量効果として扱った場合、他の種の係数と類似した値となっており、適切な推定が行えているように見えます。では、この GLMM のパラメータ推定を NIMBLE で実行してみましょう。

13.3 BUGS 言語によるモデルの記述

GLM の場合と異なっているのは、当然ですが変量効果の指定の部分です。変量効果を含むモデルは、BUGS 言語で以下のように記述できます。

```

library(nimble)
modelcode <- nimbleCode(
{
  for (i in 1:N) {
    # 決定論的モデル (切片 +Deer+DBH+ 変量効果)
    # とリンク関数 (logit)
    logit(p[i]) <- intercept + bx1*Deer[i] + bx2*DBH[i] + bSP[Species[i]]
    # bSP は変量効果のパラメータ
    # 確率論的モデル (ベルヌーイ分布)
    Debark[i] ~ dbern(p[i])
  }
  #/パラメータの事前分布
  intercept ~ dnorm(0.0, 1.0E-3) # 切片
  bx1 ~ dnorm(0.0, 1.0E-3)       # Deer の係数
  bx2 ~ dnorm(0.0, 1.0E-3)       # DBH の係数
  # 変量効果の事前分布
  for (j in 1:Nsp) {             # 種数だけ繰り返す
    bSP[j] ~ dnorm(0.0, tau)    # 分散 (sigma*sigma) の逆数を指定
  }
  tau <- pow(sigma, -2)          # 標準偏差 (sigma) を tau に変換
  sigma ~ dunif(0, 100)          # sigma の事前分布
}
)                                # モデルの記述はここまで

```

13.3.1 変量効果の実装

`ranef` が変量効果（今回のデータではプロットの効果）です。変量効果の事前分布は正規分布ですが、分散の逆数が `tau` という値に制限されています。固定効果であれば、`tau` の値を非常に小さくして（つまり分散を非常に大きくして）、どのような値でも取り得るようにします。変量効果の場合は、分散をある程度制約することで、取り得る値に制約を与えています。

では、この `tau` の値をどの程度にすればいいのでしょうか。これは、漠然事前分布を与えてデータから推定することができます。上記の例では、標準偏差 ($= \sigma$) に範囲が広い一様分布を与えています。そして、標準偏差を 2 乗して逆数を取ることで、`tau` としています。以前は正規分布の超パラメータとして逆ガンマ分布が広く用いられていましたが、分散の真の値が小さい場合、逆ガンマ分布は推定結果に偏りをもたらすことが明らかにされており (Gelman 2006)、現在では一様分布を使う事が一般的です。

- Gelman A (2006) Prior distributions for variance parameters in hierarchical models. *Bayesian Analysis* 1: 515-534.

13.4 データの準備

```
# データと定数の準備
```

```
list_data <- list(Deer=treedf$Deer, DBH=treedf$DBH, Debark=treedf$Debark)
list_cons <- list(N=nrow(treedf), Nsp=length(unique(treedf$Species)),
  Species=as.numeric(as.factor(treedf$Species)))
```

13.5 初期値の設定

```
set.seed(2)
init1 <- list(intercept=0, bx1=0, bx2=0, bSP=rnorm(list_cons$Nsp, 0, 0.1),
  sigma=5)
init2 <- list(intercept=-0.01, bx1=-0.01, bx2=-0.01,
  bSP=rnorm(list_cons$Nsp, -0.01, 0.1), sigma=1)
init3 <- list(intercept=0.01, bx1=0.01, bx2=0.01,
  bSP=rnorm(list_cons$Nsp, 0.01, 0.11), sigma=10)
inits <- list(init1, init2, init3)
```

13.6 監視対象パラメータの設定

```
parameters <- c("intercept", "bx1", "bx2", "bSP", "sigma")
```

13.7 MCMC 法の条件設定

```
nc <- 3
```

```

nb <- 2000
ni <- 5000
nt <- 3

```

13.8 MCMC 法の実行

```

out2 <- nimbleMCMC(code=modelcode,
                      data=list_data,
                      constants=list_cons,
                      inits=inits,
                      monitors=parameters,
                      niter = ni,
                      nburnin = nb,
                      thin = nt,
                      nchains = nc,
                      progressBar = TRUE,
                      samplesAsCodaMCMC = TRUE,
                      summary = TRUE,
                      WAIC = FALSE
)

```

13.9 結果の検証

```

# 要約量の計算
library(coda)
resdf <- as.data.frame(out2$summary$all.chain)
GR.diag <- gelman.diag(out2$samples, multivariate = FALSE)
resdf$Rhat <- GR.diag$psrf[, "Point est."]
resdf

```

	Mean	Median	St.Dev.	95%CI_low	95%CI_upp	Rhat
bSP[1]	-1.244750681	-1.218136760	0.675782651	-2.58130056	0.009635721	1.082530
bSP[2]	-2.095838277	-2.035407397	0.882912392	-3.95642597	-0.547330724	1.027670
bSP[3]	-0.637357250	-0.647378004	0.663687497	-1.95982069	0.606859838	1.152375

(以下、省略)

結果を見ると、一部のパラメータの \hat{R} が 1.1 以上となり、収束していないようです。

13.10 収束しないときの対処法

MCMC が収束しないことは、よくあります。その際は、まず MCMC の挙動を確認することが基本になります。図を描いてみて、状況に応じた対応を取る必要があります。今回、 \hat{R} が一番大きかった切片 (intercept) の挙動を見てみます。

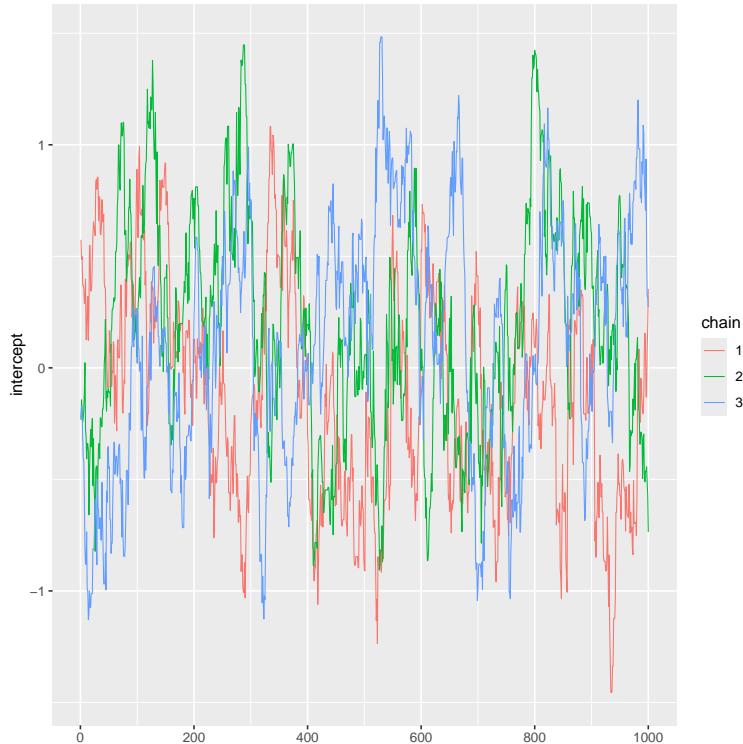


図 36 パラメータ intercept の事後分布

図から何を読み取るかですが、burnin 後の値は一定の範囲を取り続けているようですが、連鎖の混交が十分でないよう見えます（図 36）。収束しない場合の連鎖の挙動と対策は、基本的には以下のとおりです。

連鎖の軌跡が左と右で一致していない 初期値の影響が残っている可能性があります。burnin の回数を増やしましょう。

連鎖の軌跡は一定だが連鎖同士が混ざっていない 計算回数を増やす、間引きの間隔をより大きく取る。

今回は後者のようですので、burnin はそのままにして、計算回数を増やし、間引き間隔を広げてみましょう。

```
nc <- 3
nb <- 2000
ni <- 12000
nt <- 10
(後は同じ)
```

結果を見ると、全てのパラメータは収束したようです。切片のパラメータの連鎖も、よく混交しています（図 37）。

```
resdf
```

	Mean	Median	St.Dev.	95%CI_low	95%CI_upp	Rhat
bSP[1]	-1.205358885	-1.18052525	0.663275766	-2.561084533	-0.015810994	1.000310
bSP[2]	-2.108601339	-2.04725115	0.908666351	-4.120359737	-0.508786307	1.002011
bSP[3]	-0.598346179	-0.57446324	0.624920573	-1.834556363	0.590748980	1.000793

（以下、省略）

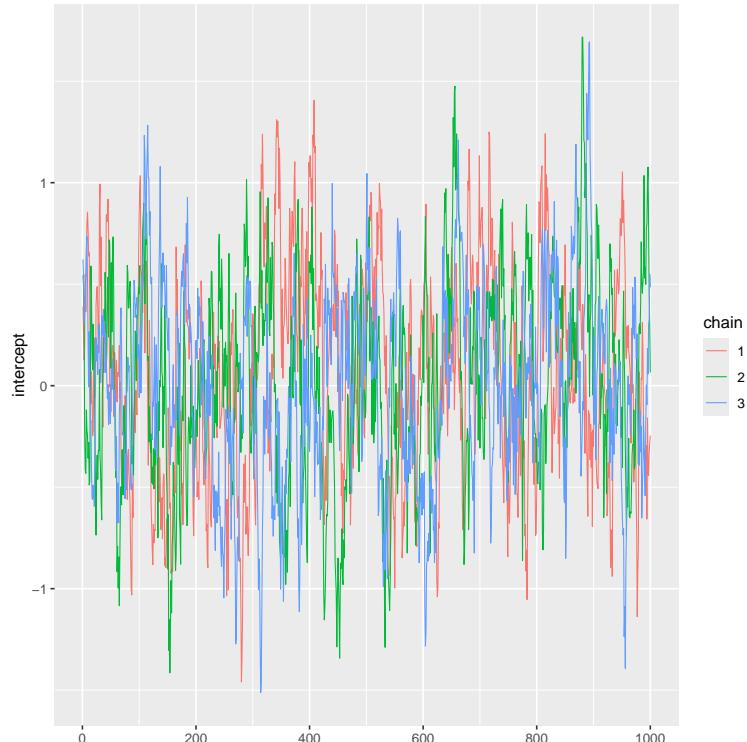


図 37 再計算後のパラメータ intercept の事後分布

では、結果が収束しましたので、glmmTMB の推定結果と比べてみましょう。固定効果、変量効果共に、ほぼ同じ値となっているようです（図 38）。

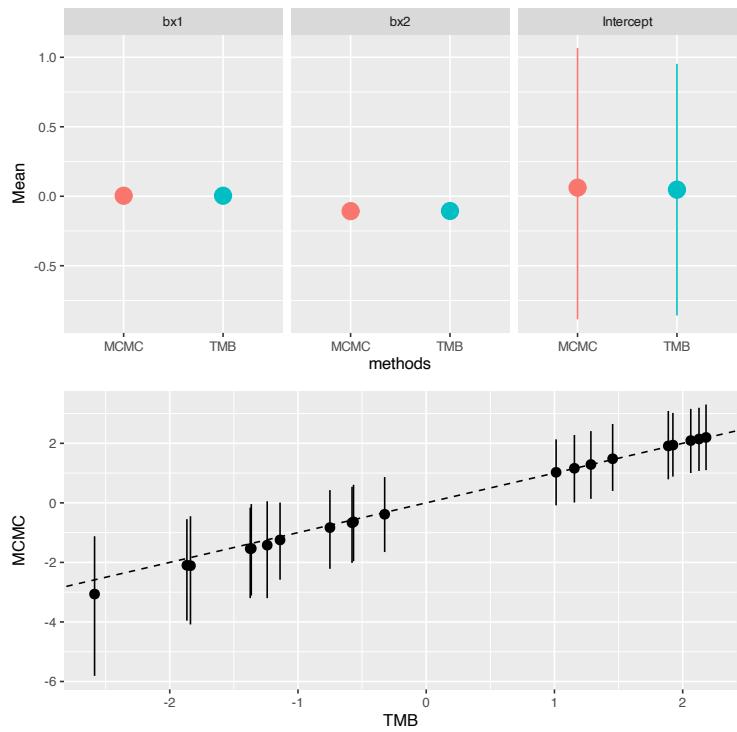


図 38 推定方法ごとのパラメータの事後分布

第 IV 部

Rmarkdown による簡易レポート作成

14 はじめに

R を使って図を書いたり解析を行った後、通常それらは論文や報告書などに挿入する必要があります。しかし、図をわざわざ貼り直したりするのは手間ですし、R で出力された推定値を目で見ながら書き写すと間違が生じるかもしれません。

近年、マークダウンと呼ばれる記法が普及しています。これは、細かい装飾などの機能を省き、その代わりに簡易な記号で階層構造を持った文書を作成できる記法です。マークダウンは github などで使われていますが、それを R のコードと並列で表記して文章を作れるようにしたものが Rmarkdown です。上記の通り、あまり凝った文章は作成できませんが、R でデータ操作、作図、解析などを行いつつ、その結果がそのまま文章として作成できるので、簡易的なレポートなどを作成には有用です。

Rmarkdown は、Pandoc というシステムを利用し、pdf など様々な形式で文書を出力することができます。ですが、本資料では Word の出力に絞って説明を行います(というか白状すると、飯島はそれほど Rmarkdown に詳しくありません...)。Pandoc は R とは別なので、Pandoc の HP <https://pandoc.org/installing.html> からダウンロード、インストールして下さい。

15 Rmarkdown による文書の基本

15.1 文書作成の手順

Rmarkdown では、以下の手順で文書を出力します。

- 文章や、図などを生成する R のコードを記載したテキストファイルを作成する（拡張子を Rmd とする）
- R で、上記で作成した Rmarkdown ファイルから文章を作る命令を出す

前者のテキストファイルを作成する上で重要なのが、「マークダウン記法」、「チャンク」、「yaml ヘッダ」の理解です。これらについて、順に解説します。

15.2 マークダウン記法

マークダウン記法は、簡単な記号で文書の体裁を整えることを可能にする記法です。主な記号は、以下のとおりです。

- # 見出し記号。この記号の後に書かれた内容は、見出しになる。1~4 個の範囲で使用でき、数が多くなるほどより小さい見出しになる。
- 箇条書き記号。この記号の後に書かれた内容は、箇条書きになる。この記号の前に半角スペースを 4 つ入れると、インデントされる。
- 1. 数字の箇条書き記号。この記号の後に書かれた内容は、数字付きの箇条書きになる。
- \$ 数式記号。この記号に挟まれた部分は数式として評価される。数式の記法は LATEX とほぼ同じ (LATEX については今回は説明しません。すいません)。

```
''' この記号（バックチック ×3）で挟まれた部分は、記述したそのままが出力される。プログラムのコードなどを書くのに便利。
```

15.2.1 見出し

これは非常に便利です。

```
# 材料と方法
```

```
## 調査地
```

と記述すると、以下のように出力されます。

材料と方法

調査地

15.2.2 箇条書き

```
- 材料と方法
```

```
  - 調査地
```

と記述すると、以下のように出力されます。

```
● 材料と方法
```

```
  - 調査地
```

15.2.3 番号付き箇条書き

```
1. 材料と方法
```

```
  1. 調査地
```

と記述すると、以下のように出力されます。

```
1. 材料と方法
```

```
  1-1 調査地
```

15.2.4 数式

```
$y_{\{i\}} \sim \mathrm{N}(\mu_{\{i\}}, \sigma^2)$
```

と記述すると、以下のように出力されます。

$$y_i \sim N(\mu_i, \sigma^2)$$

15.2.5 そのまま表示

```
'''
```

```
y_{\{i\}} ~ dnorm(mu[i], tau)
```

```
'''
```

と記述すると、以下のように出力されます。

```
y_{i} ~ dnorm(mu[i], tau)
```

15.3 チャンク

チャンクとは、R のコードを記述する部分のことです。基本的な記法は、以下の通りです。

—— チャンクの記法 ———

```
'''{r, チャンクの名称}
#ここに R のコードを書く
'''
```

以上の通り、チャンクで囲った部分は R の世界、それ以外の部分はマークダウン記法の文書となります。また、チャンクのヘッダ部分は、以下のいずれかの引数を使うことができます。

`echo` コードを、結果のレポートに表示させるか（コードはこの引数に関わらず実行される）

`eval` コードを実行するか

`include` コードを、結果のレポートに含めるか（`FALSE` にすると、実行されるがコードは表示されず、またその結果生成される図なども表示しない）

`warning` コードを実行した際に R が output する警告を出力するかどうか

ヘッダの設定については、チャンクごとに設定もできますが、よく使う設定については最初のチャンク内で `opts_chunk` 関数で以下のように記述しておくと、全てのチャンクに適用されるので便利です（この例では、R が output するメッセージと警告を表示しないようにしています）。

```
knitr::opts_chunk$set(echo=FALSE, message=FALSE, warning=FALSE)
```

チャンクの名称は、チャンクを識別するのに用います。名称を設定すると、本文中でそのチャンク内で作られた図や表の番号を自動で参照できます。そのため、図や表を生成する部分のコードは、それぞれ独立したチャンクに記述します。具体的には、以下のとおりです。

—— 図を作るチャンクの記法 ———

```
'''{r, チャンクの名称, fig.cap="図の表題"}
#ここに作図する R のコードを書く
'''
```

表を作るチャンクの記法

```
```{r, チャンクの名称}
#ここに表を作る R のコードを書く
knitr::kable(表の元となるデータ, format="pandoc",
 col.names="ここで説明を指定できます",
 digits=出力する数字の桁を指定できます,
 caption = "表題")
```
```

一方、本文中で図や表の番号を参照するためには、図 \eref{fig: 図を生成したチャンクの名称}、表 \eref{tab: 表を生成したチャンクの名称} と記述します。

15.3.1 チャンク外での R の結果の参照

‘r ここに R のコードなり変数とか’を使います。‘は’（クオーテーション）と似ていますが、バックチックという別の記号ですので、注意して下さい。使い所としては、R で推定された係数や個体数推定値などを本文中に記載するといったことが考えられます。

15.4 yaml ヘッダ

こちらは文章の内容ではなく、出力するファイルの形式などに関する「設定」です。上記の通り、ここでは Word ファイルを出力するために必要な点に絞って解説します。

名前にあるように、中身の文章を書く前の一番上に記載します。簡単な例を、以下に示します。

```
---
title: "rmarkdown による簡易レポート"
author: "飯島 勇人"
date: "‘r format(Sys.time(), '%Y/%m/%d')’"
output:
  bookdown::word_document2:
    number_sections: false
---
```

output のところで、出力するファイル形式を選択します。Word で出力する場合は `word_document` なのですが、上記のように相互参照の機能を使うための `bookdown` パッケージを使うので、それ専用の形式を指定します。また、標準では見出しに番号が振られるので、`number_sections: false` で番号が振られないようになっています。

先頭の半角空白 2 つはインデントで、これがないと正しく認識されないので注意して下さい。

15.5 文書の出力方法

上記を参考に、自分で作成した内容をテキストファイルなどに打ち込み、拡張子を `.Rmd` として保存します（サンプルファイル `report.Rmd` を配布します）。注意点として、Rmd ファイル内で R にデータを読ませたり

作図したりしますが、そのための作業ディレクトリの指定などは Rmd に書かれた内容で正確に動作しなければならないということです。そのため、Rmd に限っては、作業ディレクトリの指定は、コマンドで行う必要があります。

上記の点を確認できたら、R を立ち上げ、以下のコードを走らせることで、文書が出力されます。

Rmd ファイルから文書を作るコード

```
library(rmarkdown)
library(bookdown)
library(knitr)
render(input="report.Rmd")
```

上記の例では report.Rmd までのパスを明示的に指定していませんが、ホームディレクトリ以外にある場合は、R を立ち上げた後に作業ディレクトリを Rmd ファイルのある場所に変更するか、`render` 関数でファイルまでのフルパスを指定して下さい。

第 V 部

演習の答え

I 部の演習

演習 1：プロットかつ種ごとの出現個体数の計算

```
treedf %>%
  group_by(Region, Stand, Species) %>%
  reframe(個体数=n())
```

演習 2：個体ごとの段面積の計算

```
treedf %>%
  mutate(断面積=3.14*(DBH/100/2)^2)
```

演習 3：毎木プロットごとの胸高断面積合計の計算

```
treedf %>%
  mutate(断面積=3.14*(DBH/100/2)^2) %>%
  group_by(Region, Stand) %>%
  reframe(断面積合計=sum(断面積)/(20*20)*100*100)
```

演習 4：断面積を計算して新しい列として付与する

```
treedf <- treedf %>%
  mutate(断面積=(DBH/2)^{2}*pi)
```

演習 5：種ごとの平均 DBH を計算する

```
treedf %>%
  group_by(Species) %>%
  reframe(平均 DBH=mean(DBH))
```

演習 6：Region と Stand の結合

```
treedf %>%
  mutate(Plotid=str_c(Region, Stand, sep="_"))
```

演習 7：文字列の置換

```
treedf %>%
  mutate(Sp2=str_replace(Species, "Genus_species", "")) %>%
```

```
# 以下は、単に表示のためで、本来は不要  
select(Species, Sp2)
```

演習 8：一連のひらがなの抽出

```
str_extract(char1, "\\p{Script=Hiragana}+")
```

演習 9：全角から半角

```
stri_trans_general(J_text, "Fullwidth-Halfwidth")
```

演習 10：Stand をキーにしてデータを結合する

```
treedf <- treedf %>%  
  inner_join(., standinfo, by=c("Region", "Stand"))
```

演習 11：ニホンジカの撮影枚数を Region 単位で集約する

```
deerphoto <- cam_df %>%  
  filter(str_detect(species, "deer")) %>%  
  group_by(Region) %>%  
  reframe(Deer=sum(count))
```

演習 12：一致データのみを結合

```
treedf <- treedf %>%  
  inner_join(., deerphoto, by="Region")
```

II 部の演習

演習 13：一致データのみを結合

```
standloc %>%  
  mutate(lon=st_coordinates(.)[,1], lat=st_coordinates(.)[,2])
```

演習 14：座標参照系の変換

```
yama <- yama %>%  
  st_transform(., crs=6676)  
standloc <- standloc %>%  
  st_transform(., crs=6676)
```

演習 15：半径 500m のバッファを作る

```
standloc2 <- standloc %>%
  st_transform(., crs=6676) %>%
  st_buffer(., dist=500)
```

演習 16：座標参照系を 6676 に変更

```
lu <- lu %>%
  st_transform(., crs=6676)
```

演習 17：ベクタファイル同士の結合

```
standloc2 <- standloc2 %>%
  st_intersection(., lu)
```

演習 18：土地利用種ごとのデータ数の計算

```
landuseratio <- standloc2 %>%
  group_by(Region, Stand, lu) %>%
  reframe(Landuse=n())
```

演習 19：プロットごとの土地利用種の割合の計算

```
landuseratio <- landuseratio %>%
  group_by(Region, Stand) %>%
  mutate(LanduseRatio=Landuse/sum(Landuse))
```

III 部の演習

演習 20：正規乱数の生成

```
hist(rnorm(10000, -1, 3))
```

演習 21：二項乱数の生成

```
rbinom(100, 1, 0.5)
```

演習 22：ポアソン乱数の生成

```
rpois(100, 5)
```

索引

across(), 34
annotation_map_tile(), 60
as.data.frame(), 19
as.list(), 19
as.matrix(), 19
as_tibble(), 19

bayesplot パッケージ, 88
bind_rows(), 31
bookdown パッケージ, 102
BUGS 言語, 82

case_when(), 26
coda パッケージ, 87
contains(), 34

data.frame(), 19
EPSG コード, 51

facet_wrap(), 42
file.path(), 7
for(), 29
full_join(), 32

gelman.diag(), 87
geom_bar(), 35
geom_boxplot(), 36
geom_histogram(), 36
geom_point(), 35
geom_sf(), 59
geom_violin(), 36
ggplot2 パッケージ, 35
ggsave(), 48
ggspatial パッケージ, 60
GLM (一般化線形モデル), 74
GLMM (一般化線形混合モデル), 90
glmmTMB(), 91
glmmTMB パッケージ, 91
GR.diag(), 87
group_by(), 11, 20

here(), 7
here パッケージ, 7

if_else(), 26
inner_join(), 32
is.na(), 28

JAGS, 82

knitr パッケージ, 102

labs(), 44
leaflet パッケージ, 61
leaflet.minicharts(), 61
leaflet.minicharts パッケージ, 61
left_join(), 32
length(), 20
list(), 19

map_db1(), 21
matrix(), 19

max(), 20
MCMC (マルコフ連鎖モンテカルロ) 法, 85
mean(), 19
min(), 20
mosaic(), 54
mutate(), 20

n(), 12
NA, 27
NIMBLE, 82
nimble パッケージ, 83
nimbleCode, 83
nimbleMCMC(), 86

opts_chunk(), 100

patchwork パッケージ, 42
path.expand(), 7

read_excel(), 9
read_sf(), 51
read_stars(), 53
readxl パッケージ, 8
reframe(), 11, 21
render(), 102
 \hat{R} , 81
right_join(), 32
rmarkdown パッケージ, 102

scale_x_continuous(), 44
sd(), 19
setwd(), 7
sf パッケージ, 50
Stan, 82
st_area(), 55
stars パッケージ, 50
st_as_sf(), 52
st_buffer(), 54
st_centroid(), 55
st_coordinates(), 55
st_intersection(), 56
str_c(), 22
stringi パッケージ, 25
str_trans_general(), 26
str_replace(), 22
str_to_lower(), 25
str_to_title(), 25
str_to_upper(), 25
st_transform(), 54
sum(), 19
summary(), 20

terra パッケージ, 54
theme_bw(), 47
theme_classic(), 47
tibble(), 19
tidyverse パッケージ, 8

unique(), 20

yaml ヘッダ, 101
オブジェクト, 19

確率分布, 65
確率論的モデル, 74
行列, 19
決定論的モデル, 74
最尤推定値, 71
最尤法, 71
座標参照系, 51
事前分布, 84
正規表現, 23
正規分布, 65
対数尤度, 72
チャンク, 100
ティブル, 19
定常分布, 80
データフレーム, 19
二項分布, 66
漠然事前分布, 84
パイプ演算子, 11
変量効果, 90
ペイズの定理, 63
ベクトル, 19
ポアソン分布, 68
マークダウン記法, 98
尤度, 71
リスト, 19
リンク関数, 74
オブジェクト, 9