RとStanで学ぶフリーで楽しい心理統計の世界

心理学データ解析応用2

データの背後のメカニズムを解析する方法

小杉考司

心理学データ解析応用 2

小杉 考司

Last Compiled on 2024.8.1

昨今はデータサイエンス，情報科学の領域が非常に隆盛で，コンピュータを使ってデータを分析し，経済の動向や購買行動などの予測に用いられることが広く行われている。

人の行動や考え方をどのようにデータにするかについては，当然ながら心理学には一日の長がある。また，人が頭の中でどのような考え方のプロセスをたどるのか，それをどのように検証するのかについても，心理学はその短い歴史の中で徹底的にその技法を洗練させてきた。このような根源的なレベルでの理論や方法論は時代が変わっても色褪せることなく，また今後ますます必要とされてくる時代になっている。

本講ではデータ解析の応用段階として，より実践的なテーマを扱う。すなわち，**心理尺度が作られる理論的背景**と，**データの背後のメカニズムを解析する方法**を知ることである。後期配当の心理学データ解析応用 2(旧カリキュラム名心理学データ解析 2B) では，数値計算を通して実践されている推測統計学的手法の原理・メカニズムを知り，ひいてはデータ生成メカニズムを知る技法を学ぶ。

心理学研究では，調査・実験・観察など様々な手法で得られたデータに対し，統計的な処理で「意味があったかどうか」といった判定を下す。データは知りたい全体 (知的生命) の一部を使って得たものだから，標本の平均値から母集団の平均値を推測する推測統計学の手法が用いられる。推測統計学では確率や微積分の概念が必要であり，高度で抽象的な数学ツールを実感的に理解することは難しい。畢竟，推測統計のメカニズムの理解がおぼつかないまま，ただ機械から出力される数値を確認するだけになりがちである。生兵法は大怪我の基である。そこで，原理の理解度を上げるために確率変数を数値シミュレーションによって具体的な数値にすることで，抽象度を下げて理解することを考えよう。この方法は，ひいては心理的なメカニズムを数式的に表現することで分析する，**数理モデリング**というアプローチにつながる。このモデリングの基礎的な知識，方法論の習得を目指す。

なお，2024 年度から前期と後期の資料を分割して提供するようにした。通年で履修する学生が少ないことと，後期で扱うテーマとして数値シミュレーションを含むように再構成したからである。とくに数値シミュレーションについては拙著「数値シミュレーションで学ぶ統計の仕組み」(小杉他, 2023) としてまとめたので，詳細シラバスの第 2 講から第 8 講に該当する箇所では，この外部テキストを参照してほしい。この資料に含まれるのは小杉他 (2023) のエッセンスと，そこから外れる Stan を用いたモデリング部分である。前期後期でテーマが明確になり，またスリムになった分，利用しやすくなったのではないかと思っているが，どうだろうか。

授業のテーマ

データから意味のある情報を取り出すための，さまざまな分析法を習熟するにあたって，その背後にあって語られることのない「発想」の観点から理解する。数値だけに振り回される状態から脱却し，数値を算出する数式に込められた意味について考える視点を持つ。さらにこれらに習熟することで，どのような研究対象に対してどのような心理統計的アプローチができるかを，俯瞰的に見られるようになろう。

一年を通じて伝えたいポイント

乱数生成による確率の理解　ランダムであることを概念的に理解することは難しい。一方で，偶然生じた事象は身の回りに溢れている。確率変数から生じた乱数を用いることで，確率を身近に理解し，推測統計学の理解を深める。

データ生成メカニズム　乱数を用いて，データが生まれてくるメカニズムを考えるトレーニングを行う。データからメカニズムを再現するアプローチは，心理学的事象の深い理解につながる。

統計環境 R と確率的プログラミング言語 Stan による実践　統計環境 R と確率的プログラミング言語 Stan に習熟することで実際に計算し，確認しながら分析を進めることができるようになる。

その他

授業シラバスとこの講義資料を掲載したサイト (https://kosugitti.github.io/psychometrics_syllabus/) で，最新版のシラバスと授業資料，授業で用いるサンプルデータやコードの配布を行なっています。

目次

第 1 章

プログラミングの基礎

この授業は，R や Stan といったプログラミング言語を扱います。プログラミングはやったことがない，という人でも理解できるように，ゆっくり話を進めていきます。プログラミングのいいところは，人間相手とのやり取りと違って，必ず理屈があって振る舞いが生じている点です。対話を通じて根気よく付き合えば，必ず問題は解決できます。また昨今では，生成 AI がプログラミングをアシストしてくれるようになりました。この授業は TA がついていますから，わからないところがあれば，随時 TA に尋ねてくだされば結構です。それでも人間相手の質問ですから，「こんなことを聞いたらバカだと思われるんじゃないだろうか」「声をかけるのが恥ずかしい」ということがあるかもしれません。でも生成 AI とのやり取りは，いつでも気軽に行えます。また決してこちらを蔑んだり恨んだりすることはありません (というかあっても怖くありません！)。同じ質問を何度しても，生成 AI は一向に意に介さないので，積極的に利用してください。ただし生成 AI は正しい答えを教えてくれるとは限りませんので，**必ず自分が確認できるような質問をしてください**。何かわからないけど，バグが出ないからいいや，と思っていると，自分がなんの成長もしないのはもちろん，大変な問題を引き起こすかもしれませんので。

1.1 パソコンの基礎

21 世紀に生きる私たちは身の回りをコンピュータに囲まれて生きています[*1]。それは携帯電話の形をしていたり，ノートパソコン，デスクトップパソコンの形をしていたりします。また時々テレビなどで報道されますが，気象予報や飛沫がどのように飛び散るかをシミュレーションする大型計算機「富嶽」などもコンピュータですね。

みなさんはスマートフォン，タブレット，ゲーム機には慣れ親しんでいると思います。パソコンも「GIGA たん」などで触れていると思います。パソコンでは文書，プレゼン，図表を作ったり，絵や音楽を作ったりしているでしょう。このようにパソコンを**使う**ことには慣れていても，パソコンに**命令する**という感覚はないかもしれません。

プログラミングとは，計算機に計算をさせる指示書を書くことです。パソコンをはじめとするコンピュータ機器は，その形状はさまざまですが，いずれも電子計算機であり，なんらかの形で計算をさせていることになり

[*1] 私は 1976 年生まれですが，生まれた頃は周りにコンピュータなんかありませんでした。小学生の頃にマイコン (マイクロコンピュータ，小さなコンピュータという意味でもありますが，My Computer，私のコンピュータという意味でもあります。つまり個人単位でコンピュータが使えるようになった，というだけでも大きな出来事だったのです。) という言葉が出てきて，なんかかっこいいなと思った記憶があります。私が 10 歳になったころ，ビデオゲーム (テレビゲームでやるゲームが家庭でできるようになり，それをこのように呼びました。) が身の回りに出てきました。ファミリーコンピュータ，とくに「スーパーマリオブラザーズ」によって日本中の子供たちが熱狂したのが 11 歳の頃，「ドラゴンクエスト」によって社会問題になったのが 12 歳の頃になります。ともかくこの頃は，コンピュータといってもゲーム機のような扱いでした。

ます。ゲームで遊ぶことも，動画を見ることも，文書を作ることも，すべて電子的な計算の結果なのです。プログラミングをする上では，基本的な計算機の仕組みを知っておいても損はないでしょう。

電子計算機には次の 5 つの装置があります。

入力装置　キーボードやマウス，タッチパネルなどを使って情報を取り込むデバイス (装置)

出力装置　モニタやプリンタ，タッチパネルなどを経由して情報を出力するデバイス

演算装置　プログラムの命令に従って計算処理 (四則演算や論理演算) をする装置。一般に電子計算機の中央で一括して処理するので，Central Processing Unit(CPU) と呼ばれるものです。

制御装置　演算結果に従って他の装置に指示を出す装置のこと。演算装置とまとめて CPU に実装されています。

記憶装置　計算結果などの情報をいったん保持しておく装置のこと。コンピュータの内部にあって一時的な計算に使われる一次記憶装置，ハードディスクドライブ (Hard Disk Drive,HDD) やソリッドステートドライブ (Solid State Drive,SSD) などの二次記憶装置など。

最後の記憶装置については，一次記憶装置，二次記憶装置と種類が分かれていますがこの区別は簡単で，電源を落とした時に記憶が消えてしまうのが一次記憶装置，電源を落としても記憶が消えないのが二次記憶装置です。たとえば $12 + 38 =$ という計算をするとき，頭の中で「えーっと一の位が 2 と 8 だから 10 になって繰り上がるから･･･」と考えてから，ノートに $= 50$ という答えを書くと思います。次の問題に進むと，先ほどの「1 繰り上がるから･･･」という情報は忘れてますよね。でもノートに書いた $12 + 38 = 50$ というのは残っています。このノートがいわば二次記憶装置であり，頭の中で一時的に保持していた情報が一次記憶装置ということになります。一次記憶装置は RAM(Random Access Memory) とも呼ばれます[*2]。

ところで，コンピュータがやっているのは計算だけです。私はこの資料を PC に向かって書いており，キーボード (入力装置) を叩きながら，画面 (出力装置) を見て文字を連ねています。これも「キーが押されたら文字を表示させ，その文字列を記録する」という処理を機械が淡々とこなしているに過ぎません。あるいはマウスやトラックパッドで，アイコンを指し示し，カチリと押す[*3]ことで選択し，押し込んだまま移動させ (ドラッグ)，離すことで別の場所に置いたりします (ドロップ)。トラックパッドの場合は 2 本指で同時に押したりしますし，タッチパネルの場合は二本指を広げたり (ピンチアウト)，逆に二本指を狭めたり (ピンチイン)，3 本以上の指でファサーっと触って場所を広げたりします。たとえば「ファイルを掴んでゴミ箱に捨てる (削除する)」というのが我々にとっての操作ですが，コンピュータの内部では実はこんなことをしていません。ファイルは (二次) 記憶装置に書き込まれた情報です。記憶装置は原稿用紙のように小さなマス目がたくさんあって，ファイルとはそのマス目の XXX 番目から YYY 番目までの情報，ということです。このファイルを削除するというのは，記憶装置のある場所 (アドレス) に「削除されたものなので画面に表示しない」という情報を書き込む，という操作をしているだけです。じゃあなぜ私たちは「ゴミ箱にドラッグ＆ドロップ」なんてするのでしょう？ それはそのほうがわかりやすいからですよね。「ファイルを削除する」というのは，「メモリアドレスの XXX 番地に別の情報を書き込む」という操作だと言われてもピンとこないので，コンピュータが人間にとってわかりやすい表現をして見せてくれているのです。このユーザにとってわかりやすい幻を見せてその気にさせてくれるというデザインのことを，ユーザーイリュージョンと言います。ともかくこういう「画面で見ながら操作する」ことをグラフィカル・ユーザ・インターフェイス (Graphical User Interface,GUI) といいますが，これのおかげでコン

[*2] 実はこのように人間を 1 つのコンピュータに喩えて，そこではどのように計算がされているのか，人間の記憶装置や演算装置，入出力装置の特徴はどうなっているのか，というのを研究するのも心理学の仕事です。記憶や演算，制御については認知心理学や学習心理学，入出力については知覚心理学や生理心理学が専門的に扱っています。そういう意味でもコンピュータの登場は心理学に大きな影響を与えているのですが，それはまた別の講釈で。

[*3] 押すときの音から，この操作をクリック click と言います。2 回続けて押すことをダブルクリックと言います。

ピュータの操作は随分楽になっています[*4]。

プログラミングとはこうした装置を使う指示書を書くことでもあります。この授業で扱うのは主に数値計算ですから，逐一「この装置にこの動きを命令」といったところまで書き込む必要はありません。そのあたりのことは，言語環境がやってくれますのできにする必要はないのですが，根本的なところで何が動いているかはイメージしておいた方が良いでしょう。

1.2 プログラミングの基礎

あらためて，プログラミングとは，コンピュータに計算をさせるその仕様書を作ること，だと思ってください。お料理のレシピのように，計算レシピを書くわけですね。ただ相手は計算機ですので，言葉の端々まで正確に伝えないと理解してくれません。よく「思った通りに動いてくれない！」と不満を訴える初心者がいますが，それは当然で，プログラムは**思った通りに動くのではなく，書いた通りに動く**のです。思った通りに動かないのであれば，それは仕様書・レシピの方に間違いがあります。

今の第一原則に加えて，プログラミングを進める上での注意点をあげておきます。簡単なことだと思うかもしれませんが，基本的なルールをしっかり守ることが上達への近道です。

1. プログラムは思った通りには動かない。書いた通りに動く。大文字，小文字，スペルの違いに注意する。
2. 書き間違えないための工夫は美しさ。綺麗に書くことが大事。
3. 一言一句すべてに意味がある。ただの写経ではなく，意味を考えながら書く。
4. 「遊び心」をもって！ここを変えたらどうなるか，を少しずつ「やってみる」が大事
5. 変更は少しずつ。一気に変えるとどこが変わったかわからなくなるから。

1.2.1 綺麗に書こう

綺麗に書く，というのはコードの可読性を上げるために必要です。綺麗な書き方として，松浦 (2016) は次の 5 点をあげています。

- インデントは必ずする
- データを表す変数の先頭の文字は大文字。パラメータを表す変数の先頭の文字は小文字。
- 各ブロックの間は1行あける
- camelCase ではなく `snake_case` で。
- 「 」や「＝」の前後は 1 スペースあける

1 つめのインデントとは，字下げのことです。行の頭を少し凹ませることで，違う人まとまりであることを明示するのです。たとえば次の 2 つのコード[*5]は同じ働きをしますが，1.2 の方が見やすいと思いませんか？

code：1.1 インデントのないコード

```
1 data {
2 int<lower=0> N;
```

[*4] 実は人間の意識もこのユーザーイリュージョンのようなもので，実際の体の動かし方や感覚情報の受け止め方，処理の仕方は別ですよね。すべての情報を意識に上げるのではなく，「私は XXX をしている」と身体がそれっぽい幻想を投影して見せてくれているのが意識の正体ではないか，という議論があります。興味のある人は Norretranders (1999 柴田訳 2002) を読んでみてください。

[*5] このコードは確率的プログラミング言語 Stan の書き方で，この科目の後半で学ぶないようですから，今その中身までは理解しなくても結構です。

```
 3  array[N] y;}
 4  parameters {
 5  real mu;
 6  real<lower=0> sigma;}
 7  model {
 8  for(i in 1:N)
 9  y[i]~normal(mu, sigma);}
10  }
```

code : 1.2　インデントのあるコード

```
 1  data {
 2    int<lower=0> N;
 3    array[N] y;
 4  }
 5
 6  parameters {
 7    real mu;
 8    real<lower=0> sigma;
 9  }
10
11  model {
12      for( i in 1:N ){
13          y[i] ~ normal(mu, sigma);
14      }
15  }
```

インデントのある 1.2 は，data というブロック ({ と} で囲まれている領域) の中に，2 つの行が入っている，ということが明確にわかります。また model というブロックの中には，for から始まる分がありますが，これも複数行に渡るブロックを構成する文なので，その中身はインデント (字下げ) されています。このように，インデントすることでどこの列がどこまでブロックを組んでいるのかがわかりやすくなるのです。ちなみにこのインデントを作るのは TAB キーを使います。TAB キーは普段，どう使うのかわからないものだと思われがちですが，このインデントをしてくれるためのものです[*6]。コードエディタによっては，カッコで括ったときに自動的に閉じるカッコを用意し，また改行ごとに字下げ位置を合わせてくれるものがあります。これらの機能を使って是非わかりやすいコードを書いてください。

松浦 (2016) の指摘の 2,3 番目についてはお好みで，4 番目もそれほど強い・広く浸透した決まりではありません[*7]。ただし，5 番目のスペースの前後に少し空白を取るのは，見やすさのためにも是非実行してもらいたいところです。

これらの書き方は，**可読性**を上げるためのものです。プログラムは仕様書・レシピですから，あとで読み直した時やほかの人が読むときも，意味がわかるようにしておいた方が良いのです。自分だけわかれば良い，と

[*6] ここには流派があって，TAB でインデントする派とスペースキーを 4 回または 8 回押してインデントする派がいます。どちらでも働きは同じなのですが，個人的には数回の字下げを 1 回のキーで行ってくれる TAB のほうが良いと思っています。

[*7] ちなみにキャメルケース Camel Case，スネークケース Snake Case とは変数名のつけ方のルールです。R などプログラミング言語ではあらゆるものに名前をつけて管理しますが，名前のつけ方は任意です。命名はわかりやすい方がいいですが，長いものになると区切りを入れたくなるかもしれません。さて Camel とはラクダの意味で，ラクダのコブのように区切りのところだけ大文字にする，という記法です。Snake は蛇のことで，区切りのところにアンダースコア_を使うというものです。たとえば野球チーム，という変数名をつけたい時に，キャメルケースのやり方だと BaseballTeam のように書きますし，スネークケースのやり方ですと Baseball_team のような書き方になります。ちなみに R は日本語での命名も許しますが，そのほかの言語では一般的ではありませんし，何より全角と半角を切り替える時のミスやエラーが多くなるので，半角英数字での命名をすべきという点はどちらでも共通です。

思うかもしれませんが，その自分ですら何があるのかわからなくなってしまう可能性があります。そのような読みにくさはすぐにバグにつながりますから，綺麗に書くことを常々心がけておいて欲しいのです。

1.2.2 意味を考えて書こう

プログラミング言語は，ほとんど英単語のようなものです。プログラミング上の都合から，短い言葉に省略されていたり，アンダースコアやピリオドでつながって書いてあったりしますが，比較的わかりやすい言葉であることに違いはありません。

プログラムを習得し上達するためにすべきことは，最初は写経と呼ばれる，テキストや指示にしたがって書き写すことです。もちろんネット上にあるサンプルコードなどをコピー＆ペーストしても良いのですが，自分用の分析コードを書くためには手を入れる必要があったりします。ですから，最初はなるべく自分でキーボードを叩いて，コピー＆ペーストではなく入力する経験を積んでください。もちろん間違えてしまうことが多いでしょうが，転ばずに歩けるようになった人が一人もいないように[*8]，ミスを犯すことでどういうミスだったかを考え，自覚することではじめて，すこしずつですが b ミスが減っていきます。失敗する経験も時には必要なのです。

ということで，時折自分なりにコードを書いてもらうのですが，その時に「ただの記号列」と思わないでください。すでに書いたように，英語あるいはそれを省略した表現になっているので，見慣れないものかもしれませんが必ず意味があります。プログラムを始める最初の頃は，ミススペルが多く，あちこちでエラーが出て気持ちが萎えることもあるかもしれません。エラーが出る時は目を凝らして，どこに問題があるかを考えて修正することになります[*9]。ここで難しいのが，x や s などの大文字と小文字の区別がつきにくいもの，1 と l のように違いがわかりにくいものがあるということです。私は職業柄，多くの人に教え，多くのバグを見つけてきましたが，その中でも特別見つけるのに苦労したのが，`norma1` というミススペルでした。みなさん，これのどこがミススペルかわかりますよね？でもこれ，書く時に「正規分布のことだな」と思っていれば，そもそも入力時にそんな入れ方はしないと思うのです。ただの字だ，と心を無にしてしまうと，後々見つけにくいエラーを作ることにもなりますから，この文字・この名前・この関数はどういう意味だろう，と少し考えながら取り組んでみてください。

1.2.3 遊び心が大事

プログラミングを進める上では，遊び心が大事です。うまく動くコードがかけたら，もうおかしなことになりたくない，と手をつけないままにしてしまう人がいます。

ところで，今まで教えてきた経験からいって，プログラミングが上達する人は，うまくコードが書けたらすぐに「ここをこう変えたらどうなるんだろう」と試してみる人だと思います。最初はプログラムの意味がまったくわからないので，どこをどういじっていいのか見当もつかないかもしれません。しかしたとえば `blue` という文字が出てきたら，これは色の青のことじゃないか，と思いますよね。ではその `blue` を `red` に変えたらどうなるんだろう？やってみよう！となるような，そういう遊び心が大事なのです。

もちろん変えてしまったことでエラーになって，動かなくしてしまうことがあるかもしれません。その場合は，元に戻して元通り動くかどうか，さらに確認すれば良いのです。数字や文字を少しいじっただけで，PC が爆発してしまうようなことはありませんから，ちょっと遊び心を出して，どうなるのかな？と思う気持ちを大事にし

[*8] お釈迦様は除く。

[*9] ちなみにプログラミング歴 30 年以上の私でも，簡単な英単語，たとえば library のスペルを間違えるなんてことはしょっちゅうです。

てください。

1.2.4　変える時は少しずつ

プログラムを色々いじって遊ぶことが成長につながる，という話をしました。ただし遊び方には気をつけましょう。まずうまくいくコードができれば，それは保存しておいて，また同じ内容のものを別名で保存し，「遊んでいい方」「壊れてもいい方」を作って，そちらで色々変えて遊ぶといいでしょう。最悪なことがあっても，うまくいくバージョンは常に残っているわけですから。

そして色々試す時のコツは，一箇所ずつ変更していくことです。たとえば `blue` を `red` に変え，`line` を `box` に変え･･･と複数箇所を同時に変更して，うまくいかなかった場合，どちらが原因になっているのか把握できませんよね。これは心理学実験でいうところの交絡です。操作が交絡してしまって，原因が特定できなくなるのです。

また，実行するときも 1 行ずつやりましょう。とくに R は一問一答型，つまり 1 つの命令について 1 つの反応を随時返してきます。複数行をまとめて実行することもできますが，まとめて実行している途中でエラーが出ていて，その後の計算がすべて空回りしているということも少なくありません。エラーがどこで生じているのか，しっかり特定することが重要です。そのためにも焦らず，1 つずつ確実にできることを積み重ねていくという姿勢が重要です。

統計分析も複雑で細部まで設定し尽くしたモデルを作り上げていくことができますが，いきなり全体が完成するのではなく，小さなピースの積み重ねなのです[*10]。

1.3　いくつかのプログラミング言語

前置きが長くなりました。それではプログラミングを始めていきましょう。プログラミングには，プログラムを計算機への命令文として伝達する「言語」を利用することになります。この授業では R という言語を使います。R 言語は統計計算に特化した言語であり，心理学はもちろん統計に関する領域では多岐にわたって利用されています。比較的初心者に優しい言語設計になっており，また導入当初から豊富な関数，可視化 (グラフ化) 機能をもっています。統計解析を含んだ学術論文でも R を使ったものが多く，SPSS や SAS，Stata といった統計アプリケーションよりも広く使われています。その理由の一つは，R がフリーソフトウェアであるということです。マイクロソフトや IBM といった企業が提供するソフトウェア (プロプライエタリといいます) は，企業秘密の名目で，アプリの中で何がどのように計算されているかが隠されています。計算結果に間違いはない，と信じたいところですが，隠れている部分がある以上完全には信用できません。R はオープンソースのフリーソフトウェアです。すなわち，全ての計算過程を公開しており，お金をとって保証やサービスに代えるということをしていません。科学という営みにおいては，公明正大であることが特に重要ですから，この点は強調してもしすぎるということはないでしょう。

ところで，最近流行の AI，人工知能，機械学習といった領域では，R よりも Python の方がよく利用されています。Python も R と同様にオープンソースのフリーソフトウェアですし，優れた計算関数がパッケージとして提供されています。Python でも R と同様のことができますが，R よりもプログラミングにおいて厳格さを要求すること，複数のバージョンが乱立・併用して使われていることなどが理由でこの授業での採用をみおくりました。統計計算という意味では，Julia という言語もありますので，興味がある人はさまざまな言語を

[*10] R やプログラミングで統計を学ぶ意義はここにあります。GUI で操作できる統計パッケージも少なくありませんが，それらは画面が出てきた時にデフォルトで設定されている値があるのがほとんどで，自分が何をやっているのか細部まで気づかないまま実行できてしまうのです。そうすると当然，エラーが出たり，うまくいっても誤用している，ということにもなりかねません。自分で理解できていない技術に振り回されないようにするために，しっかりとわかるピースを積み重ねていく必要があるのです。

使ってみてください。

ところで，R，Python，Julia といった言語は**インタプリタ**型の言語です。Interpret(翻訳)，という言葉からわかるように，機械に命令をするときに逐一命令文を翻訳して伝えます。「足し算をしろ (命令)」「したよ (結果)」「掛け算をしろ (命令)」「したよ (結果)」というように，一問一答型でプログラムが進んでいきますから，自分が今何を命令して，どのような答えが出たかが明確です。

これとは別に，**コンパイラ**型言語というのもあります。C，Java などが代表例ですし，この授業の後半で利用する Stan という言語もコンパイラ型です。この方式は，一連の作業工程をかいたプログラムを，まるごと翻訳してから計算を始めます。この翻訳のことを Compile する，といいます。一連の作業工程のどこかにミスがあれば – それがスペルミスのような単純なものであっても – 翻訳の時にエラーが出ます。エラーには「XX 行目でエラーが出たよ」というメッセージが表示されますが，必ずしもその行に問題があるとは限りません。その命令文の前の方で問題が生じていて，発見したのがたまたまそこになっただけという可能性もあります。さらにいえば，コンパイル時はエラーが出ず，計算が実行されてから，計算結果が間違っているとか，変な結果が表示されるということもあります。その時は元の設計書のどこに問題があるか，目を皿のようにして探し回る必要があります。インタプリタに比べて，どうにも不便が多いようにも思えますが，コンパイラの利点はその計算速度です。インタプリタは逐一のやり取りになるので，どうしても計算が遅くなりがちです。一気に計算させたい時，まとまった計算単位がある時はコンパイラ型の方が有利なのです。

1.4 R 言語の基本的な挙動

前置きが長くなりました，と言ってからさらに前置きをしてしまいました。それでは早速 R を使っていきましょう。R はそのままでも使えますが，RStudio を使って利用した方が便利です。すでに一年時に，RStudio の基本的な使い方については触れていると思います。

RStudio をひらいて，この授業用に新しく**プロジェクト**を準備しましょう。プロジェクトを設定することで，ファイルの管理がやりやすくなります。ソースコードに簡単な四則演算を書いて実行し，また簡単な関数 (`sqrt` や `help` など) を実行してみてください。問題なく利用できるでしょうか。以下に第一回目の課題を記しますので，早速やっていきましょう。

もしうまくできない場合は，教員や TA，周りの人を頼ってください。あるいは，生成 AI に質問してもいいでしょう。生成 AI はうまく利用すると，R のコードも書いてくれたりします。簡単な問題であれば，結構正しく動くコードを返してくれます。ただし，生成 AI のいうことを丸呑みにせず，必ず自分で試しながら「意図した通りのことができているか」を確認するようにしましょう。

1.5 課題

1.5.1 R と RStudio の基礎

1. この授業のために，新しいプロジェクトを作成してください。プロジェクトは新しいフォルダでも，既存のフォルダでも構いません。
2. プロジェクトが開いた状態のとき，RStudio のウィンドウ・タブのどこかに「プロジェクト名」が表示されているはずです。確認してください。
3. 新しい R スクリプトファイルを開き，空白のままで結構ですからファイル名をつけて保存してください。
4. RStudio を終了あるいは最小化させ，OS のエクスプローラ/Finder から，プロジェクトフォルダに移動してください。先ほど作ったファイルが保存されていることを確認してください。

5. プロジェクトフォルダには，プロジェクト名 +.proj というファイルが存在するはずです。これを開いて，RStudio のプロジェクトを開いてください。
6. RStudio の File > Close Project からプロジェクトを閉じてください。画面の細部でどこが変わったか，確認してください。
7. RStudio を終了し，再び RStudio を起動してください。起動の方法はプロジェクトファイルからでも，アプリケーションの起動でも構いません。起動後に，プロジェクトを開いてください (あるいはプロジェクトが開かれていることを確認してください。)。
8. R を起動し，新しいスクリプトファイルを作成してください。そのファイル内で，2 つの整数を宣言し，足し算を行い，結果をコンソールに表示してください。
9. スクリプトに次の計算を書き，実行してください。
   (a) $\frac{5}{6} + \frac{1}{3}$
   (b) $9.6 \div 4$
   (c) $2.3 + \frac{1}{2}$
   (d) $3 \times (2.2 + \frac{4}{5})$
   (e) $(-2)^4$
   (f) $2\sqrt{2} \times \sqrt{3}$
   (g) $2 \log_e 25$

第 2 章

いんたーみっしょん；Stan の概略と環境の準備について

このセクションでは，確率的プログラミング言語 Stan を導入するにあたっての，周辺知識を解説します。授業中に解説するものではありませんし，ここに書かれていることのすべてを理解していないと Stan を使えないというわけではありませんが，導入や利用にはトラブルやエラーも多く，その際に前提知識，周辺知識があるとないとでは理解度が大きく異なります。「わからなくても，なんとなく動いた」という状態で受講し続けるのは自身のためにならないだけでなく，**面白くない**です。知識があってはじめて価値がわかるということもありますので，共用としてご一読いただければと思います。

2.1 はじめに

この授業では統計言語としての R，それを使う統合開発環境としての RStudio，そして**確率的プログラミング言語 (stocastic programming language)** としての Stan を利用します。大学での PC ルームの利用にはこれらの環境がすでにある程度準備されていますが，最新バージョンではありませんので，自分の PC に環境を準備することを強く推奨します。今後の研究室配属やその後の卒業研究などでも活用することになると思いますので，まずは自身の PC 環境に準備することを考えてみてください。

この付録資料は，これらの環境を準備する方法について解説するものですが，提供されるソフトウェアのバージョンや対応する環境などは日々発展するものですので，必ずしも最適な情報提供になり得ません。基本的な情報は提供いたしますが，執筆時[*1]での情報であることも多く，より新しい情報についてはインターネットなどでキーワード検索を行なって，より良いものを探してみてください。

PC の操作が苦手に思っている人のために付言しますが，うまくいかなかったからといって癇癪を起こしたり，諦めたりしてはいけません。相手は機械ですから，命じられたことを愚直に実行しているだけです。また文房具のようなものですから，恐れて嫌うのではなく，愛して可愛がってあげることが肝要です。もしみなさんの中に，お持ちの PC に名前をつけていない人がいるようでしたら，今すぐ命名することをオススメします。パソコンが，と思うと腹が立ちますが，わたしの XXX ちゃんが，と思うとエラーも「ちょっとご機嫌斜めなのかしら」と受け入れやすくなります[*2]。

また，困った時は教員，TA，先輩などに相談することが重要ですが，その際には症状や経緯を正確に伝えることが必要です。ペットを病院に連れていくときに「何もしてないのに勝手に病気になった。治してほしい」

[*1] この記事は 2022 年 11 月 10 日に加筆修正されました。

[*2] ちなみに私が初めて手に入れたノート PC には「秀吉」という名前をつけました。Mac に変わってからは代々数学者の名前をつけることにしています。私はいま，Gauss ちゃんや Hermite ちゃんを持ち歩いています。

といわれても獣医さんは困ると思います。**普段の様子や症状についての丁寧な報告**を心がけてください。とくに PC は機械ですので，何もしていないのに壊れるということはありません。自分が自覚していないことであっても，「なにかをした」から様子がおかしくなるのです。自分はそんな大それたことはしていない[*3]，と思っても必ず何かトリガーがあったはずなのです。どこをクリックしたか，どういうコマンドを入れたかという**履歴をしっかりと把握し，丁寧に報告する**ことを心がけてください。

2.2 Stan の位置付け

確率的プログラミング言語 (stocastic programming language) とは，確率モデルを記述し，データと合わせることで統計的推論をしてくれるコンピュータ言語のことです。Stan ができる前は，**BUGS** や **JAGS(と)** いうものがありましたが，BUGs は開発が終わってしまいました[*4]。今，こうした言語は Stan が最先端だといって間違い無いでしょう。これらの言語は，具体的には確率モデルとデータの関係を記述することで，事後分布からの乱数を生成するというものです。以下ではこの言語の利用形式について解説します。

2.2.1 コンパイラとインタプリタ

プログラミングはコンピュータを動かすための仕様書を書くことです。ここで「コンピュータ」というのは計算機という意味だと思ってください。もちろんコンピュータは数字の計算だけでなく，音楽を奏でたりゲームを楽しませてくれたり，と色々なことができるのですが，その背後にあるのはとにかく 0/1 の数値演算です。0 か 1 かという数字がどうして映像や音声，通信になるのか，と疑ってしまうほど異なっているようですが，それでもすべて数字の計算から構成されています。

コンピュータができ始めたごく初期の頃は，これを動かすのに 0 と 1 からなる数字の羅列で仕様書を用意していました。流石にそれではわかりにくく表現力にも乏しいので，次にできたのがマシン語とよばれる 16 進数による表現でした。この段階でも，知らない人にとっては暗号か，意味のある連なりに思えない文字列にすぎません。画面に線を引いて欲しい時に `line` という命令で伝えられるようになって，やっと普通の人間にも意味がわかるレベルになってきます。このように，人間がわかるレベルでコンピュータに命令が伝えられるような言語のことを**高級言語**と言います。高級言語は人間寄りで，直接コンピュータが理解できませんので，この高級言語を機械の言葉に翻訳するためのアプリケーションを介在させます。それがいわゆる**プログラミング言語 (programming language)** と呼ばれるものです。

プログラミング言語は，古くは BASIC，PASCAL，FORTAN などと呼ばれる書式のものがあり，その後出てきた C 言語，その改良版である C++ 言語[*5]などが有名です。他にも JAVA や Object C，Python などが有名ですね。R も言語の一種で，統計解析に特化したものです。また，この授業で扱う Stan という**確率的プログラミング言語 (stocastic programming language)** は，確率モデルの分析に特化したものです。みなさんがデータサイエンス業界に進むのであれば，Python や R を使えるようになっておくといいでしょう。これらの言語の特徴の 1 つは，パッケージを追加することで機能が拡張できることにあります。とくに**機械学習**系 (AI など) のパッケージが豊富なのが Python です。またこれらの言語は基本的にフリーソフトウェアであり，導入にお金がかからないところもポイントですね。タダで始められるおもちゃみたいなもの

[*3] たとえば計算途中だけど時間が来たので電源をオフにした，といったことでも，内部ファイルにアクセスしているときの中断であれば，十分に PC を破壊することができます。

[*4] JAGS はまだ開発が続いているようで，R から使うこともできます。

[*5] この ++ という書き方は，C 言語特有の「変数に 1 加える」という表記法であり，C 言語の改良版という意味で C++ と命名されています。読み方はシープラスプラスですが，シィプラプラの愛称でも知られています。

です！[*6][*7][*8][*9]

プログラミング言語の分類には，その設計思想や書き方などを基準にすることもできますが，実行方法を基準にするものとして**インタプリタ**型と**コンパイラ**型とに分けることができます。インタプリタ型は，interpret，つまり翻訳型です。これは毎回の命令を機械語に翻訳して実行していきます。R はインタプリタ型言語で，毎回の命令を逐一翻訳して実行していきます。R を実行するとき，コンソールに>というマークが出ていますよね。これは R が聞き耳を立てている状態，入力待ちの状態なのです。ここに命令を書くと (たとえば 2+3 と書く)，R はすぐさま答えを返してきます (たとえば [1] 5 と返す)。基本はこのように一問一答，ひとつひとつの命令を毎回機械語に翻訳して実行し，その答えを出力するという手続きをとっています。もちろん私たちの分析プログラムは，一行で書ける簡単なものではありませんから，複数行に渡る長々としたファイルを書くことが多いでしょう。こうしたファイルはプログラムともいわれますし，一行一行の命令のことを**スクリプト (script)** ということもあります。プログラムファイルあるいはスクリプトファイルを開いて，一行ずつ実行するのが R の基本的なスタイルであり，RStudio はスクリプトファイルを編集する**エディタ (editor)** がセットになっているのが便利な点なのでした。

これに対して，インタプリタ型は，スクリプトファイルをまとめて機械語に翻訳 (コンパイル) し，実行ファイルというのを別途作ります。その上で，実行ファイルを実行すると計算が進められるのです。どうしてこんな手間がかかることをするのでしょうか。ひとつには，ひとつひとつの命令分を逐一翻訳しているのでは，実行スピードが遅くなるということが挙げられます。命令を逐一聞き耳を立てて待ち，全ての命令を聞き終わるまで途中の指示を記憶しておいて，命令文が終わってはじめて行動に移す，というのでは記憶容量も時間もかかるわけですね。これに対して，一連の計算命令文書を一括で渡すことができれば，機械は内部的に効率よく計算手順を配置して実行できます。コンパイラ型は計算スピードが速いのです。ちなみに実行ファイルは機械がわかる言葉に書き換わっているので，人間が見ても意味がわかりません。また，OS や CPU に理解できる形に書き換えていますので，MacOS の実行ファイルを Windows で実行する，ということはできません。翻訳後の言語が違うからです。スクリプトファイルは環境を超えて共有できますが，それを翻訳したり最適化したりする仕組みは，個別の環境に依存します[*10]。

Stan はコンパイラ型です。Stan のファイルを読み込んで，機械にわかるように「事後乱数生成命令文」に翻訳して実行します。確率モデルから乱数を生成するのは非常に高度な計算機能ですから，コンパイルしてまとめておくことでスピードアップの効率が断然良くなるわけです。コンパイラ型の利点はこのスピードにありますが，欠点は「命令に変更やミスがあれば翻訳し直さなければならない」というところです。あり得ない命令を出しているようであれば，コンパイルの段階で「おかしな文法だよ」とエラーを出して翻訳を止めてくれます。翻訳が通れば良いかというとそうではなく，翻訳はあくまでも文法上の手続きであって，数値や内容に間違いがあっても翻訳自体は完了させることができるのです。翻訳 (コンパイル) には少し時間がかかりますか

[*6] 余談ですが，Scratch や任天堂 Switch の「ナビつき! つくってわかる はじめてゲームプログラミング」などにみられる，GUI ベースのプログラミング言語もあります。これはプログラミングの要素がアイコンやキャラクターになっていて，それらを組み合わせることで計算させるという方法をとっています。その操作のしやすさ (ミススペルがない！) から，小学生などにも導入できる素晴らしい取り組みです。同様の発想は，LOGO 言語などにもみられました。

[*7] Perl や Java Script など，Web ブラウザ上で動くプログラミング言語もありますが，これはブラウザ上での操作に限定されており，数値計算にはあまり向かないのでここには取り上げていません。さきほどあげた JAVA と JAVA Script は別物なので注意してください。

[*8] GUI アプリケーションを作ることに特化した言語もあります。数値計算と違って，マウスがどこでクリックされたか，と言った「動き」を契機としてプログラムが走る，イベントドリブン型の言語で，Microsoft 社の Vsial Basic などが有名です。これは Microsoft Office 製品の中に埋め込むこともでき (VBA)，これを使うと Excel でも高度な統計解析ができたりします。Excel を使った統計ソフトウェアとして代表的なものに HAD があります清水 (2016)。他にも Delphi などの言語がありました。

[*9] シ (2016) には，数え方にも癖がありますが，117 ものプログラミング言語が紹介されています。

[*10] ちなみに R や Rstudio はもちろん，Word や Excel などのオフィスアプリ，果ては OS そのものも，コンパイルされた実行ファイルを PC が計算して実行しているに過ぎません。

ら, 文法上も内容上も間違いのない命令分をしっかり準備しておくことが重要になってきます。

2.2.2 ファイルのやり取りについて

ここで, R/RStudio をつかって Stan を使う際の, ファイルのやり取りを見ておきましょう (図 2.2)。

Stan のコードは R のコードの中に書くこともできますが, 別のファイルに保存しておくのがベターです。Stan ファイルは拡張子を `.stan` とすることが一般的です。このファイルの中身はプレーンテキストでコードを書いていますから, 一般的なエディタ[*11]で編集できます。RStudio のエディタ画面も必要十分なエディタ機能を持っていますので, RStudio をエディタとして利用するのがいいでしょう。

RStudio で File > New File として新しいファイルを開くときに, Stan ファイルとして開くことが可能です。Stan ファイルとして開くと, 初心者向けの配慮からか, 最初からちょっとしたコードがすでに書かれています。コードの書き方を示すサンプルコードなのですが, 実際にこのコードを使って何かすることはありませんので, 中身は全部削除してしまいましょう。それでも RStudio はその画面が Stan ファイルのものだということは認識していますから, そこで書いたファイルは Stan ファイルとして扱ってくれます。ここでまちがって新しく R Script で開いてしまった, というときに, 保存するときだけ `.stan` をつけておけばいいか, という対応をすると失敗します。RStudio では拡張子をあえて表示していませんので, ファイル名として `hoge.stan` と書くと実際には `hoge.stan.R` というファイル名になっています。拡張子は最後のピリオドで判断されますので, これは R ファイルなのです。このミスを防ぐために, ファイル名にピリオドは使わないこと, ファイルの種類を変える時はエディタ画面の右下でファイル種別を選択してください (図 2.1)。

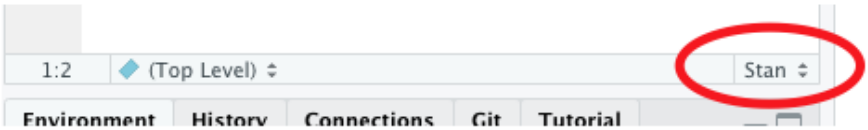

図 2.1 ファイル種別の変更

RStudio を使って Stan を書くと便利なことがいくつかあります。ひとつは強調表示機能で, ブロックや関数名など Stan で使うことが決まっている専門用語は色やフォントが変わって表示されます。`transformed parameters` のように長い専門用語を書くときはミススペルが心配ですが, 書いた後で強調されなければスペルミスがある, ということがわかります。もうひとつの利点は文法チェックの機能です。コンパイル型なので, Stan に書かれた命令文は実行前に一括で翻訳されますが, スペルミスや型の違いなど, 文法的に間違っているところがあればこれも自動的にチェックして警告記号がでます。また Stan ファイルが開かれているペインの左上に `Check` というボタンがあり, これを押すことでファイル全体の文法チェックをしてくれます。もし文法上の問題がなければ,「hogehoge.stan is syntactically correct.(hogehoge.stan というファイルは文法的には正しいですよ)」というメッセージがコンソールに表示されます。逆にいうと, これが表示されないというときは何か間違っているので, コンパイルに進む前に修正しましょう。

さてこのようにして Stan ファイルを準備します。また分析用のデータセットが外部ファイルにある場合などは, これまで同様, 当該プロジェクトフォルダ内に置いておくと良いでしょう。これらはいずれも, R のコードで

[*11] エディタとは ASCII ファイル, いわゆるプレーンテキストを編集するアプリケーションの総称で, OS にデフォルトで「メモ帳」などの名称で含まれています。デフォルトのアプリは文字を読み書きできるだけですので, もう少し発展的, あるいは便利な拡張機能を持っている専用のアプリを使うことが一般的です。筆者は最近もっぱら VS Code というエディタを利用しています。他にも MacOS でしたら,「mi」というアプリを好んで使っていました。Windows でしたら「秀丸」というアプリが有名です。Ubuntu には gedit というアプリがデフォルトで入っていますし, 歴史やユーザの多さでは vim や emacs などがあります。Linux 界隈で, vim 派か emacs 派かの話題は, きのこたけのこ戦争ばりに激しい戦いになります。

呼び出して使うことになります (図 2.2 の 1. と 2. のステップ)。R では，Stan ファイルやデータファイルを読み込んで，これを PC 内部にある Stan に渡します (図 2.2 の 3. のステップ)。Stan は R と違う言語であり，Stan 独自の計算機能で計算してくれるわけですから，R と Stan の橋渡しが必要になります。R ではこれを `cmdstanr` や `rstan` というパッケージを経由して行います。これらのパッケージが Stan を呼び出してくれるわけです[*12]。

データと確率モデルの関係を記した Stan ファイルと，データのそのものを受け取った Stan は，コンパイルして PC 内に乱数発生装置を作ります (図 2.2 の 4. のステップ)。事後分布の関数を書き出すのではなく，その形状とそれを代表する乱数を作る状態をもつわけです。R の指示をうけて，そこから必要な数だけ乱数を作り出します (図 2.2 の 5. のステップ) 。R はこれを受け取って，統計解析を始めるわけです。R は統計に特化した言語ですから，集計や可視化などの作業はお手のもの。事後分布から得られた大量の乱数は，事後分布の特徴を反映した大量のデータセット，事後分布の具体例たちになっているのでした。

分析の進め方は以上のようになります。R でデータを整形し，Stan のファイルも別途用意して，R から Stan を呼び出す，Stan か返してきたデータを R で解析する，ということです。R の拡張子は `.R` で，Stan の拡張子は `.stan`，データファイルの拡張子は `.csv` などでしょうか。いずれにせよ，複数の種類のファイルをやり取りしながら進めるので，相互の関係についてしっかり理解しておく必要があります。

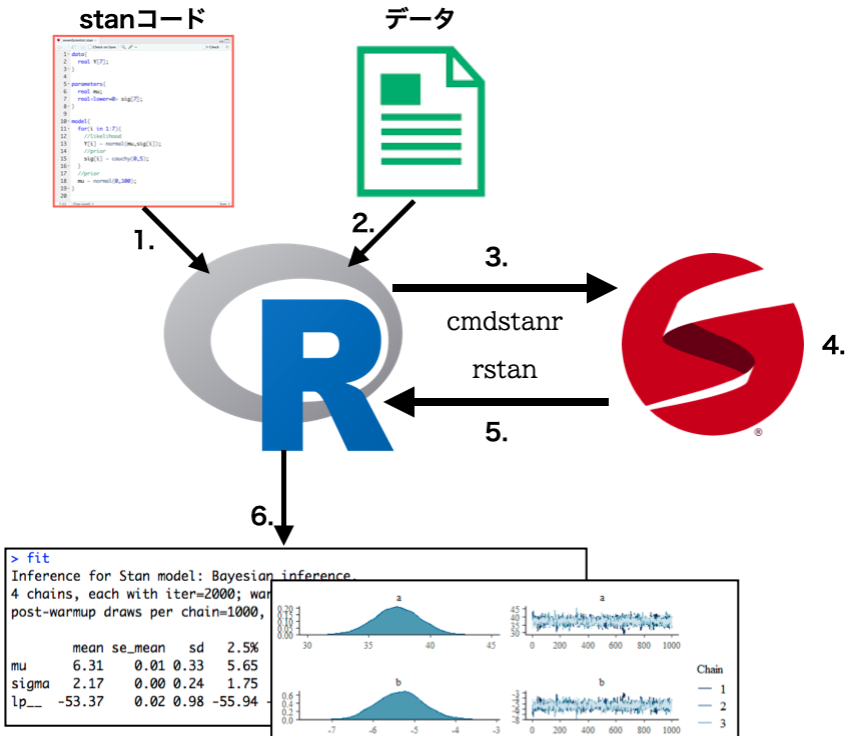

図 2.2 R/RStudio と Stan ファイルのやり取り

[*12] このように，Stan は独立した計算環境であり，R 以外のアプリケーションから呼び出して使うことが可能です。Python から使いたい場合は，PyStan というパッケージ経由で，Julia から使いたい場合は StanJl，コマンドラインから使いたい場合は cmdstan といったように，いろいろなルートがあります。

2.2.3 2 つのルート

ここでは R から Stan を呼び出す 2 つのルート，具体的には `rstan` パッケージと `cmdstanr` パッケージについて説明します。この 2 つのパッケージは，いずれも Stan を呼び出してつかう，R と Stan の間を取り持つインターフェイスの役割を持ったパッケージです。間を取り持つだけなので，Stan ファイル自体は同じであっても構いません。同じ Stan ファイルを `rstan` から呼んでも，`cmdstanr` から呼んでも，結果は同じです。ただしインターフェイスが違うので，結果の扱い方が変わってきます。他にも違うところがいくつかありますので順に説明していきますが，結論からいうとこれから[*13]は `cmdstanr` のほうを使うほうがお勧めです。

歴史的には `rstan` のほうが古くから存在します。このパッケージも最初の頃は導入に一苦労するものでしたが，2016 年ごろから CRAN を通じて配布されるようになりました。要するに，他のパッケージと同じように，`install.packages("rstan")` と R で書くだけで導入できるようになったのです。このようにして導入するとわかりますが，`rstan` は多くの依存パッケージがあります。Stan はコンパイルするのに OS に応じた C++ コンパイラを借用しますが，R から C++ を呼び出すパッケージや R と C++ の間に入る StanHeader と呼ばれる緩衝材など，関連する多くの環境を整えてやっと R から Stan へのルートが通じるのです。ユーザにとってはそのような苦労は知ったことではない，という感じですが，「間に多くの調整が入る」ということがエラーの温床になっていました。つまり，いくつかの内部的ステップのどこかでバグがある，互換性がない，というようなことがあると，Stan ファイルが同じでも，バージョンが変わるとコンパイルできないということがよくあったのです。Stan 本体の方も開発が盛んですから，Stan の新しいバージョン，新しい機能を使いたいと思っても，パッケージが対応するのを待たなければなりません。パッケージが対応しても，途中の媒介パッケージが追いついていなければバグになったりします。これらを使う R や OS もアップデートしていきますから，その度に「動かなくなる」ということはよくありました。研究の再現性が問題になる昨今ですが，自分の研究とデータであっても，自分の環境で再現できないということが少なくなかったわけです[*14]。そうなると，今動いているからそれでいい，と環境のアップデートを控えるようになりますが，ご存知の通り OS やアプリのアップデートはバグやセキュリティホールの修正など，安全につかうための基本的な機能に関わってきますから放置もできない，という問題があるわけです[*15]。

このような不安定な環境になる原因は，R と Stan の間にあるさまざまな障壁と，それをなんとかして調整しようという媒介パッケージの存在でした。そこでなるべくシンプルに R と Stan をやりとりする方法はないものか，ということで考えられたのが `cmdstanr` です。これはコマンドライン[*16]から Stan を呼び出す `cmdstan` を R から呼び出して使おう，というものです。Stan を R になんとか取り込んで動かす，というのではなく，Stan の部分は OS のプリミティブな環境にお任せ，R は結果をもらうだけ，というシンプルな設計にしようというわけです。たとえば `rstan` は Stan の計算結果を R オブジェクトとして取り込みますが，`cmdstanr` は実は MCMC の結果は csv ファイルのような形で吐き出しており，それを読み込んで使います。御用聞きや仲介業者を介して情報をもらうのではなく，現地で直接買い付けしてくるようなイメージでしょうか。このよう

[*13] 執筆時点，2022-11-14 現在

[*14] 実際 R や OS のアップデートに関わるエラーは大変で，筆者は一度 OS をアップデートした後半年ほど Stan が使える環境が作れず，研究が頓挫したことがあります。そのために別途，PC を購入し直したぐらいです。

[*15] OS のアップデートなど，通知が来ても「今支えているからいいや」と無視する人もいますが，できれば OS やアプリは最新版であるほうが良いのです。セキュリティホールの問題などを放置しておくと，インターネットを介して自分の PC の中身が全部見られて公開される危険性があります。Stan しか使わないネットに繋がない PC というのがあればいいのかもしれませんが，文房具としての PC はもっと一般的な用途がありますから，そうもいってられませんよね。また，一昔前の OS アップデートは，アップデートの失敗で PC が動かなくなるというようなこともありましたから，アップデートが公開されてすぐに対応するのは「人柱」などと揶揄されていましたが，最近はそういったこともほとんどありませんので，安心して常に最新の状態にしておいてください。

[*16] MacOS や Linux では端末，ターミナル，Windows ではコマンドプロンプトなどと呼ばれる，PC に直接コマンドで命令を出すプリミティブな実行環境です。

に間に挟むものを減らすことによって，不安定な要素をなくしたわけです。

cmdstanr は Stan と R の複雑な絡まりを排していますので，Stan の開発が進めばすぐにそれを R に届けることができます。たとえば 2022/11/14 現在で，公式にリリースされている rstan のバージョンは 2.21.1 ですが，cmdstanr が呼び出す stan のバージョンは 2.30.1 です。cmdstanr のほうが新しいものに対応できていますね。このように，cmdstanr がでてきてからはこちらの対応，反応が早く，また動作も安定的ですから，よりお勧めしやすくなっています。

これまでに出ている R で Stan を使う方法を解説したテキストの多くは，rstan をつかっていますので，今それを使うためには多少の読み替えが必要です。とはいえ，Stan のコード自体はそのまま使えるものがほとんどで，R とのインターフェイス，やりとりの方法が違っているだけです。cmdstanr には，結果を rstan で得られたオブジェクトに変換する関数も含まれていますから，これらを使って変換すれば，以後のコードは rstan 準拠のものでもエラーになることはありません。このテキストでは第 2 版から cmdstanr を中心にするように切り替えましたが，以前のコードでも本質的に問題はありません。

2.3 導入の概略

環境の導入は，OS の種類によって変わります。Widows なのか Mac なのか，あるいは Linux なのかといった違いに合わせて導入を考えてください。OS の種類で言えば，Linux は全体的に CUI 操作が主であり，堅牢かつ合理的な使い方ができるものです。如何せんマイノリティですので，ググってもあまり情報が出てこないところが玉に瑕です。Mac は Linux ベースの OS になっているので，比較的堅牢かつ合理的な使い方ができます。ハードウェアも OS も 1 つの会社 (Apple) が作っているので，サポートもしやすいという利点がありますが，日本では少数派なのが残念なところです。Windows は多数派の OS ですが，OS メーカはソフトウェア会社でハードウェアが別会社なことが多く，利用される範囲は広いのですが，その分問題が生じたときにケースバイケースになりがちです。多くのハードウェアの違いを吸収するために最大公約数的な設計をするせいか，ソフトウェアデザインの統一性がないところが欠点です。ユーザ数は最も多いので，一生懸命調べたら同じような問題で困っている同じような機体の人に出会える可能性が高いのがせめてもの救いです。また光明として WindowsOS も Linux ベースの仕組みを取り入れ始め (WLS)，比較的合理的な振る舞いができるようになったことが挙げられますが，OS のグレードによって導入のしやすさが異なります。

ちなみに ChromeOS や iOS, Android などタブレット・スマートフォンでの統計環境の利用は限界があります。これらは計算結果を利用するフロントエンドとしては非常に高機能なのですが，インタラクティブに使うことには不向きです。いわば書籍のように多くの情報を一方的に与えてはくれるのですが，ユーザが計算するといった働きかけの補助になるような (ノートとペンのような) 使い方ができませんので注意してください。

さて，環境の導入として 2 つのルートを紹介しましょう。第一のものが最も基本的なアプローチで，自分の手元の環境を作り上げるというものです。これを**ローカル環境**での構築と呼びます。ローカル環境は自分だけの環境ということなので，自分の責任でもってしっかりと環境構築をできます。第二のアプローチとして，ある程度できあがった環境を丸ごと取り込む，**仮想環境**での構築という方法があります。PC の OS の中に別のマシン・別の OS をさらに憑依させて使うようなやりかたで，これができると取り込む環境は完成しているので細かい設定をする必要がありません。問題は「憑依させる」準備がいろいろ大変であるということです。また憑依させた PC はブラウザ経由でアクセスすることになることにも注意してください。

2.3.1 導入方法 1；ローカル環境での構築

ローカル環境での構築は，R，RStudio の導入から入ります。ここは既にできている人もいるかもしれませんが，念のためバージョンが最新のものかどうかを確かめておいてください。ローカル環境構築の方法がわからない人は，「RStudio」「インストール」などのキーワードでウェブ検索し，自分の環境にあった例を見つけてそれをみながら進めると良いでしょう。参考までにいくつかあげておきますと，新しいものでは，高知工科大学柳井先生のこちら http://yukiyanai.github.io/jp/resources/ とか，Quiita の記事 https://qiita.com/hujuu/items/ddd66ae8e6f3f989f2c0 が良いでしょう。書籍では，手前味噌で恐縮ですが小杉 (2019) などが良いでしょう[*17]。

次に確率的プログラミング言語，Stan の導入を行います。Stan を R から使うためのパッケージとして `cmdstanr` と `rstan` という 2 種類があるという話はしました。かつては `rstan` のほうがよく使われていたのですが，最近は `cmdstanr` のほうが安定的に動作し，開発も盛んになっているので，2022 年度後期からは `cmdstanr` をこの授業のデフォルトパッケージとしたいと思います。このパッケージの導入には，公式サイト https://mc-stan.org/cmdstanr/articles/cmdstanr.html を参考にしましょう。「cmdstanr インストール」で検索すると色々な紹介サイトが出てきますので，自分と同じ環境でなるべく日付の新しいものから参考にしていくといいでしょう。

インストールにあたっては，Stan がコンパイルされるときに利用する C++ 言語環境が必要です。これは MacOS であれば Xcode の導入が，Windows であれば Rtools での導入が必要になります。

■MacOS ユーザの場合　AppStore で Xcode と検索し，Xcode をインストールすれば OK です。

■Windows ユーザの場合　Rtools という R 周辺のツールをインストールする必要があります。https://cran.r-project.org/bin/windows/Rtools/ にいって，自分の R のバージョンにあった RTools をインストールしてください。この他にも Windows ユーザは導入にあたって色々障壁にあたることがありますので，こちらのサイト https://norimune.net/3609 なども参考にしてみてください。

2.3.2 導入方法 2；仮想環境での構築

第二のルート，仮想環境を構築する場合ですが，これについては我らが国里先生が大変詳しいサイトを作ってくださっています。日本心理学会のチュートリアルワークショップ，「再現可能な日本語論文執筆入門: jpaRmd で実現する再現可能で低コストな日本語論文執筆のはじめの一歩」で使われた時の資料集がそれで，このサイト https://ykunisato.github.io/jpa2021-tws-jpaRmd/ の事前準備編をよく読むと導入できます。通信環境にもよりますが，慣れているとものの 10 数秒でシステム導入できるという優れものです。面倒な設定が怖い人は，是非試してみてください。

2.4 導入方法 3；外部サーバの利用

最近，Google 社がオンラインで利用できる分析環境，Google Colaboratory を提供してくれています。これは Google 社が提供する，ブラウザで実行できるプログラミング環境です。Google が教育，研究用に提供しているもので，90 分間は無料で利用できます。90 分を超える場合でも，書いたプログラムを保存して

[*17] インストールに際しては，RStudio Desktop の Free edition を選んでください。RStudio Cloud や RStudio Server は別のものです。

おけば，新しいセッションを開始することで続けて利用できます。これを用いる利点は，どのユーザにも同じ環境を提供できることです。

基本言語環境は Python で提供されており，Jupyter Notebook を使いますが，次のアドレス (に含まれるコード) を使えば言語環境を R に変えて利用することができます。

https://colab.research.google.com/notebook#create=true&language=r

ここでウィンドウに R のコードを入力し，実行していきます。`cmdstanr` や `rstan` もインストールできます。インストールに際して，`install.packages("rstan")` のように入力することもできますが，この方法ですとインストールに随分と時間がかかります。そこで少し例外的ですが，システムコマンドを直接利用して，`system("apt install -y r-cran-rstan")` のようにすることで，大幅にスピードを短縮することができます。

2.4.1 導入後の確認

インストールが終わったらサンプルコードを実行して「動くかどうか」を確かめてみてください。サンプルコードは Getting started with CmdStanR のサイト，あるいは RStan Getting Started のサイトに掲載されています。どちらのパッケージから実行しても，文字列がズラズラっと出力されればとりあえず動いたと思っていただいて結構です。赤い文字やエラーなどが表示されうる，あるいは全く表示されない場合は，インストールがうまくいっていないことを疑いましょう。上に戻って，丁寧に説明に目を通して一歩ずつ進めてください。機械に命令することですので，自分勝手にショートカットしたり変更したりしないことが重要です。

2.5 Stan を使ってみよう

具体的な Stan コードの書き方や統計モデルへの応用については，本書の次の章から順に説明していきます。それに先立って，全体的な注意をしておきたいと思います。具体的には，バージョンによる違い，パッケージによる違い，そして本書の以下の章で使う関数の紹介です。

2.5.1 バージョンによる書き方の違い

`cmdstanr` と `rstan` はパッケージの使い方以上に，それらが呼び出す Stan のバージョンの違いがあります。上で解説したように，`cmdstanr` のほうが最近は開発が進んでいて，より新しい Stan の機能を使っていることになります。バージョンが違っても，基本的な書き方などに違いはないのですが，バージョン 2.26 から Stan の文法に変更が予定され，導入されました。

データ X が N 人分あったとします。数学的フォームでは，$X_1, X_2, X_3, \cdots, X_N$ というようなものですね。これはコンピュータ上では `X` と名付けられた配列，サイズ N というように考えます。これは，Stan では次のように書くものでした。

code : 2.1 Stan2.26 以前の書き方

```
1  int n[5];
2  real a[3, 4];
3  real<lower=0> z[5, 4, 2];
4  vector[7] mu[3];
```

これらは上から順に，サイズ 5 の整数変数 `n`, サイズ 3×4 の実数変数 `a`, 下限 0 のサイズ $5 \rightarrow, es4 \times 2$

の配列 z, 長さ 7 のベクトル mu が 3 つ, ということを意味しています[*18]。

この書き方が，Stan のバージョン 3.32 以降は次のような書き方になります。

code : 2.2　Stan2.32 以降の書き方

```
1  array[5] int n;
2  array[3, 4] real a;
3  array[5, 4, 2] real<lower=0> z;
4  array[3] vector[7] mu;
```

このコード 2.2 はさきほどのコード 2.1 と同じ意味で, 書き方が変わっただけです。一瞥してわかるように, サイズを先に array という言葉で宣言しましょう, というだけですね。

さて, 今現在は Stan2.26 と 3.32 の間, 過渡期になります。ですから, どちらのコードで書いてもエラーにはなりません。ただし，(ここからが少し面倒なのですが)

- RStudio のコードチェック機能は新しいコードの書き方に対応しておらず[*19], 新しい書き方をするとエディタ上でエラーの警告 (赤いバッテンがつきます) が出ますし, チェックボタンを押しても SYNTAX ERROR が出ます。
- rstan パッケージは使っている Stan が古いこともあって, 新しい書き方のコードをコンパイルしようとするとエラーになります。RStudio と同じ，SYNTAX ERROR が出ます。
- cmdstanr パッケージは逆に, 新しい Stan の書き方を推奨していますので,「その書き方は古いよ」という警告を出してきます。Declaration of arrays by placing brackets after a variable name is deprecated and will be removed in Stan 2.32.0. Instead use the array keyword before the type. This can be changed automatically using the auto-format flag to stanc. という警告がでますが, これは「配列のカッコを変数の後ろに置く書き方はもうダメで，Stan2.32 以降は無くなります。型の前に array と書くようにしてください。これは auto-format フラグを立てることで自動的に変更するようになります。」という意味です。

つまり, 本書は cmdstanr を推奨していますが, そうすると RStudio のエディタ上ではエラーが出て文法チェックができず, コンパイルした後でいざサンプリング, というときにエラーが出たりします。チェックをするには, コンパイルしたオブジェクトを使ってサンプリングをする前に，model$check_syntax() 関数を実行する必要があります。rstan パッケージを使うとスマートに文法チェックもできていいのですが，rstan そのものが不安定ですし, 今後徐々に開発・発展をしなくなっていくことが明らかですので, どこかで新しい方に舵を切る必要があります。

本書のコードは新しい書き方に統一していますが, 最初のうちは, そして今しばらくは, 実際にコードを書くときは古い方で書いたほうが便利かもしれません。

2.5.2　パッケージによる指示と出力の違い

さて今度はパッケージごとの違いを見ていきましょう。

[*18] ベクトルは数字をセット, 人まとめとして演算するものです。最後の変数は，7 つの数字のセットが 3 つという意味で，7 つセットで 1 つのまとまりですよ, ということを明示的に宣言していることになります。

[*19] RStudio のバージョン 2022.07.2 Build 576 で確認しました。

指示の仕方の違い

まずは実行方法です。Stan では事後分布からの乱数を生成しますが，それにあたって「乱数の数」「ウォームアップ期間の長さ」「チェインの数」などオプショナルに指定できるものがあります。これは見てもらったほうが早いかと思いますので，同じことをそれぞれのパッケージで実行するときにどのように指定方法を変えるかを見てみましょう。まずは rstan パッケージの書き方からです。

code：2.3 rstan のスタイル

```
fit <- rstan::sampling(model,
  data = dataSet,
  chains = 4,
  iter = 6000,
  warmup = 1000
)
```

ここで指定しているのは，データを dataSet として設定し，同時に 4 本のチェインを発生させ，6000 個の乱数を作って，そのうち 1000 個はまだ機械が温まっていないので候補から除外する，というものです。複数のチェインを作ること，温まっていない部分を捨てる理由などは後ほど説明しますが，結果的に合計 20000 個の乱数が作られることを知っておいてください。この数字は $(6000 - 1000) \times 4 = 20000$ という計算からでてきます。また，乱数の発生は並列計算させた方が良いので，rstan パッケージを使う場合は事前に，options(mc.cores = parallel::detectCores()) という一行を入れておきます。

さて，同じことを cmdstanr パッケージでやると次のようになります。

code：2.4 cmdstanr のスタイル

```
fit <- model$sample(
  data = dataSet,
  chains = 4,
  parallel_chains = 4
  iter_sampling = 5000,
  iter_warmup = 1000,
)
```

ここにあるように，並列するチェインの数をオプションに書き込む必要があります。またサンプルの数は捨てる部分を別にして (含めずに) 計算させることができます。このコードで 20000 個のサンプルが得られます。細かいことですが，多少の違いがあるわけです。

出力の違い

さて，今度は出力結果の使い方についての注意です。rstan パッケージは，出力結果を stanfit オブジェクト型という形にします。rstan のもっているさまざまな関数，たとえば要約した結果の出力やプロットなどは，stanfit オブジェクトだとわかるとそれに対応した出力に変えて表現してくれるわけです。cmdstanr の出力はまた別物[*20]です。

[*20] class 関数で rstan の出力がどういうクラスなのかを表示させると，stanfit と出てくるのですが，cmdstanr の出力を同様にチェックすると"CmdStanMCMC" "CmdStanFit" "R6"という答えが返ってきます。クラスというのはデータの形を定義する方法で，オブジェクト指向プログラミングに必要な概念なのですが，ここではややこしすぎるのでパス。違うということだけわかってもらえれば結構です。RStudio のばあいは Environment タブを見るだけでも違いがわかると思います。rstan パッケージの出力はちゃんとしたクラス (Formla Class stanmodel) なのですが，cmdsatnar パッケージの出力は Environment(グローバル環境の要素) として剥き出しの出力が出ている感じです。

同じモデルをそれぞれのパッケージで出力させて, 違いを見てみましょう。まずは rstan パッケージの出力から。

rstan の出力 1: rstan パッケージの出力

```
Inference for Stan model: ttest01.
4 chains, each with iter=6000; warmup=1000; thin=1;
post-warmup draws per chain=5000, total post-warmup draws=20000.

       mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
mu1   59.96    0.04 5.58  49.01  56.33  59.99  63.60  70.91 16043    1
mu2   40.04    0.05 5.62  28.98  36.38  40.03  43.67  51.17 14330    1
sig   15.54    0.03 3.12  10.86  13.35  15.08  17.23  22.92 12118    1
lp__ -51.58    0.02 1.36 -55.12 -52.18 -51.24 -50.60 -50.05  7115    1

Samples were drawn using NUTS(diag_e) at Tue Nov 15 09:50:17 2022.
For each parameter, n_eff  a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

rstan パッケージが出力しているのは次のような情報です。

- モデル名, チェイン数, 反復回数, 最終的に得られた MCMC サンプル数 (draws) などの説明
- パラメータの事後分布の記述統計量, すなわちベイズ推定による事後分布の情報。順に平均値 (mean), 平均値の標準誤差 (se_mean), 標準偏差 (sd), 2.5%,25%,50%,75%,97.5% パーセンタイル。
- MCMC サンプルについての特徴量。有効サンプルサイズ (n_eff) と Rhat(Rhat)
- サンプリングに関するあとがき

それぞれの内容についてはあとの章で説明するとして, 続けて cmdstanr パッケージの出力をみてみます。

cmdstanr の出力 1: cmdstanr パッケージの出力

```
variable   mean median   sd  mad     q5    q95 rhat ess_bulk ess_tail
lp__ -51.56 -51.20 1.34 1.07 -54.18 -50.12 1.00     7799    10071
mu1   59.99  60.00 5.63 5.47  50.82  69.16 1.00    16325    12764
mu2   40.05  40.05 5.48 5.18  31.09  49.05 1.00    16511    12466
sig   15.48  15.03 3.07 2.75  11.41  21.07 1.00    14489    12137
```

同じような情報ですが, ちょっと違うところもあるようです。前から順に, 事後分布の平均値 (mean), 中央値 (median), 標準偏差 (sd), 平均絶対偏差[*21](mad), 5%,95% パーセンタイルなど分布の特徴[*22]と, Rhat や有効サンプルサイズに関する情報 (ess_bulk,ess_tail) が示されています。しかし, まえがき・あとがきのようなものがなく, 結果だけがシンプルに出力されていますね。

[*21] **中央値絶対偏差 (Median Absolute Deviation)** とは, 中央力のばらつきを表す指標で, 各データ点の中央値からの偏差の絶対値をとり, その中央値を計算したものです。標準偏差とは違う分布の幅を表現する方法で, 中央値を使って計算しますから外れ値に強いという特徴があります。

[*22] rstan パッケージは 2.5% から 97.5% のパーセンタイル, すなわち囲まれる区間の面積が 95% です。cmdstanr パッケージは 5% と 95% のパーセンタイル, すなわち囲まれる区間の面積が 90% です。なぜだかわかりませんが, ちょっとした違いがありますね。

出力としては rstan パッケージの方が丁寧で良いかもしれません。さて，cmdstanr の出力は実は csv ファイルとして一時フォルダに保存されているので，これを rstan パッケージの関数を使って stanfit オブジェクトに変換することができます。関数の使い方の例は次のようになります。

code：2.5 cmdstanr の出力を rstan のオブジェクトに変える

```
fitR2 <- fitC$output_files() |> rstan::read_stan_csv()
```

コードの中身ですが，まず fitC と書いてあるのが cmdstanr の出力オブジェクトです。このオブジェクトがさまざまな情報を持っているという形になっており，ここから出力ファイルの場所を聞き出すのが output_files() です。このファイルを rstan パッケージの read_stan_csv() 関数に渡してやると，csv ファイルを読み込んで rstan のオブジェクトに変えてくれます[*23]。変えたものをここでは，fitR2 オブジェクトに保存しています。この方法を使うと，先ほどの cmdstanr の出力が rstan の出力と同じになります (同じなので再掲しません。)。

このコードを覚えておけば，rstan パッケージ向けに書かれたものであっても，cmdstanr パッケージで実行したあと同じ関数を適用できるので，使いやすいですね。

2.5.3 本書で利用する準備関数

どちらのパッケージであっても，欲しいものは事後分布からの乱数をデータセットにしたものであり，そのデータセットさえ持っていれば，既存の関数を使わなくとも自分で工夫して加工すれば良いでしょう。

そこで本書では，MCMC サンプルを抜きだしてデータセットに変換する関数を作り，これを利用して話を進めることにします。ここではその関数の中身を解説しておきます。

MCMC サンプルをデータフレームにする関数

まずは MCMC サンプルをデータセットにして渡してくれる関数です。この関数は伴走サイトのコードの冒頭に必ず含まれており，以後はこれを使って結果の要約を示すといった使い方をしていますので，中身を知っておいてもらった方が良いかもしれません。自分の使っているパッケージの方だけでも目を通してください。この関数を経由すると，出てくるオブジェクトは同じ形式になっているところがミソです。

■stanfit オブジェクト (rstan パッケージ) の場合 rstan パッケージを使って stanfit オブジェクトとして MCMC サンプルを得た場合，そのオブジェクトから乱数部分だけを抜き出す関数は rstan::extract 関数です。これで取り出したものをデータフレーム型にして返す関数として，次のようなものを作りました。

code：2.6 MCMCtoDF 関数 (rstan 版)

```
MCMCtoDF <- function(fit) {
  fit %>%
    rstan::extract() %>%
    as.data.frame() %>%
    tibble::as_tibble() %>%
    tibble::rowid_to_column("iter") %>%
    dplyr::select(-lp__) %>%
    tidyr::pivot_longer(-iter) -> MCMCsample
  return(MCMCsample)
```

[*23] 当然のことながら，rstan パッケージがないとこの関数も呼び出せませんので，こちらもインストールしておく必要があります。なおこのコードの，|>は tidyverse パッケージのパイプ演算子，%>% と同じ機能を持つ R の演算子で，ネイティブパイプと呼ばれています。ネイティブパイプは R4.1.0 以降に導入されたものです。

```
10  }
```

■**コード解説**

1 行目　関数を作る宣言。引数として stanfit オブジェクトをとります。
2 行目　引数で引き受けたオブジェクトを加工していきます。
3 行目　まずは MCMC サンプルを抜き出します。
4 行目　data.frame 型にします。
5 行目　tibble 型にします。別にしなくてもいいんですが, 著者の好みです。
6 行目　行番号を表す変数 iter を作ります。
7 行目　変数の中から lp__を除外します。
8 行目　行番号変数は残して，tidy なデータにし，MCMCsample というオブジェクトに保存します
9 行目　戻り値として MCMCsample を返します。

■**cmdstanr パッケージの場合**　cmdstanr パッケージを使って MCMC サンプルを得た場合, そのオブジェクトから乱数部分だけを抜き出す関数は draws 関数であり, これをオブジェクトに直接作用させます。これで取り出したものをデータフレーム型にして返す関数として, 次のようなものを作りました。

code : 2.7　MCMCtoDF 関数 (cmdstanr 版)

```
1  MCMCtoDF <- function(fit) {
2    fit$draws() %>%
3      posterior::as_draws_df() %>%
4      tibble::as_tibble() %>%
5      dplyr::select(-lp__, -.draw, -.chain, -.iteration) %>%
6      tibble::rowid_to_column("iter") %>%
7      tidyr::pivot_longer(-iter) -> MCMCsample
8    return(MCMCsample)
9  }
```

■**コード解説**

1 行目　関数を作る宣言。引数として stanfit オブジェクトをとります。
2 行目　引数で引き受けたオブジェクトに draws 関数を作用させ, サンプルだけ取り出します。
3 行目　取り出したサンプルをデータフレーム型にする posterior パッケージの as_draws_df 関数を適用します。
4 行目　tibble 型にします。別にしなくてもいいんですが, 著者の好みです。
5 行目　変数の中から lp__や他の隠し変数を除外します。
6 行目　行番号を表す変数 iter を作ります。
7 行目　行番号変数は残して，tidy なデータにし，MCMCsample というオブジェクトに保存します
8 行目　戻り値として MCMCsample を返します。

■**MCMCtoDF 関数の結果**　出力結果は, どちらも同じく以下のような形式になります。

R の出力 2.1: MCMC サンプルをデータセットにしたもの

```
# A tibble: 60,000 × 3
    iter name  value
   <int> <chr> <dbl>
 1     1 mu1    60.3
 2     1 mu2    41.0
 3     1 sig    12.5
 4     2 mu1    62.9
 5     2 mu2    37.6
 6     2 sig    11.5
 7     3 mu1    68.7
 8     3 mu2    46.4
 9     3 sig    13.7
10     4 mu1    67.7
# … with 59,990 more rows
# ⨂ Use `print(n = ...)` to see more rows
```

ここで iter とあるのは MCMC のステップ番号，name とあるのが変数名，value とあるのがその値です。tidy な形になっていますから，このまま分析や描画に用いることができます。

MCMC サンプルの結果を要約する関数

MCMC の結果を分析するにあたっては，bayestestR パッケージなど既存のパッケージを使うと便利でしょう。ですが，こうしたパッケージはどんどん開発が進んでアップデートされたり，rstan か cmdstanr かで書き方が変わったりするなど，テキストで紹介するには難しいところがあります。そもそも MCMC サンプルを加工して記述統計を示したり，描画したりするわけですから，上で紹介したように MCMC サンプルを抜き出すことができれば自力でもできるはず。ということで，自作の関数を用意しました。本書では以下，この関数をつかって解説しますので中身を知っておくと良いでしょう。

■MAP 推定関数　確率分布の特徴を報告するときに，その確率分布に従う乱数を使って，期待値や中央値を計算するのは簡単です。記述統計としての平均値やパーセンタイルでよいからです。しかし確率分布の確率密度が最も高くなるところ，すなわち **MAP 推定値 (MAP Estimaiton)** の計算はちょっと難しいですね。というのも，関数であれば微分して極値を求めれば良いのですが，MCMC サンプルは関数ではなくて値なので，ヒストグラムを書くしかなく，最大の値を求める計算がむずかしいからです。

でも大丈夫，R の描画関数を駆使して対応することができます。MAP 推定値を計算する関数は，次のように書きます[*24]

code：2.8　MAP 推定する関数のコード

```
1  map_estimation <- function(z) {
2    density(z)$x[which.max(density(z)$y)]
3  }
```

これは R の density 関数でヒストグラムに密度関数を当てがい (カーネル密度推定)，その関数の特徴から値を返すようになっています。

[*24] このコードは関西学院大学社会学部の清水裕士先生に教えてもらったものです。記して謝意を表します。

■MCMC サンプルの要約関数 先ほどの MAP 推定する関数を含めつつ, また MCMC サンプルをデータフレームにする関数を使いつつ，MCMC サンプルの要約を表示する関数を次のように作りました。

code：2.9 MCMC の要約を報告するコード

```
1  MCMCsummary <- function(MCMCsample) {
2    MCMCsample %>%
3      dplyr::group_by(name) %>%
4      dplyr::summarise(
5        EAP = mean(value),
6        MED = median(value),
7        MAP = map_estimation(value),
8        SD = sd(value),
9        U95 = quantile(value, prob = 0.975),
10       L95 = quantile(value, prob = 0.025)
11     ) %>%
12     mutate(across(where(is.numeric), ~ num(., digits = 3)))
13 }
```

■コード解説

1 行目 関数を作る宣言。引数として MCMCtoDF 関数の戻り値をとります。

2 行目 引数で引き受けたオブジェクトを加工していきます。

3 行目 変数名でデータをグループ化します。

4-10 行目 記述統計の計算を通じて，EAP,MED,MAP,SD,95% 区間を返します。

11 行目 出力する数値データは, 少数下 3 桁まで表示させるようにします。

この関数は, 先ほどの MCMCtoDF とセットで使い, 次のような結果を得ます。

R の出力 2.2: MCMCsummary の結果

```
> fit %>% MCMCtoDF() %>% MCMCsummary()
# A tibble: 3 × 7
  name        EAP       MED       MAP        SD       L95       U95
  <chr> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!>
1 mu1      59.897    59.888    58.867     5.547    48.672    70.847
2 mu2      39.958    39.935    39.748     5.523    28.973    50.943
3 sig      15.519    15.062    14.646     3.074    10.877    22.788
```

ここにあるように, 各変数の EAP 推定値，MED 推定値，MAP 推定値，95% 確信区間が表示されます[*25]。

[*25] この区間はコードから明らかなように，quantile 関数で計算しています。下から 2.5%, 上から 2.5% を取り除くと 95% の区間が残ることになりますが, このように分布の両端を均等に切った区間を厳密には**等裾区間 (Equal-tailed interval;ETI)** というのでした。これに対して, 分布の最も信頼できる部分, かつ分布の大部分をカバーし, 区間内部のすべての点が, 区間外の任意の点よりも高い密度を持っているように推定するものを**最高密度区間 (Highest-Density Interval;HDI)** と呼びます。これらは左右対称の単峰分布であれば同じになるのですが, 事後分布の形がぐちゃっとしている時は必ずしもそうなりません。両者の違いについては Kruschke (2014 前田･小杉監訳 2017) を参考にしてください。また，HDI を出力する関数は bayestestR::hdi() です。

740 これらの推定値や，そもそも事後分布が何をどのようなことを表しているのか，といった点については今後
741 の授業で説明していきます。以下は結果をしれっとこの関数で表現していきますので，何をやっているのかわ
742 からなくなったらここに立ち戻ってきてください。

第 3 章

モデリングの目から見た検定 1；二群の平均値の差

前回はデータ生成メカニズムという観点で考えることで，標準偏差 (分散) を「測定精度」と考えた検討ができることを示しました。今回は，心理学でもよく使われている平均値の差を検討対象とするモデルを考えることにします。

皆さんは帰無仮説検定のことを覚えているでしょうか。正規分布を仮定した母集団からの標本統計量は，正規分布に従うことを利用して，母平均の区間推定を行い，群間に差があるかないかといった判断を確率的に行うのが帰無仮説検定でした。心理学では要因計画と帰無仮説検定が合体し，群間の差の形でデータが得られるようにすることで結論を導き，考察を重ねてきました。また帰無仮説検定という手法は，誤用や誤解が多く，ひどい時には悪用もされるということについてもみてきたと思います。ところで，帰無仮説検定の考え方はデータが得られたところから考え始めるデータ駆動型の分析モデルでした。ではデータがどのように出てきているのかを考える，**データ生成モデリング**の考え方をつかうと，この平均値の差についての検討はどのように姿を変えるのでしょうか。

ここでは **t 検定**を例に話を進めていきたいと思います。

3.1 t 検定の過程と実際

t 検定は，二群の平均値の差を比較し，結論を下すという最も典型的な帰無仮説検定の例の 1 つです。とくに独立した二群の検定は，最も単純な要因計画 (一要因二水準 Between デザイン) ということができるでしょう。これについて，よく思い出せない人はデータ解析基礎の資料や，山田・村井 (2004)，清水 (2021) などを読んで，分析の流れや仮定をもう一度確認しておいて欲しいと思います。

非常に駆け足ながら概略を説明してみますと，次のようになります。

1. 標本の母集団が正規分布していると考える。母集団から無作為に選ばれた標本が二群あるとする。
2. 二群の一方に何らかの処置を加え，他方には何もしない (あるいは検討したい処置と同等の効果のない処置を施す[*1])。前者を実験群，後者を統制群という。データ (標本) は正規分布するはずだから，その平均値を見ることで誤差や個人差は相殺される。つまり群間の平均値の差が効果の大きさであるということができる。

[*1] たとえば「単語リストは声に出して覚えたほうが記憶の定着度が高い」ということを検証したい場合，声に出す群と出さない群で比較することになりますが，声に出さないことが (頭の中で反芻するなど) 従属変数に与える別の効果を持っていることがあるので，音のない映像を見せるなどして効果のない同等の処置をした群を比較対象にする，ということを考えたりするわけです。

3. ところで標本平均値などの**標本統計量**も正規分布に従うことがわかっている。母集団が $N(\mu, \sigma)$ であれば, 標本平均は $N(\mu, \frac{\sigma}{\sqrt{n}})$ に従う (ここで n はサンプルサイズ)。
4. 標本平均が従う分布 (散らばりの位置と幅) が分かれば, 母平均がどのあたりにあるのかは区間推定することが可能。実験群・統制群ともに母平均の値を推定する。これが異なっていれば標本を超えて母集団で効果があった, と結果を一般化できる。もちろん点推定値は母平均の値とピッタリ同じとは思えないし, 標本平均もピッタリ同じになるはずがないから, 点推定値ではなく区間推定で考えたい。
5. ところで標本平均の従う分布の中には, σ つまり母 SD がパラメータとして入っている。母数がわからないという前提のもとでは, どの程度の幅で散らばるのかがわからないことと同じ。そこで σ の代わりに $\hat{\sigma}$ を用いて推定することを考える。この場合, 標本平均は正規分布ではなく t 分布に従うことがわかっている。
6. t 分布を使った区間推定をしても, 差があるのかないのか判断するために何らかの基準が必要である。そこで「差がない」という主張と「差がないとは言えない」という主張を戦わせて, 判定すると言う形式を取る。前者の主張を**帰無仮説**, 後者の主張を**対立仮説**という。また判断も確率的になるので, 勝敗を決める基準を 5% とする。この確率は**有意水準**とか**危険率**と呼ばれる。
7. データから t 分布に従う統計量を計算し, その値が出てくる確率を算出する。この確率は p **値**と呼ばれる。これが有意水準よりも小さいようであれば, 帰無仮説の仮定の下で算出した数字が滅多に生じ得ないことを意味するから, 帰無仮説が間違っていたのだと判断してこれを**棄却**し, 対立仮説を**採択**する。

さて, この二群の平均値を検定することの流れですが, とくに「帰無仮説のもとでの t 値を算出して判定する」というあたりがややこしかったかもしれません。数式で書くと嫌われそうですが, 帰無仮説というのは二群の平均値に差がない, という仮定だったので, 実験群から考えられる母平均 μ_A と統制群から考えられる母平均 μ_B に差がない, つまり $\mu_A = \mu_B$, からの $\mu_A - \mu_B = 0$ という位置母数が 0 の理論分布のを考えている, というところがポイントです。検定統計量である t 値は次のように計算できるのでした。

$$t = \frac{\bar{X}_A - \bar{X}_B}{\sqrt{\frac{(n_1-1)\hat{\sigma_A}^2+(n_2-1)\hat{\sigma_B}^2}{n_1+n_2-2}(\frac{1}{n_1}+\frac{1}{n_2})}}$$

とっても複雑な式に見えますが, 複雑そうなのは分母で, 分子は標本平均の差を表しているに過ぎません。参照する t 分布は, 標準正規分布が $N(0,1)$ だったように, 位置と幅のパラメータを持ち, $t(0, df)$ で考えられる分布にこの統計量を照らし合わせると, 差 0 で自由度に応じて変わる幅 df の分布から「どの程度極端な値が出てきたのか」の確率を計算できるわけです。ちなみに分母は, 母分散ではなく標本から計算される分散 s^2 からバイアスを除いた不偏推定量 $\hat{\sigma}^2$ を使っています。この計算はサンプルサイズ n ではなく $n-1$ で割ることによって計算されるのですが, 2 つの群それぞれについて $n_j - 1$ で割ったものを足し合わせるために, いったん $n_j - 1$ 倍して二群のサンプルサイズ -2 で割り直す, という作業をしているため, 分母がとくに複雑に見えているだけです。

ともあれ, こうして検定するんだったという流れを思い出したところで, データ生成モデリングの観点からこれを考え直してみましょう。仮定としておいてあるのは, 両群ともに同一の正規分布から得られた標本であり, 操作によってその平均値が異なっているはず, ということだけです。つまり実験群のデータは $X_{i,A} \sim N(\mu_A, \sigma)$, 統制群のデータは $X_{i,B} \sim N(\mu_B, \sigma)$ という分布が前提とされているのです。

これをそのまま設計図にし, コードにしてみましょう。設計図として図 3.1 が, これをもとにコードが `code:3.1` のようにかけていればいいでしょう。コードが設計図をほぼそのまま文字起こししていることを確認してください。

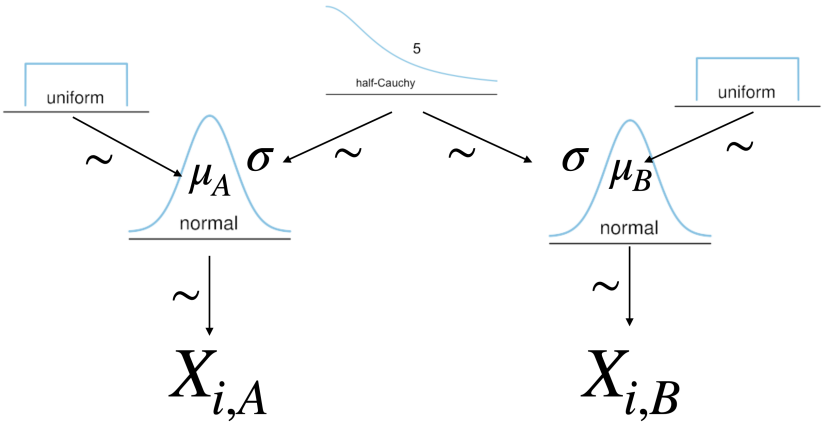

図 3.1 平均値の差を比べるときの設計図

code : 3.1 二群の平均値のコード

```
data{
    int<lower=0> N1; // Number of Subjects in Group 1
    int<lower=0> N2; // Number of Subjects in Group 2
    array[N1] real X1; // Data in Group 1
    array[N2] real X2; // Data in Group 2
}

parameters{
    real mu1;
    real mu2;
    real<lower=0> sig;
}

model{
    // likelihood
    X1 ~ normal(mu1,sig);
    X2 ~ normal(mu2,sig);
    // prior
    mu1 ~ uniform(0,100);
    mu2 ~ uniform(0,100);
    sig ~ cauchy(0,5);
}
```

ここで Stan コードにちょっとした工夫を 2 点入れています。1 つ目は data ブロックにある，変数 $N1, N2$ の存在です。二群のデータについて，サンプルサイズがとくに決まっていませんから，ここで外部から入力することにしています。2 つの群それぞれのサンプルサイズをデータとして取り込み，その数と同じだけデータ数を配列として宣言しているところがポイントです。もう 1 つのポイントは，尤度のところです。丁寧に書くならば，次のようにするべきでしょう (コード 3.2)。

code : 3.2 丁寧な尤度の記載

```
    ...
```

```
2  model{
3      for( i in 1:N1){
4          X1[i] ~ normal(mu1 , sig);
5      }
6      for( i in 1:N2){
7          X2[i] ~ normal(mu2 , sig);
8      }
9      ...
10 }
```

このように，それぞれの群にいて for 文を回し，各データ点が正規分布から出てきているよ，ということを明示するのです。しかしそうしなかったのは，Stan が「分かりきったことは書かなくていいよ」という優しい設計になっているので，変数 X1 の要素すべてがある分布に従うのであれば，まとめて X1 ~ normal(mu1, sig); と書いても良い，つまり X1 の配列要素を逐一指定しなくても同じ尤度関数をあてがってくれるというのを利用しています。

ともかくこれでできるのでした。では少し具体例をつかって，これがどういう結果をもたらしているのかを確認しましょう。

3.1.1 t 検定の具体例

> 統計学の新しい指導法の教育効果を見るため，全国の大学の心理学科から学生を無作為に 16 名選び，従来の指導法グループ（統制群）と，新指導法のグループ（実験群）にランダムに 8 名ずつわりつけました。プログラム終了後，心理統計のテストを行いました。統制群のスコアは $20, 40, 60, 40, 40, 50, 40, 30$，実験群のスコアは $30, 50, 70, 90, 60, 50, 70, 60$ となりました。新たしい指導法は，心理学科の学生に効果があると言えるでしょうか？ 5% 水準で検定してください。

これを t 検定するのは簡単ですね。計算式は複雑でも，機械がやってくれるから楽なのが NHST[*2]のいいところです。

code : 3.3 帰無仮説検定のコード

```
1  groupA <- c(30, 50, 70, 90, 60, 50, 70, 60)
2  groupB <- c(20, 40, 60, 40, 40, 50, 40, 30)
3  ## t 検定
4  t.test(groupA, groupB, var.equal = TRUE)
```

結果は出力 3.1 のようになります。検定統計量である t 値，検証のための自由度 df，帰無仮説のもとでの出現確率 p 値，が表示されています。5% より小さいので有意差あり，という判断ができます。もっとも，t 値とか自由度とかは，データに関係のない数字なのでピンと来ないというところもあるかと思います。

[*2] **帰無仮説検定 (Null Hypothesis Significance Test)** の略です

R の出力 3.1: 検定の結果

```
> t.test(groupA, groupB, var.equal = TRUE)

 Two Sample t-test

data:  groupA and groupB
t = 2.6458, df = 14, p-value = 0.01919
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  3.786937 36.213063
sample estimates:
mean of x mean of y
       60        40
```

3.1.2 モデルで推定してみる

つづいて，先ほどのコードを使ってモデリングによる推定をしてみたいと思います。私の環境下では，次のような数字になりました。

MCMC の結果 1

```
# A tibble: 3 × 7
  name        EAP       MED       MAP        SD       L95       U95
  <chr> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!>
1 mu1      60.006    59.976    59.700     5.608    48.869    71.420
2 mu2      39.990    39.997    39.650     5.679    28.654    51.128
3 sig      15.534    15.048    14.070     3.111    10.893    22.877
```

これを見ると，未知のパラメータ μ_1, μ_2, σ の値がそれぞれ推定されていますが，「判定」のような結果は出てきていません。というのも，勝負するような形で考えているのではなく，パラメータがどこにありそうか，ということを示す**確率分布**を求めているからです。それでも，**EAP** をみると 60006 と 39.990，95%CI をみると実験群は [48.869, 71.420]，統制群は [28.654, 51.128] とあり，実験群の平均値が 51 より小さい値になる確率はほとんどなく (2.5% 以下)，統制群の平均値が 49 以上より大きくなることもまたほとんどないわけですから，確実に差があると言ってもほぼ過言ではない，と言い切れるでしょう。

モデリングの利点は，t 値や p 値のような直接のデータと関係ない値を出すのではなく，データに直接関係する数字で考えさせてくれるので，イメージしやすいところもあるかもしれませんね。

3.1.3 分散の等質性

ところで，t 検定の場合は「分散が同じであるかどうか」という条件によって，算出方法を補正するという考え方があるのを覚えていますでしょうか。**Welch の補正**というやつで，こちらの方が一般に条件が緩いものですから，t 検定では普通こちらの方が使われます。

R の出力 3.2: Welch の補正の例

```
> t.test(groupA, groupB, var.equal = FALSE)

 Welch Two Sample t-test

data:  groupA and groupB
t = 2.6458, df = 12.274, p-value = 0.021
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  3.570406 36.429594
sample estimates:
mean of x mean of y
       60        40
```

出力 3.2 にあるように，オプションとして指定するだけで瞬時に答えが出てきます。t 値，自由度，p 値が先ほどと異なって出ていますが，結果はほぼ変わりがありません。

モデリング上ではこれをどう表現すれば良いでしょうか。これは実は簡単で，分散が実験群と統制群で異なるというのですから，別々に推定してやれば良いのです。

code : 3.4　分散が異なる場合の推定モデル

```
1          ...
2  parameters{
3      real mu1;
4      real mu2;
5      real<lower=0> sig1;
6      real<lower=0> sig2;
7  }
8
9  model{
10     // likelihood
11     X1 ~ normal(mu1,sig1);
12     X2 ~ normal(mu2,sig2);
13     // prior
14     mu1 ~ uniform(0,100);
15     mu2 ~ uniform(0,100);
16     sig1 ~ cauchy(0,5);
17     sig2 ~ cauchy(0,5);
18 }
```

3.2　差の分布

さて，二群のデータから考えられるそれぞれの母平均が推定されました。今回は実験群と統制群の平均値差が大きく離れており，一方の 95% 上限が他方の 95% 下限を上回っているという状況でしたので，「差がある」と判断できましたが，そうでないこともありそうです。つまり，微妙な差のとき，ですね。次のようなデータで試しに推定してみましょう。まずは 5% 水準で検定をしてみます。

code : 3.5　帰無仮説検定のコード

```
groupA <- c(30, 50, 70, 90, 60, 50, 70, 60)
groupB <- c(30, 45, 60, 40, 60, 50, 40, 30)
## t検定
t.test(groupA, groupB, var.equal = FALSE)
```

今度は $t(12.175) = 2.0761, p = 0.0597$ で有意とは言えなくなりました[*3]。このデータを，先ほどのモデルで推定してみましょう。目にも分かりやすくするために，推定された平均値の事後分布をプロットしてみたいと思います (図 3.2)。

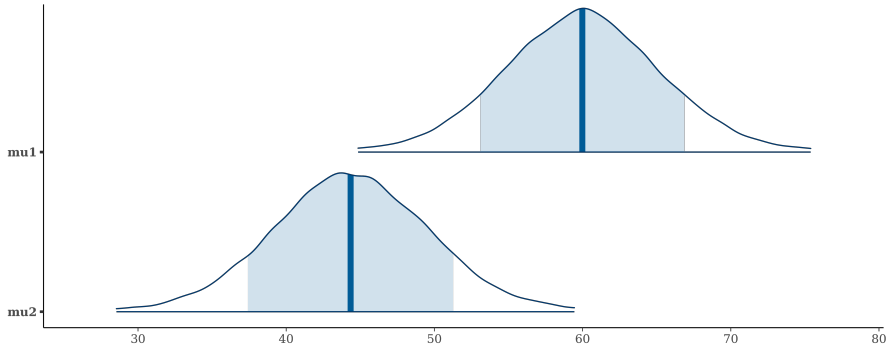

図 3.2 二つの平均値パラメータの事後分布

今回は 80% 確信区間と，確率分布の両端を 99% のところまで描画してみました。2 つの平均値パラメータは，分布の代表値で見ると異なっているのですが，重複している領域が多く一概に「一方が他方より大きい」と言いにくい結果になっています。帰無仮説検定のようにズバっと「差がある」「ない」と言い切れればいいのですが，このような場合はどうしたらいいでしょうか。

正解は「このままにしておく」です。急いで差があるとかないとか言い切るのではなく，これが区間推定の結果そのものですから，この程度の重複があり得るもの・それほど明確な答えは出ないもの，として考え続けるしかありません。

しかしまあ，このままでは考えるヒントが少ないので，ここで少し工夫をしてみましょう。帰無仮説検定では，$\mu_A = \mu_B$ という帰無仮説が反駁できるかどうかが問題なのでした。今回，μ_A や μ_B を代表する数字の候補が色々ありますので，これを使って「実際どの程度差があったのか」を計算してみれば良いのです。MCMC サンプルは，毎回推定したいパラメータの同時分布からの代表値を持ってきているという話をしました。たとえば 1 回目のサンプルは $\mu_A = 62.4, \mu_B = 25.7$，2 回目のサンプルは $\mu_A = 60.4, \mu_B = 36.8$ と言った具合に，です。それらは事後分布からのサンプル，事後分布を代表した実現値の 1 つですから，これを使って $\mu_A - \mu_B$ の計算をできます。すると**差の分布**の代表値が MCMC サンプルで計算できることになります。

これを使って，$\mu_A - \mu_B = 0$ になる確率はどれぐらいか，というのを考えれば良いのかもしれません。もっとも，連続変数におけるある一点の確率はゼロですし[*4]，代表値といってもピッタリゼロになることはほぼあり得ないでしょうから，「どの程度ならゼロと見做すか」ということを考える必要があります。この範囲は**実質的に等価な範囲 (Region Of Practical Equivalence; ROPE)** と呼ばれ，たとえばこのテストの例では ± 5 点ぐらいは誤差みたいなものだけど，5 点以上変わったら意味のある違いだ，ということであれば「5 点以上点差が出た確率」を計算できます！

この計算は，MCMC サンプルを取り出した R のオブジェクトを使って計算することもできますが，実は

[*3] 5.9% だから惜しい！とか「有意傾向にある」なんて変なこと言い出したらダメですよ。勝負は 5% だと決めて始めたのですから。

[*4] 確率は相対的な面積で表される数字ですから，面積を持たない点については確率が定義できないのです。

Stan の中で計算させてしまうこともできます。`generated quantities` ブロックが, この MCMC サンプルを取り出したあとの処理をするブロックに該当します。ここで計算される量のことを**生成量 (generated quantities)** と呼びます。実際にどのように使うのかをみてみましょう。

code : 3.6 生成量ブロックの利用

```
1       ...
2  generated quantities{
3       real diff;
4       int<lower=0, upper=1> FLG;
5       diff = mu1 - mu2;
6       if(diff > 5){
7           FLG = 1;
8       }else{
9           FLG = 0;
10      }
11  }
```

コード 3.6 には生成量ブロックを使う例を示しました。ここもブロックですので, 変数の宣言が必要です。今回はまず `diff` という変数を実数型で宣言してみました。これはその下で $\mu_1 - \mu_2$ として定義されていることからわかるように, 差の分布 (の代表値) を生成す流のです。これを見ると, 差の生じる確率を考えることができます。次に整数型 (int 型) で `FLG` という変数を宣言しました。上限が 1 で下限が 0, というか実質的には 0 か 1 の数字しか入らない, ある/なし, 成立する/しないといった情報しか持たない変数です。いわゆる「フラグが立ったかどうか」の指標であり, その実態は下の `if` 文で表現されています。ここでは先ほど計算した `diff` が 5 よりも大きいのであれば 1, そうでなければ 0 を代入する, というコードになっています。

これを実行すると, パラメータのサンプルに加えて, そのパラメータから計算されるさまざまな量も同時に算出されます。出力例を見てみましょう。

MCMC の結果 2

```
# A tibble: 6 × 7
  name        EAP       MED       MAP        SD       L95       U95
  <chr> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!>
1 diff     15.650    15.605    14.940     8.156    -0.516    31.706
2 FLG       0.911     1.000     1.000     0.285     0.000     1.000
3 mu1      60.035    60.035    60.112     6.737    46.602    73.575
4 mu2      44.385    44.374    44.291     4.594    35.249    53.696
5 sig1     18.509    17.472    15.960     5.369    11.414    31.792
6 sig2     12.496    11.752    10.761     3.683     7.629    21.594
```

ここにあるように, 差の 95% 確信区間は `[-0.516, 31.706]` であることがわかります。差が大きければ 31.70, 小さければ逆転して-0.52 になる可能性すらあるのです。この確信区間は 0 を含んでいますから, 差がない可能性は否定できません。そういう意味で, 帰無仮説検定的表現を借りるなら, 5% 水準で差がないとは言い切れない, ということができるでしょう。

とはいえ, せっかく分布として情報が得られているのに, 点推定にして勝負をするのはもったいないともいえるわけです。結論を急がずに, どの程度の差があるのかをじっくり考えてみるのがいいでしょう。また変数 `FLG` を見ると, その平均が 0.908 となっています。これは変数 `diff` が 5 よりも大きい時は 1, そうでなければ 0 という条件の数字で, その平均は 1 になった数を総 MCMC サンプル数で割った相対度数になっている

のですから，確率を表す数字だと考えても良いことになります。つまり，差が 5 よりも大きくなる確率は 90.8% である，と考えることもできるのです。これを応用すると，さまざまな仮説を考えることができそうですね。

3.3 帰無仮説検定を省みる

さてこのように考えてくると，帰無仮説検定について異なる視点で見ることができるようになったのではないでしょうか。ここでは，帰無仮説検定の独特さを 3 つの点から考えてみたいと思います。

3.3.1 不平等な対決

帰無仮説検定は帰無仮説 vs 対立仮説という勝負に持ち込んで判断する方法である，ということを冒頭に述べました。平均値の差の検定の文脈で言えば，帰無仮説は $H0 : \mu_1 = \mu_2$ であり，対立仮説は $H1 : \mu_1 \neq \mu_2$ です。このように，帰無仮説は「差がない」の一点張りであり，対立仮説は「それ以外ならなんでもあり」という状態を指しているのでした。そもそも，帰無仮説というのは非常に限定的な仮説だったのです。無に帰してほしい仮説とは言え，あまりにも不利な勝負なのでした。その上で，勝負は「差がない」「ある」のどちらかにしかなりません。ごく微妙な差であっても，判定によれば差が「あった」といえますし，ギリギリでもなければ「なかった」というしかないのです。このような過度に結果を際立たせる報告は，科学的な研究実践の文脈では拙速なことになりかねませんので，注意が必要なのでした。

データ生成モデリングのやり方で同じデータを分析してみると，差も分布の形で描かれますから，「あったか，なかったか」という単純な問題にせずに色々考えてみたくなるのではないでしょうか。

3.3.2 量的な判断を

もっとも，出力 3.2 の t 検定の結果をよくみてみると，t 値や p 値の下に 95% 信用区間が示されており，ここでは [3.570406, 36.429594] という数字が出ています。そうです，帰無仮説検定をやっているようでも，しっかり区間推定した結果も表示されていたのです。これは差の信用区間で，この区間が 0 を跨いでいないということは，差がないとは言えない $(\mu_A - \mu_B = 0)$，ということを意味しています。この区間の幅をみると，3 から 36 と随分ひろくとられているように思えます。つまり，今回の検定結果は差があるとなったけれども，それほど確かなものではないかもしれないぞ，と慎重に判断することもできたはずなのですね。

この差についての量的な評価については，**効果量 (effect size)** という指標で表現されるのでした。効果量とは，標準化された差の大きさです。標準化するというのは，標準偏差で割って幅を整える，あるいは標準偏差何個分という単位で表現することでもあります。比べられるスコアはさまざまですから，その標準偏差で表現することで一般的に議論できるのです。平均値の差の検定においては，**Cohen の d(Cohen's d)** などが指標としてもいいられますが，それは $\frac{\bar{X}_1 - \bar{X}_2}{\sigma}$ で計算されるのでした。

これは生成量を使って簡単に計算できます。今回の場合，計算する場合は，生成量に次のようにすれば良いのです。

code : 3.7 生成量ブロックで効果量の算出

```
1      ...
2  generated quantities{
3      real diff;
4      real cohen_d1;
5      real cohen_d2;
6      diff = mu1 - mu2;
```

```
7       cohen_d1 = diff / sig1;
8       cohen_d2 = diff / sig2;
9   }
```

どちらの群の標準偏差を基準にするか，ということを考える必要がありますが[*5]，相対的な大きさの比較も簡単にでき，しかもその結果も分布の形で得られる，すなわち効果量の確信区間 (効果が少なくともどの程度ありそうかとか，大きければどの程度ありそうかとか) を考えることもできるのです。

もちろんこれらの計算は，帰無仮説検定の文脈，言い換えれば伝統的な統計手法 (**モーメント法**による推論) であってもできたことなのですが，データ生成モデルという観点からみるといっそう理解がしやすいのではないかと思います。

3.3.3 方向性を持った仮説も

今回は最後に，「実験群が統制群よりも 5 点以上大きくなる確率」というのを考えました。この「一方が他方よりも大きい」という仮説は，帰無仮説検定の文脈では**片側検定 (one-tailed test)** と呼ばれます。一般的に「差がある」という対立仮説は，プラスであれマイナスであれ違いが生じている，という意味であり，統計量は分布の右側でも左側でも，とにかく極端な値になりさえすれば良いという判断でした。そうではなく「$A > B$ のはずだ」という仮定であれば，小さくなる可能性を考えなくていいのですから，統計量のどちらか一方の極について考えれば良いことになります。

とはいえ検定ですから，ある有意水準で一方が他方より大きいと判断すると間違える確率が云々，という判定に落とし込むこという点では同じでした。

今回は生成量を使って，5 点以上大きくなる**確率**を求めることができました。これは検定統計量や p 値のように架空の値ではなく，もとのデータに直結した数字や確率になっていますから，解釈が比較的自然にできるというのが利点です。プログラム上の表現も，たとえば今回のように `if` 文を使って表現するのは非常に直感的で，「もしあれがこうなってそれがこうなったらどうなる？」といろいろな仮説を考えていくこともできますね。帰無仮説検定のロジックは，結果判定のために仮説を限定的な型にはめ込んでしまいます。しかしその型を飛び越えていろいろなことをやってもいいのだ，できるんだというのは，データがどうやって生まれてきているかを考えている「創造主」としての特権かもしれません。

ただし注意してもらいたいのは，これらの検証の仕方は，今回のデータと仮定されたモデルという前提の上で成立する割合を確率とみなしているに過ぎない，ということです。データが変わればその数字も変わるでしょうし，そもそもデータが正規分布から出てきていないのかもしれません。本当は XX 分布から出てきているのに，YY 分布を仮定したモデルで計算したら，仮説が正しい可能性が 100% !といっても虚しい (明らかに間違っている) ことになりますね。心理学のデータの場合はえてして，どういう分布やメカニズムになるのかがはっきりわからないものです。ただ，あまりにもわからなさ過ぎて，さまざまな影響が混ぜ合わさっているので，色んな要素が相殺しあって結局ほとんど正規分布とみなしていいよ，ということだったりします。心という目に見えないものに，不良設定問題を解くためのツールで立ち向かうという，無理に無理を重ねるような推論の世界であるということに，自覚は持っていたいですね[*6]。

[*5] 二つの標準偏差の平均をとった，プールされた標準偏差で計算することもあります。

[*6] いいんです，それでも。だってそんなよくわからない人間というのが好きなんですから。

3.4 今回のまとめ

我々がデータを分析する時は，そのデータの数字がどうであったか，ということを真っ先に考えたいはずです。身長の差は何センチあったのか，最終学歴によって生涯年収は何円差がつくのか，といった具体的な単位に基づく**実質的な差**が一番大事なはずです。心理学をやっていくと，尺度をはじめとした「絶対的なスコア」が出てきませんから，尺度で何ポイント差があったのか，ということがあまり意味のある実態と対応しないことも少なくありません。そう言った時のために，**標準化された差**を考えるようになったのです。その計算は非常に面倒ですが，機械がすぐに計算してくれることによって結論を急ぎすぎ，単位も実際のデータとも関係のない統計量を参照して**有意差**を求めるようになってしまったのでは意味がありません。

データがどういうメカニズムで生まれてきたかを考え，母数をベイズ推定するというやり方は，いちいちコードを書かなければならないこともあって，非常に手間がかかるようにも思えます。しかし自分が何を仮定していたのか，どういうメカニズムの元で考えていたのかを自覚し，またその推定値を使って自由に仮説を考えることで，決まりきった分析手続きに落とし込むという単調な作業から解放され，クリエイティブにデータに向き合えることでもあります。

3.5 課題

次の計算をする R/Stan コードや回答を記述し，提出してください。なお提出されたコード単体でバグがなく動くことが確認できないものは，未提出扱いになります。コードの書き方などわからないところがあれば，曜日別 TA か小杉までメールで連絡し，指導を受けてください。

■フライドポテトの研究 あるコンビニで，ホットスナックのフレンチフライを 2 つ買ったところ，どうも一方より他方の方が長くて大きい気がした。そこでそれぞれの袋から取り出して長さを測定してみた (単位 cm)。測定結果が表 3.1 である[*7]。このデータを用いて次の問に答えなさい。なおこの数値はシラバスのサイト上

表 3.1 二つの店舗のポテトの長さ

A	8.4	11.3	8.1	11.2	5.8	6.3	7.1	10.9	7.1	6.5	5.0	3.0	7.2	6.5	6.4	6.4	9.3	8.3
B	6.7	7.2	4.2	11.0	7.5	8.9	7.0	8.0	7.2	4.2	6.0	9.0	8.6	9.0	5.0			

のコードで取得可能です。

1. ポテトの長さは正規分布に従うと仮定します。また両群の分布の幅は同じであると仮定します。t 検定で二群の平均値に差があるかどうか，判断してください。t 値や自由度，p 値を参照しながら説明すること。効果量も算出するとなお良いです。
2. ポテトの長さは正規分布に従うと仮定します。また両群の分布の幅は同じであると仮定します。その上で，群 A と群 B の平均値を推定するモデルを作り，ベイズ推定してください。その上で，結果の平均値，中央値，MAP 推定値，90% 確信区間を報告してください。
3. 両群のポテトの長さの平均が，3cm 以上違うようであればクレームをつけに行こうと思います。3cm 以上の差がある確率はどれぐらいですか。推定してください。

[*7] このデータは実際に筆者がコンビニの二つの店舗でポテトを購入し，長さを測定した結果に基づいています。

4. よく考えてみれば，両郡の分布の幅が同じであるという仮定はおかしいような気がしてきました。そこで，それぞれの群毎に分布の幅を推定するモデルに作り替え，ベイズ推定してください。その上で，結果の平均値，中央値，MAP 推定値，90% 確信区間を報告推定してください
5. 異なる分散を指定したモデルで，両群のポテトの長さの平均差が 5cm 以内であれば，クレームをつけにいくのはやめておこうと思います。差が 5cm 以内である確率はどれぐらいですか。推定してください。

第 4 章

モデリングの目から見た検定 2；パタメータの世界とデータの世界

前回は，二群の平均値差の検定を行うデータ生成モデルを考えました。平均値差の検定には正規分布が仮定されますから，データ生成モデルも正規分布からデータが作られていると考えれば良かったわけです。仮定に忠実に設計図を書き，それに対応した Stan コードを書くと母平均の推定値が出てきます。検定の時も母平均の推定値を使って考えるのですが，検定統計量にしてしまうので実感が湧きにくいところがあったかもしれません。データ生成モデルの場合は，直接パラメータを考えるのでイメージがしやすいという側面がありました。

さらに，MCMC による推定ですから事後分布からの代表値が，実現値として手元のオブジェクトに代入されています。これをつかった計算をすることで，たとえばパラメータの差を計算でき，その分布を考えることができるのでした。こうした計算は Stan の `generated quantities` ブロックを使うことで，MCMC の結果と同時に考えることもできるのでした。

さて今回も，この `generated quantities` ブロックをつかって，生成量からいろいろ考えてみましょう。

4.1 事後予測分布

前回，二群のデータ $X_{i,A}$ および $X_{i,B}$ に対して，次のようなデータ生成モデルを考えました。

$$X_{i,A} \sim N(\mu_A, \sigma), X_{i,B} \sim N(\mu_B, \sigma)$$

このパラメータ μ_A, μ_B, σ の推定値，$\hat{\mu_A}, \hat{\mu_B}, \hat{\sigma}$ の分布からの代表値が MCMC によって得られるのでしたね。MCMC サンプルは iteration ごとに 1,2,3...M 個得られたとすると，その期待値すなわち **EAP** は，次の式で求めていることと同じです[*1]。

$$\hat{\mu_{A}}_{eap} = \frac{1}{M}\sum \mu_A^i = \frac{1}{M}(\mu_A^1 + \mu_A^2 + \cdots + \mu_A^M)$$

$$\hat{\mu_{B}}_{eap} = \frac{1}{M}\sum \mu_B^i = \frac{1}{M}(\mu_B^1 + \mu_B^2 + \cdots + \mu_B^M)$$

$$\hat{\sigma}_{eap} = \frac{1}{M}\sum \sigma^i = \frac{1}{M}(\sigma^1 + \sigma^2 + \cdots + \sigma^M)$$

[*1] ここで上つきの数字は MCMC のステップ番号を表しているもので，冪乗の数字ではありません。また M は Stan のデフォルトでは 4000 です。

さて, 今回は得られたデータ $X_{i,.}$ から μ, σ を推定したわけですが, これはいわばデータ生成メカニズムの設定値を見つけたことと同じですね。例え話で考えるなら, こんな感じです。ある企業 A が製品 a を量産しているとします。ライバル会社 B が, この製品 a の類似品を作ろうと考えたとします。そこで B 社は A 社の製品 a を作っている製造機械と同じ型番の機械を購入します。この機械には製品を生成するに当たっていくつかの設定をしなければならないとします。アナログな例えで恐縮ですが, ツマミが 2 つ 3 つ付いていて, それをひねれば何らかの製品ができるといった感じです。でも A 社が製品 a をつくるのに, どういう設定にしているのかはわからない。そこで製品 a をいくつかサンプルとして入手します。入手したサンプル a1,a2... と見比べながら, この辺りかな？この辺りかな？とツマミを調節し, サンプル a1,a2... と同じような製品 b1,b2... ができるのはこの辺の設定だな, というのを見つけたという感じ。

この例え話を MCMC の用語を使っていうと, 入手したサンプル a1,a2... がデータです。このデータをつくる製造機械が**確率分布**です。機械には設定のツマミがついているんですが, そのツマミは**パラメータ**のメタファーになっています。サンプルを参考に, この辺りかな, この辺りかな, とツマミの値を探るステップが MCMC の各段階で, まずは $\mu_A^1, \mu_B^1, \sigma^1$, 次にそれからちょっと動かして $\mu_A^2, \mu_B^2, \sigma^2$, またちょっと動かして･･･と 4000 回試すことで「だいたいこのデータを作るための設定はこの辺り」というのが決まってくるわけですね。「この辺り」というのを点推定する場合は **EAP** や **MAP** などを使いますし, 幅を持って推定したい場合は**確信区間**で表現するのでした。

ということで, いくつかの手元のデータから, 製造メカニズムの設定を手に入れました。このとき B 社はきっと, 「じゃあこの推定値をつかって (模造品である) 製品をジャンジャン作っていこう」となるでしょう。この推定値から作られる新しい製品 (模造品, 新しいデータとも言えます) のことを MCMC ではとくに**事後予測分布 (posterior predictive distribution)** と言います。分布となっているのは, 作られる製品も散らばりますので, その散らばり方を分布で考えるからです。

事後予測分布は, 点推定値を使って計算することもできます。つまり, データが $X_i \sim N(\mu, \sigma)$ というメカニズムで作られていて, それから推定値 $\hat{\mu}, \hat{\sigma}$ を得たわけですから, これを使って $X_{new} \sim N(\hat{\mu}, \hat{\sigma})$ とすれば良いのです。これはただの乱数生成でもありますから, たとえば R で `rnorm(mu,sigma)` とするときに `mu` と `sigma` に推定値を入れてやればいいでしょう。でも点推定値では決め打ちが過ぎますので, 区間推定した上限と下限も入れますか？いや, それならいっそ, MCMC で得られた事後分布からの代表値を全部入れてしまえばいいのではないでしょうか。

それを実現するために,`generated quantities` ブロックを使うといいでしょう。コード 4.1 にその例を書いてみました。データやモデルは前回の二群の平均値差の検定と同じものです。

code : 4.1　事後予測分布を作るコード

```
1  data{
2      int<lower=0> N1; // Number of Subjects in Group 1
3      int<lower=0> N2; // Number of Subjects in Group 2
4      array[N1] X1; // Data in Group 1
5      array[N2] X2; // Data in Group 2
6  }
7
8  parameters{
9      real mu1;
10     real mu2;
11     real<lower=0> sig1;
12     real<lower=0> sig2;
13 }
14
```

```
15  model{
16      // likelihood
17      X1 ~ normal(mu1,sig1);
18      X2 ~ normal(mu2,sig2);
19      // prior
20      mu1 ~ uniform(0,100);
21      mu2 ~ uniform(0,100);
22      sig1 ~ cauchy(0,5);
23      sig2 ~ cauchy(0,5);
24  }
25
26  generated quantities{
27      real Xpred1[N1];
28      real Xpred2[N2];
29      for(i in 1:N1){
30          Xpred1[i] = normal_rng(mu1,sig1);
31      }
32      for(i in 1:N2){
33          Xpred2[i] = normal_rng(mu2,sig2);
34      }
35  }
```

ここで，generated quantities には，normal_rng という関数が入っています。この_rng は確率分布の後ろにつけて，乱数を発生させろという意味です。R でいう r****みたいなもんですね。このコードでは各群と同じサイズだけの事後予測データセットを作っています。まあ同じ μ と σ からの乱数ですから，データの数だけ作ったりしなくてもいいんですがイメージしやすくするためにそうしてみました。この μ と Xpred を図にしてみたのが図 4.1 です。この図の左側は μ_1 の**事後分布**が，右側には X_1 の**事後予測分布**が描かれ

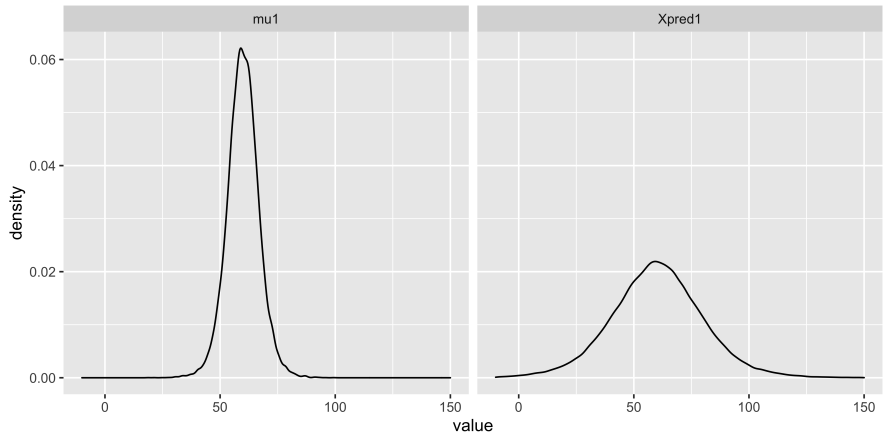

図 4.1　事後分布と事後予測分布

ています。左側は**パラメータの世界**の話であり，右側は**データの世界**の話になっていることに注意してください。

パラメータ μ_1 は，EAP で 59.96,95%CI[48.81,71.22] となっています。そこからでてくるであろうデータは，25 から 100 弱の幅に分布していますね。平均が 60 であっても，SD が 20 弱ありますから，データのレベルではその散らばる幅が大きくなるのも当然ですよね。でもこれはとても大事なことで，我々は「群間に差があった」となるとその効果について色々考えますが，それはパラメータの話，あるいは**平均因果効果**の話で

あって，個々のデータではまた事情が違うのです。たとえば，男女差があると言っても平均的な男性，そうでない女性などさまざまなケースがあるように。たとえば，平均的に 80% 成功する手術があると言われても，自分の身に置き換えると生きるか死ぬかは割合の問題ではないように。

また，この事後予測分布の形は，もとのデータの分布の形と似ているかどうかを視覚的に確認するために使われます。図 4.2 に，事後予測分布ともとのデータのヒストグラムを合わせて表示してみました。事後予測分布は MCMC サンプルの数だけありますが，ここでは一部だけ取り出しています。これを見ることで，もと

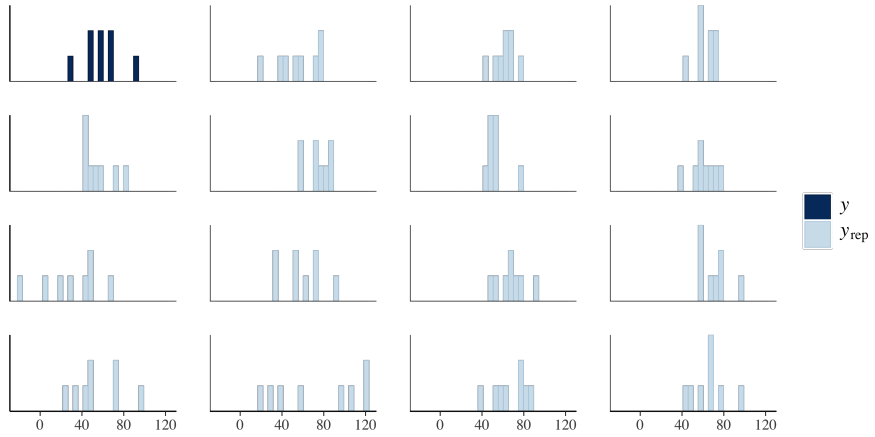

図 4.2　データの分布と事後予測分布の視覚的な確認

のデータと事後予測分布の形が似ているかどうかがわかります。もし模倣した機械の設定値が正しいのであれば，手元にある商品サンプルと同じような分布をするデータ分布が得られるはずですね。視覚的な確認ですから，見て「うーん似てるな，似てないな」というのを見て考えるだけではあります。今回はデータの数が 8 件と少ないので何ともイメージしにくいところですが，データが多くなるとデータの分布の形状で比較することもしやすくなるでしょう。もし分布の形が似ていなければ，購入した製造機械が違うものだったのかもしれない，ということになります。というのも実際のデータ分析の場合は，製造機械 (確率分布，モデル) が正しいかどうかはわからないので，それすらも仮定，検証すべき対象だからです。

4.2　データレベルの仮説

事後予測分布の利用方法は他にもあります。事後予測分布は，確率モデルと推定されたパラメータを利用した「新しいデータの生成」だったわけですが，この新しく生まれるであろうデータについての，データの世界での仮説を考えることができるのです。

帰無仮説検定で扱っていたのは，パラメータの世界の仮説でした。たとえば二群の平均値差の検定について言えば，帰無仮説は $\mu_1 = \mu_2$ であり，対立仮説は $\mu_1 \neq \mu_2$ です。これはギリシア文字 μ を使って書いてあることから明らかな通り，母数の仮説であり，$\bar{X}_1 = \bar{X}_2$ のような，データについての仮説ではありませんね。ではデータについての仮説というのはどういうものがあり得るでしょうか？たとえば次のようなものが考えられます。

- 無作為に選んだ統制群のデータと実験群のデータを比べたとき，実験群が統制群を上回っている確率はどれぐらいあるだろうか？
- 無作為に選んだ統制群のデータと実験群のデータの差が，基準点 c より大きくなる可能性はどれぐらいあるだろうか？

第 1 の点は**優越率 (probability of dominance)** という指標です。これはパラメータの世界の比較ではなく，実データの比較ですからイメージしやすいのではないでしょうか。たとえば男女差が平均的にはあるとされていたとしても，それはパラメータの違いであって，実際これから新しく出会う男性 (女性) が自分より優れている (劣っている) 可能性は，と考えた方が実質的な意味がありそうです。性差の話の多くは，平均値差だけでしか議論されませんが，効果量[*2]で考えると小さいことが少なくありません。それでも有意差が検出されてしまうのは，非常に大きなサンプルサイズをとっているから，ということもあったりします。サンプルサイズを大きくとるのは研究者の工夫や努力なのですが，小さな効果を取り上げて針小棒大に騒ぎ立てるのは科学的に良い態度とは言えません。そんな時に優越率で考えてみると，差があると言っても無作為に二群から新しい情報を得た場合の優越率が 50% ちょっとしかない，つまり超えるか超えないかが半々ぐらいですから「何の情報ももたらさない」のと同じこと，ということになったりします。くどいようですが，実データのレベルでの比較になるので，群間の差の表現としてよりわかりやすいということが言えるでしょう。第 2 の点は，優越率に一定の違いの差 c を含めて考えるもので，**閾上率 (probability beyond threshold)** といいます。これもデータレベルでの仮説の 1 つで，たとえば新しいトレーニング方法を使えばタイムが 3 秒小さくなる確率は X%，などということが分かれば具体的にイメージしやすい伝達方法になると思いませんか。

ちなみにこの優越率，閾上率などの指標は古くから指摘されてきていたものであり[*3]，ベイズ推定を使わない，**モーメント法**による推定でも計算可能な量でした。まぁしかし，このような統計量が実際に使われているケースってほぼ見かけないんですけどね。しかし，これらの指標はベイズ推定を使うことによって，というより乱数による近似計算をすることによって，遥かにイメージしやすくなっています。ここで示された数値はいずれも，`generated quantities` ブロックの中に `if` 文をつかって書いてやれば表現できることだからです！

実際に考えてみましょう。これら「条件が成立する確率」については，成立すれば 1，成立しなければ 0 というフラグをたてて MCMC サンプルを生成し，その平均値すなわち相対頻度で考えてやれば良いのでした。

code : 4.2 優越率や閾上率を計算するコード

```
data{
    int<lower=0> N1; // Number of Subjects in Group 1
    int<lower=0> N2; // Number of Subjects in Group 2
    array[N1] real X1; // Data in Group 1
    array[N2] real X2; // Data in Group 2
    int<lower=0> C; //constant
}

...

generated quantities{
    real Xpred1;
    real Xpred2;
    int<lower=0,upper=1> FLG1;
    int<lower=0,upper=1> FLG2;
    Xpred1 = normal_rng(mu1,sig1);
    Xpred2 = normal_rng(mu2,sig2);
    //probability of dominance
    if(Xpred1 > Xpred2){
        FLG1 = 1;
    }else{
```

[*2] 標準化された平均値差のことでしたね。

[*3] たとえば南風原・芝 (1987) による優越率の解説は 1987 年の論文です。

```
22          FLG1 = 0;
23      }
24      //probability beyond threshold
25      if(Xpred1 - Xpred2 > C ){
26          FLG2 = 1;
27      }else{
28          FLG2 = 0;
29      }
30  }
```

コード 4.2 は，generated quantities ブロックで 2 つの FLG 変数を用意しています。FLG1 が優越率，FLG2 が閾上率のために用意したものです。このブロックで，推定されたパラメータ mu1,mu2,sig1,sig2 からそれぞれ生成される二群の「新しいデータ」を作り (Xpred1,Xpred2)，成立するかどうかを検証します。FLG1 は if 文の中にあるように，Xpred1 > Xpred2 が成立すれば 1，しなければ 0 になる数字です。FLG2 は，同じくその差分が一定の値 C を超えていれば成立，超えていなければ不成立というフラグです。ちなみにこの値 C は data ブロックで外部から読み込むことにしてあります。

これと前回のデータを使って推定させてみましょう。

code : 4.3　優越率，閾上率を推定するコード

```
1  groupA <- c(30, 50, 70, 90, 60, 50, 70, 60)
2  groupB <- c(20, 40, 60, 40, 40, 50, 40, 30)
3  dataSet <- list(X1 = groupA, X2 = groupB, N1 = 8, N2 = 8, C = 3)
4  ### rstanの場合
5  sampling1 <- sampling(modelR, dataSet, warmup = 1000, iter = 4000, chains = 4)
6
7  ## cmdstanrの場合
8  sampling2 <- modelC$sample(
9    data = dataSet,
10   chains = 4,
11   iter_sampling = 5000,
12   iter_warmup = 1000,
13   parallel_chains = 4
14 )
```

MCMC の結果 3

```
# A tibble: 8 × 7
  name         EAP       MED       MAP        SD       L95       U95
  <chr>  <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!>
1 FLG1       0.800     1.000     0.999     0.400     0.000     1.000
2 FLG2       0.764     1.000     1.001     0.425     0.000     1.000
3 mu1       59.933    59.936    59.833     6.914    46.147    73.895
4 mu2       40.006    40.002    40.141     4.656    30.869    49.362
5 sig1      18.598    17.495    15.846     5.480    11.310    32.054
6 sig2      12.640    11.941    11.071     3.725     7.623    21.903
7 Xpred1    59.880    59.828    58.098    20.590    18.660   100.984
8 Xpred2    39.887    39.978    40.457    13.984    11.642    67.691
```

ここにあるように，優越率は 80.0%，閾上率は 76.4% ということになりました。実験群 (GroupA) のほうが

統制群 (GroupB) よりもスコアが高くなる確率が 80%, スコアが 3 点以上大きくなる確率は 76% もあるのですから, 実験群の効果は結構おおきいぞ, といったことがわかるようになります。

こうしたデータに基づく仮説は, 単位がもとのデータに対応しているから想像しやすく, より実践的な意味を想像しやすいですね。

まとめておきます。パラメータの世界に関する仮説は, 推定されたパラメータはどれぐらいの値なのか, どれぐらいの「差」「違い」があるのかを考えることでした。あるいは, 推定されたパラメータはどれぐらいの幅に分布するのか, という差の分布を考えることで, 効果の有無の確からしいさを量的に評価できるのでした。データの世界に関する仮説は, データ生成機がどういうデータを作るのか, ひいてはそれが今のデータと同じ形をしているかを見ることで, モデルが正しいかどうかを検証できます。また未来のデータの分布はどんな形になるのか, という意味では, データという具体的な数字でもって未来予測をできます。

ただしいずれの世界においても注意しておいて欲しいのは, データ生成機すなわち確率モデルが正しい場合であり, これが間違っていると `generated quantities` などで作られる数字もまったく意味のないものになってしまう, ということです。もちろん **t 検定**など帰無仮説検定を行う時も, データが正規分布していなければ意味のない検定結果になるので, 「そもそも統計的な仮定が間違ってた」というのでは意味がないのはいずれも同じです。推測統計学は手元の少しのデータから, 未知の世界を予測検証するという不良設定問題に対応するためのものですから, 仮定や前提が正しいかどうかについては常に敏感でなければなりません。

4.3 パラメータ・リカバリ

そもそもの仮定が間違っていた, というのを考えすぎて何もできなくなっても良くないですから, そこはいったん OK だとしましょう。そう考えると, 仮定・前提をつけた上での話になりますが, 未知の世界や未来を予測できるというのは, とんでもなく強力なツールであるような気がします。株価や競馬の予想ができれば, 大儲けできる薔薇色の人生が待っているかも。ということで, ベイズ推定の勉強にも身が入るというものです。推定を可能にしてくれた MCMC に感謝！

半ば冗談, 半ば本気なのですが, Stan や JAGS などの**確率的プログラミング言語 (stocastic programming language)** が登場したおかげで, 事後分布の計算が可能に, 簡単便利にできるようになったというのは大変ありがたいことです。これのおかげでどんなモデルでも, 設計図が書ければ答えが出せちゃう。複雑な確率の式とか微分積分を考えなくても, なんとかなるような気がします。しかし簡単なツールが出てきたときの常で, ちょっと注意が必要なのですが, MCMC は優秀すぎて正しくない計算式からでも答えが出てしまう, という可能性があるのです。

MCMC がうまく行ったかどうかは, **Rhat** や**有効サンプルサイズ**, **トレースプロット**などをチェックすればよい, というのはすでにお話しした通りです。しかしここで指摘したいのはそうではなく, これらの MCMC がうまく行っていたとしても, 推定結果が正しいとは限らない可能性です。MCMC は「あり得そうな答えの可能性」を大量に出力し, その分布で我々は考えるのですが, その答えの可能性群が全部的外れなところに行ってるかもしれないのです。実際にデータを分析していると感じるのですが, MCMC はかなり複雑なモデルを考えても答えを出してくれます。とにかく答えてくれる機械が手に入ったとしても, その信憑性をチェックすることを忘れてはいけません。

ではこれをチェックするためにはどうすればよいでしょうか。ひとつは, 答えが先にわかっている問題を作って, 機械が正解を当てられるかどうかを検証するということが考えられます。この「わかっている正しい値」を当てることができるのであれば, 実際のデータを用いた「わからない値」を推測する方法として有用だということがわかります。正解を当てることができないようであれば, この計算機械は当てにならないので, 実際のデータを使っても正しく推測できているとは言えないでしょう。

この検証方法のことを**パラメータリカバリ (parameter recovery)** と言います。実際のデータ分析をする場合は，いきなり未知の答えに推測機をあてがうのではなく，推測機が事前に設定したパラメータをリカバリ (回復) できることを確認してから利用するようにしましょう。実際の研究実践の場合，その分析手順は，以下のようになります。

1. データ生成メカニズムを考える (紙とペンによる設計図の作成)
2. モデルを式で表現する
3. モデルの式からデータ生成をシミュレーション
4. Stan でコードを書く
5. シミュレーションデータを Stan が再現していることを確認する
6. 実際のデータでやってみる
7. 結果の解釈(図による確認も)

ここで第 3 のステップ，仮想データの生成から第 5 のステップ再現の確認が，パラメータリカバリという作業になります。推定したい平均値などのパラメータを事前に仮に入れてみて，その答えが復元できるかどうかを見るのですね。

なんだかまだるっこしい！と思う人もいるかもしれません。自分で設定したパラメタ通りになるのって当たり前じゃないか，何がおもしろいのか？というわけですね。ですが意外と当たり前でないことがあるんです。間違えた答えであれば，実践では役に立ちません。あるいは，いきなりデータでいいじゃない，固いこと言うなよ，と思う人もいるかもしれません。これが設定ミスの場合は答えを出さない・答えられないという機械であればそれでもいいのかもしれませんが，MCMC は大概なんでも答えてしまうので，いきなりの運用は怖いと思いませんか。またあるいは，基礎実験とかそのほかの授業でも，今までそんなのやったことないよ，という意見もあるかもしれません。大学の授業で行うような演習・実習は，そもそも効果がはっきりあること，再現する現象であることがある程度確からしいものを選んでやっていますからいいのですが，これから皆さんは研究者の卵として，新しい現象，十分に確認されていない現象を研究対象にすることになります。そのとき，いきなり出てきた結果がどれほど信用できるでしょうか。実際，最近では心理学実験が再現できないケースも数多く指摘されており，その理由のひとつとして統計モデルの乱用 (悪用・誤用) にあったことは何度もお伝えしてきた通りです。さらに，ごく微小な効果を偶然検出してしまった，という可能性もあります。事前に推定機の精度がわかっていればこうした問題を回避できるのです。この話はベイズ推定をする場合だけでなく，帰無仮説検定など従来のやり方にも通じ，**効果量**や**例数設計**の話に直結します。

4.3.1　二群の平均値差の例

ここまで扱ってきた，二群の平均値差を検証する例で考えてみましょう。もともとの設計図 (図 3.1) がここでも役に立ちます。データがどういうメカニズムで出てきたかを考えていたわけですから，これを使えばデータ生成を考えることができますね。乱数生成のアプローチはデータを取り始める前にも有用なのです！

今回はパラメータを仮に設定してやることができます。二群の平均とその差を次のように決めてしまいましょう。

code : 4.4　平均値差の設定

```
1  mu1 <- 50
2  diff <- 10
3  mu2 <- mu1 + diff
4  sig1 <- 5
5  sig2 <- 8
```

第一の群の平均は mu1 <- 50 としています。第二群の平均は mu2 <- 60 というようにしてもいいのですが，差を考えるという意味を強調するためにいったん変数 diff を作って差の量を定め，それを mu1 に加えるという方法をとっています。2 つの群は正規分布に従いますから，それぞれの標準偏差を sig1,sig2 として設定しました。ここから仮想データを作ります。データの生成は乱数に従うので，rnorm 関数の出番です。

code : 4.5 仮想データの生成

```
1 set.seed(12345)
2 N <- 10
3 X1 <- rnorm(N, mu1, sig1)
4 X2 <- rnorm(N, mu2, sig2)
```

これで実際に生成された仮想データは次のようになっています。

R の出力 4.1: 検定の結果

```
> X1
[1] 52.92764 53.54733 49.45348 47.73251 53.02944 40.91022 53.15049
[8] 48.61908 48.57920 45.40339
> X2
[1] 59.07002 74.53850 62.96502 64.16173 53.99574 66.53520 52.90914
[8] 57.34738 68.96570 62.38979
```

ちなみにこの 2 つのデータセット，平均は $\bar{X}_1 = 49.33528, \bar{X}_2 = 62.28782$ となっています。理論的にはそれぞれ $50, 60$ であるはずなのですが，乱数による実現値なので少し誤差が出ていますね。これは実際の研究状況でもある話です。理論的な値と，目の前の実現値というのは必ず少しずれがあるもの。その中で，きちんと理論値を推定できるかどうかが問題なのですから。さて，このデータを使って分析していきましょう。

code : 4.6 パラメータリカバリ・検証

```
1  ## t検定(モーメント法による推定と判定)
2  t.test(X1,X2)
3  ## ベイズ推定(MCMCによるベイズ推定と差の分布)
4  dataSet <- list(X1 = X1, X2 = X2, N1 = N, N2 = N, C = 3)
5  modelC <- cmdstanr::cmdstan_model("ttest03.stan")
6  sampling2 <- modelC$sample(
7    data = dataSet,
8    chains = 4,
9    iter_sampling = 5000,
10   iter_warmup = 1000,
11   parallel_chains = 4
12 )
```

まずは t 検定の結果ですが，ここでは $t(14.797) = 5.2059, p = 0.0001113$ で有意差ありと判定されました。実際 diff <- 10 で差がある設定なのですから，正しく判定できて良かったな，というところです。ベイズ推定の方はどうでしょうか。結果は次のようになりました (ここでは前回のコード 3.6 を再利用しました)。

MCMC の結果 4

```
# A tibble: 6 × 7
  name        EAP       MED       MAP        SD       L95       U95
  <chr> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!>
1 diff    -12.936   -12.957   -13.250     2.762   -18.458    -7.445
2 FLG       0.000     0.000    -0.001     0.000     0.000     0.000
3 mu1      49.349    49.350    49.499     1.477    46.381    52.302
4 mu2      62.285    62.285    62.256     2.355    57.600    66.977
5 sig1      4.482     4.257     3.868     1.194     2.842     7.406
6 sig2      7.189     6.870     6.363     1.821     4.615    11.579
```

これを見ると，mu1 <- 50 と設定したときの EAP 推定値が 49.349，95%CI で [46.381, 52.302] ですから，ほぼ正しい点推定値，また確信区間の中に真値を含んでいます。推定はうまく行っている，ということです。mu2 についても同様で，真値が 60 であり，EAP 推定値が 62.285，95%CI で [57.600, 66.977] です。点推定値は真値から 2 点ほどずれていますが，95% の区間推定にすれば真値を含んでおり，推定はうまく行っているようです。標準偏差の推定値についても，それぞれうまく行っており，差についても EAP 推定値が-12.936，95%CI で [-18.458, -7.445] です[*4]。確かに確信区間に真値は含んでいるのですが，17.45 は真値 10 から少し外れすぎかな... というようなことがわかりますね。

このように，今回の結果をみると概ね正しく推定できていると言えるでしょう。その精度についても「どの程度誤差が生じうるものなのか」が具体的にわかったことと思います。しかし次のような設定の場合はどうでしょう。

code：4.7 仮想データの生成その 2

```
1  mu1 <- 50
2  diff <- 18
3  mu2 <- mu1 + diff
4  sig1 <- 10
5  sig2 <- 15
6  N <- 3
```

この設定でデータを作って，推定した結果は次のようなものです (出力 5)。

MCMC の結果 5

```
# A tibble: 6 × 7
  name        EAP       MED       MAP        SD       L95       U95
  <chr> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!>
1 diff     -5.434    -5.630    -5.104    12.855   -31.789    21.966
2 FLG       0.171     0.000     0.000     0.376     0.000     1.000
3 mu1      53.962    53.972    54.117     4.147    45.756    62.294
4 mu2      59.396    59.671    59.608    12.124    32.811    84.278
5 sig1      6.194     5.034     3.778     4.332     2.356    17.443
6 sig2     20.651    17.148    13.108    12.966     8.682    53.683
```

MCMC の収束は問題なく，推定値は出力されていますが，本来 $\mu_1 = 50$ であるところが 53.962，95% 確信区間は [45.756 ,62.294] であり，$\mu_2 = 68$ であるべきところに至っては EAP 推定値が 59.396，95%

[*4] 生成量は diff = mu1 - mu2 で作っているので，正負が逆転しているのはご容赦ください。

確信区間は [32.811 ,84.278] です。確信区間はなんとか平均値を挟み込みましたが，38 から 84 までと 46 点も幅がある推定はあまり有効な予測とは言えませんね。差は 18 あるという設定なのですが，EAP 推定値はわずか 5.434 であり，95% 確信区間は [-31.789, 21.966] です。確信区間にゼロをふくんでいますから，「差がないかも」ということになります。実際，t 検定の結果は帰無仮説を棄却できませんでした ($t(2.2337) = 0.52828, p = 0.6451$)。**タイプ 2 エラー (Type II Error)** を犯してしまっています。

どうしてこんなにうまくいかなかったのでしょうか。今回は正規分布という正しい確率モデルを使っていたのに，です。その答えは明らかに，`N <- 3` という点にあります。つまり，データが 3 件しかないので予測の精度が悪くなったのです。このように，モデルが正しくても推定精度が十分ではない，あるいは判定をするときに間違った判断をする可能性がある，ということがわかります。推定精度や効果の判定については，サンプルサイズ，データの散らばり (群内分散)，データ間の差の大きさ (群間分散) などの要因が影響してきます。ここで設定値を色々変えてみて，どういう関係にあるのかをみておくのも良いでしょう。

このように仮想データを使ったパラメタリカバリを通じて，推定や検定の精度を設計したりチェックしたりできることがわかりました。これを踏まえて考えると，前回の例では $N = 8$ のデータで分析をしましたが，$N = 8$ だと平均値差を 8〜12 点ほど外した予測をしてしまうかもしれない，ということがわかります。逆に「5 点差までの推定誤差に収めたい」というような希望があった場合，データは何件ぐらい取れば良いでしょうか？このような問いについても，N の設定値をさまざまに変えてみることで，実験や調査を実際に始める前に予想を立てておくことができます。これが**例数設計**です。例数設計の重要さを確認するとともに，こうした乱数を作るアプローチで簡単に実践できますので，実際の研究の前にはぜひ一度試してみてください。

4.4 今回のまとめ

生成量をつかうことで，次のような検証をできるのでした。

- 事後予測分布を作ることで，想定した確率モデルが正しかったかどうか検証する。
- 事後予測分布を利用して，優越率や閾上率などデータに基づく仮説を検証する。

さらに，データ生成モデルを応用して，そもそも擬似データをつくることで推定モデルの精度や，どれぐらいのサンプルサイズが必要かといったことを，実践の前に検証できます。

データ生成アプローチと乱数発生技術の組み合わせで，より慎重かつ確実な研究アプローチができることを理解してください。

4.5 課題

次の計算をする R/Stan コードや回答を記述し，提出してください。なお提出されたコード単体でバグがなく動くことが確認できないものは，未提出扱いになります。コードの書き方などわからないところがあれば，曜日別 TA か小杉までメールで連絡し，指導を受けてください。

■フライドポテトの研究その 2 あるコンビニチェーンの 2 つのお店 A,B で，それぞれホットスナックのフレンチフライを買ったところ，どうも一方より他方の方が長くて大きい気がした。そこでそれぞれの袋から取り出して長さを測定してみた (単位 cm)。測定結果が表 4.1 である[*5]。このデータを用いて次の問に答えなさい。なおポテトの長さは正規分布に従い，また両群の分布の幅は同じであると仮定します。ちなみに，この数値はシラバスのサイト上のコードで取得可能です。

[*5] このデータは実際に筆者がコンビニの二つの店舗でポテトを購入し，長さを測定した結果に基づいています。

表 4.1 二つの店舗のポテトの長さ

A	8.4	11.3	8.1	11.2	5.8	6.3	7.1	10.9	7.1	6.5	5.0	3.0	7.2	6.5	6.4	6.4	9.3	8.3
B	6.7	7.2	4.2	11.0	7.5	8.9	7.0	8.0	7.2	4.2	6.0	9.0	8.6	9.0	5.0			

1. 店舗 A のポテトの平均が, 店舗 B のポテトの平均より長い確率はどれぐらいあるか。
2. 今後, 店舗 A で購入するポテトが店舗 B で購入するポテトよりも長い確率はどれぐらいあるか。
3. 今後, 店舗 A で購入するポテトが, 店舗 B で購入するポテトよりも 2cm 以上ながい確率が 30% あれば店舗 B では買わないようにしようと思う。店舗 B を利用すべきかどうか, 推定に基づいて判定してください。
4. 今後, 店舗 A で購入するポテトと店舗 B で購入するポテトの長さの違いが 5cm 未満である確率が 80% であれば, 家により近い店舗 B を利用してもいいように思う。店舗 B を利用すべきかどうか, 推定に基づいて判定してください。
5. このコンビニチェーンのポテトの長さ平均を, できるだけ正確に推定したいと思う。標準偏差は今回の MAP 推定値を使うこととして, 何本ぐらいポテトのサンプルを集めればその 95% 確信区間を 0.05cm 以内に収めることができるだろうか。シミュレーション結果に基づき, おおよその数字を答えてください。

第 5 章

モデリングの目から見た検定 3；多群の平均値差モデル

5.1　要因計画モデル

さてここまでは二水準の平均値差を考えるモデルでしたが，ここからそれを三水準に増やしてみましょう。水準数が多くなると**要因計画 (Factorial Design)** や**実験計画 (Experimental Design)** といった話になってきます。**分散分析 (ANalysis Of VAriance;ANOVA)** は要因計画と帰無仮説検定の合わせ技で，効果がある/ないの判定を分散の比をつかって行うことでした[*1]。

今回は要因計画をデータ生成モデルから考え，**ベイズ推定 (Bayesian Inference)** をすることになります。**帰無仮説検定**のような Yes/No 判断はせず，平均値や差の大きさを直接見積もることになるのは，二群の時と同じです。

まずは簡単な**群間計画 (Between Design)** の話から進めましょう。対応のない二群の時のように，それが三群になるモデルですので，モデルの設計図は非常に単純です (図 5.1)。設計図を見れば，コードはその

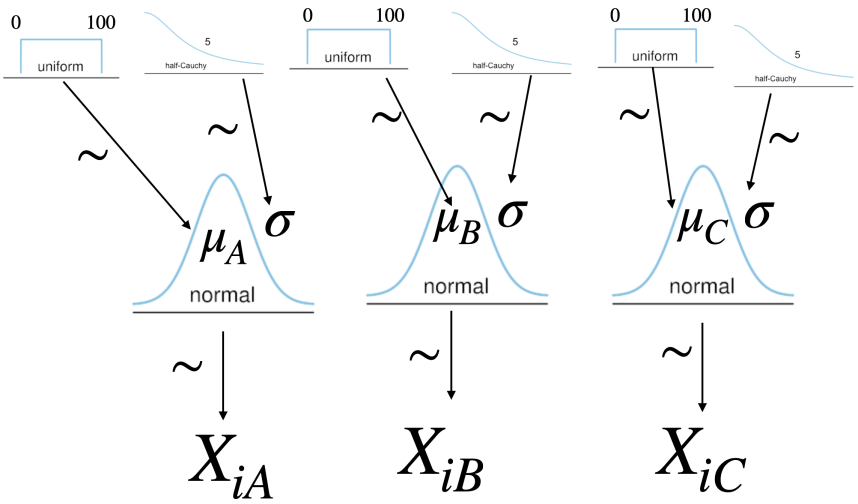

図 5.1　多群の平均値の差を比べるときの設計図

[*1] 忘れた人は，データ解析基礎のテキストに戻って確認してください。

まま置き換える形でかけますね。設計図の通りに書いてみたコードが code:5.1 にあります。

code：5.1　三群の平均値のコード

```
data{
  int<lower=0> N1;
  int<lower=0> N2;
  int<lower=0> N3;
  array[N1] real X1;
  array[N2] real X2;
  array[N3] real X3;
}

parameters{
  real mu1;
  real mu2;
  real mu3;
  real<lower=0> sig;
}

model{
  // likelihood
  X1 ~ normal(mu1,sig);
  X2 ~ normal(mu2,sig);
  X3 ~ normal(mu3,sig);
  // prior
  mu1 ~ uniform(0,100);
  mu2 ~ uniform(0,100);
  mu3 ~ uniform(0,100);
  sig ~ cauchy(0,5);
}

generated quantities{
  real diff12;
  real diff13;
  real diff23;

  diff12 = mu1 - mu2;
  diff13 = mu1 - mu3;
  diff23 = mu2 - mu3;
}
```

最後の generated quantities ブロックにあるように，各群の差の分布も計算して出せるようにしてあります。これをみながら，群 A と B，A と C，B と C などどこに差があるのか，その大きさはどれぐらいかということを考えることができます。左の大きさを標準偏差 sig で割ることで効果量を出すこともできます。

二群の平均値差の時もそうでしたが，検定と違って差の大きさを直接検証できているところがポイントです。また，検定の場合は分析によって有意差が確認できた場合，そのあと**多重比較 (multiple comparison)** へと進むのでした。これは帰無仮説が $H_0 : \mu_1 = \mu_2 = \mu_3$ となっていて，これを否定することから「どこに差があったか」をさらに詳しくみる必要があったからです。この時，統計的検定の繰り返しによって**タイプ 1 エラー (Type I Error)** が増えること，すなわち $\alpha = 0.05$ と 5% 水準で検定していても，検定を繰り返すと 5% を大きく上回る問題が生じるため，有意水準を調節するなど工夫が必要だという話で

した[*2]。

ところが，ベイズ推定の場合はこうした問題が生じません。なぜなら，確率を伴う判断をしていないからです。帰無仮説検定の場合は，「帰無仮説が正しいという条件のもとで」計算された検定統計量 (F 値や t 値) が，帰無仮説の条件下ではどれほど生じやすいかということを考え，その確率 p 値を計算していましたが，この p 値はパラメータがどこにあるかといった確率ではありません。判断のための指針としての確率に過ぎないのです。一方ベイズ推定する場合，推定されたパラメータの**事後分布**は，パラメータがこの辺りにあるだろう，という確率であり，生成量である差の分布もさの大きさがこれぐらいだろうという範囲，大きさに直接関わる確率です。ですから何 % 区間で考えてもいいですし，その判断が間違うかどうかの確率というのは問題に含まれないのです[*3]。検定の多重性の問題が生じない，というのは複雑な要因計画を考える場合には非常に助かりますね。

5.2 パラメータの変形と制約

5.2.1 パラメータの変形

ところで，先ほどは差の分布を generated quantities ブロックで表現しましたが，違う表現も可能です。差分も未知なるパラメータと考えて推定するのです。

順を追って説明しましょう。まず，最初の群の平均値を μ_1 とします。第二の群平均 μ_2 は，μ_1 と δ_1 だけ違う，つまり $\mu_2 = \mu_1 + \delta_1$ と考えるのです[*4]。同様に，第三の群平均 μ_3 は $\mu_3 = \mu_1 + \delta_2$ と考えます。こうすると，推定するべきパラメータは $\mu_1, \delta_1, \delta_2$ であり，generated quantities ブロックを使わなくても，直接差分パラメータを推定できるようになります。これを書いてみたのが，code:5.2 になります。

code : 5.2 差分を直接推定するコード

```
1  ...
2  parameters{
3    real mu1;
4    real delta1;
5    real delta2;
6    real<lower=0> sig;
7  }
8
9  model{
10   // likelihood
11   X1 ~ normal(mu1,sig);
12   X2 ~ normal(mu1+delta1,sig);
13   X3 ~ normal(mu1+delta2,sig);
14   // prior
15   mu1 ~ uniform(0,100);
16   delta1 ~ normal(0,100);
```

[*2] お前は何をいってるんだ，と思った人は，データ解析基礎のテキストに戻って確認しておいてください。大雑把に解説しますと，5% 水準というのは一回の判断で間違いが含まれる可能性を 5% にしましょう，という判定基準のことでしたが，二回判定を繰り返すと $1-(0.95^2) = 0.0975$ と 9.75% 水準になってしまうということでした。3 群の平均値を比べるときは，3 箇所比較する必要があり，**t 検定**を 3 回も繰り返すと 5% 水準ではなくなる，という問題です。

[*3] もっとも，すべて研究者の仮定したモデルのもとで，という話ではありますから，その仮定が間違っていたら全部意味のない数字である，ということにはなるかと思います。とはいえ，帰無仮説検定もデータが正規分布に従うという仮定で行うのですから，これについてはどっこいどっこい，というところでしょうか。

[*4] ここで δ はデルタと読む，ギリシア文字の d のことです。先ほどは生成量として diff と書いていましたが，ここでは未知のパラメータなのでギリシア文字に書き換えました。

```
17    delta2 ~ normal(0,100);
18    sig ~ cauchy(0,5);
19  }
```

data ブロックに違いはありませんので省略してあります。変更点として，まず parameters ブロックで推定するべきパラメータが変わっていることを確認してください。次に model ブロックで，3 つのデータがそれぞれ normal(mu1,sig),normal(mu1+delta1,sig),normal(mu1+delta2,sig) から出てきているように書き変わっています。正規分布の位置パラメータをずらすことで書き換えているのです。最後にその下，事前分布の設定ですが，差分 δ_1, δ_2 は正負どちらにも出てくる可能性があり，その大きさがわかりませんので，平均を 0 にした正規分布としてあります。これで推定した結果は，先ほどの生成量を使った結果と変わりはありません。数式の展開で書き方が変わっただけで，モデルの形は同じだからです。

ここで transformed parameters ブロックの紹介をしておきましょう。code:5.2 は悪くはないのですが，モデルのところ，normal(mu1+delta1,sigma) としているのがちょっと美しくないですね。元のコードにあった，normal(mu2,sigma) のほうがわかりやすいのに，と思った人もいると思います。まあ今の所，そこまで複雑ではないのでそれほど問題にはならないでしょうが，これからもしパラメータが複雑な構造をするようになったとき，尤度のところはもう少しスッキリ書きたいということも生じてくるでしょう。そういうときに transformed parameters ブロックを使います。このブロックは，parameters ブロックの次に書くもので，言葉の通りパラメータを変形 (トランスフォーム) するためのものです。

これを使って書いたのが code:5.3 です。

code : 5.3 パラメータ変換を入れたコード

```
 1  ...
 2  parameters{
 3    real mu1;
 4    real delta1;
 5    real delta2;
 6    real<lower=0> sig;
 7  }
 8
 9  transformed parameters{
10      real mu2;
11      real mu3;
12      mu2 = mu1 + delta1;
13      mu3 = mu1 + delta2;
14  }
15
16  model{
17    // likelihood
18    X1 ~ normal(mu1,sig);
19    X2 ~ normal(mu2,sig);
20    X3 ~ normal(mu3,sig);
21    // prior
22    mu1 ~ uniform(0,100);
23    delta1 ~ normal(0,100);
24    delta2 ~ normal(0,100);
25    sig ~ cauchy(0,5);
26  }
```

ここで注目すべきは，まず `transformed parameters` ブロックで `mu2,mu3` という変数名が宣言され，それが式によって表現されていることです。ここでパラメータ `mu1,delta1` が `mu2` に変形されていることがわかります。このようにして作られたパラメータであれば，`model` ブロックの中で使ってやることができます。現に，尤度のところが `X2 ~ normal(mu2,sig)` のようにスッキリした形になっていますね。ただし注意すべきは，事前分布のところが変わっていない，ということです。事前分布はパラメータに対しておかれますので，ここで `mu2 ~ uniform(0,100);` などと置くべきではありません。

このように変換することで，コードが多少なりともスッキリしたと思います。面倒だな，と思うかもしれませんが，コードがスッキリすることは自分にとっても誰にとっても「読みやすい」ことにつながります。そして読みやすいコードは，間違い (バグ) があったときに修正しやすいのです。あまりごちゃごちゃしたコードを書くと，問題が生じたときに対応が難しくなりますから，なるべくこうした作法を活用して，コードは単純明快なものにしておくと良いでしょう。

5.2.2 パラメータの制約

さてここでこのモデルをもう一段階，発展させてみましょう。今のコードは第一の群，μ_1 を基準に，μ_2, μ_3 を構成していました。別に第二，第三の群を基準にしても良かったですよね。つまり基準点は恣意的になっています。それで悪いということではないのですが，μ_1 だけ特別扱いです。そうではない作法として，全体平均 μ を考えて，次のように定式化してみましょう。

$$\mu_1 = \mu + \delta_1 \tag{5.1}$$
$$\mu_2 = \mu + \delta_2 \tag{5.2}$$
$$\mu_3 = \mu + \delta_3 \tag{5.3}$$

こうすることで，全体平均からの差分としてモデルを表現できます。ここで全体平均 μ はどういう意味があるでしょうか。群の違いを独立変数，結果のスコアを従属変数とした線形モデルの一環として考えると，全体平均は群を通じて変化のない水平線 (図 5.2) という関係を表していることになります。横一線で群ごとに違いがないのですから，いわばこれは**帰無仮説**のモデル，すなわち**帰無モデル (Null Model)** とでもいうべきものです。

あるいは，群ごとの平均値の違いはこの全体平均からの差分で表現されます。この群ごとの違いは，その群に入ったからこそ生じた**効果**であるとも言えます。つまり，δ は効果の大きさなのです。翻ってかんがえると，全体平均は「なんの処置もしなければ理論的にこの値になるはず」という操作なしの状態，天然の状態とでもいえるでしょう。**ランダム化比較実験 (Randomized Control Trial)** は，被験者をランダムに割り付けることによって，平均的に見れば個別の効果が相殺し合うことを用いています。平均化することで誤差が相殺するので群ごとの比較ができますし，群の平均値が変化したということは，その群に属したことの効果が現れたと考えるわけです。帰無モデルの状態は，この個別の効果を相殺させた状態であることを表しています。

さてこのように考えて，これを数式の形，モデルの形に書き起こしていくことを考えましょう。ところがこのままだと，未知のパラメータは $\mu, \delta_1, \delta_2, \delta_3$ と 4 つあることになります。今までは μ_1, μ_2, μ_3，あるいは $\mu_1, \delta_1, \delta_2$ の 3 つだったのに，です。同じ数式を書き直しただけなのに，パラメータの数が変わるのはおかしいですね。実はこれには 1 つ，制約が欠けているのです。今回は全体平均 μ からの相対比較になっていますから，$\delta_1 + \delta_2 + \delta_3 = 0$ という関係が隠れていることになります[*5]。そこで，この制約をモデルの中に書き込まな

[*5] もしこれが $= 0$ にならなければ，全体平均の計算に矛盾が生じますよね。全体平均 μ は，$\mu = \frac{1}{3}(\mu_1 + \mu_2 + \mu_3)$ ですから，

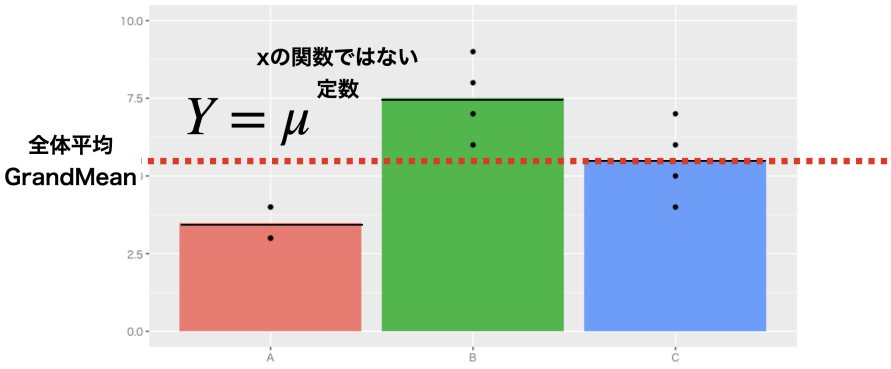

図 5.2　全体平均は水平線

ければなりません。$\delta_1 + \delta_2 + \delta_3 = 0$ という式から，変数を移項することでこれを表現します。すなわち $\delta_3 = 0 - (\delta_1 + \delta_2)$ のようにします。

これを使って書いたのが code:5.4 です。

code：5.4　制約を入れたコード

```
1  ...
2  parameters{
3    real gm;
4    real delta1;
5    real delta2;
6    real<lower=0> sig;
7  }
8
9  transformed parameters{
10     real delta3;
11     real mu1;
12     real mu2;
13     real mu3;
14     delta3 = 0 - (delta1 + delta2);
15     mu1 = gm + delta1;
16     mu2 = gm + delta2;
17     mu3 = gm + delta3;
18 }
19 ...
```

このようにすることで，先ほどと等価なモデルを別の形で書き表すことができるようになりました。

ここまでをまとめておきます。三群の平均値差を求めるモデルを，3 つの方法で書いてみました。

1. 3 つの群平均 μ_1, μ_2, μ_3 を個別に推定するモデル
2. ある群とそこからの差分，$\mu_1, \delta_1, \delta_2$ を推定するモデル
3. 全体平均とそこからの差分，$\mu, \delta_1, \delta_2, \delta_3$ を推定するモデル。ただし $\sum \delta = 0$ という制約をつける

$\mu = \frac{1}{3}(\mu + \delta_1 + \mu + \delta_2 + \mu + \delta_3) = \frac{1}{3}(3\mu + \sum \delta_j) = \frac{1}{3}3\mu + \frac{1}{3}\sum \delta_j$ となり，最初の項が $\frac{1}{3}3\mu = \mu$ ですから，第二の項は $\frac{1}{3}\sum \delta_j = 0$ でないと計算があいません！

この方法はいずれも同じ推定結果を返します。第一のモデルの生成量から差分を計算すれば，δ_1, δ_2 を計算したことと同じです。

同じことを異なる方法で実装するのは，生産的でないように思うかもしれません。しかし第 3 の形式にしておくと，モデルの一般化が容易になります。すなわち，多群の平均値差の比較をするときに，群の数が変わるたびにコードを書き換える手間がなくなるのです！

5.3 モデルの洗練

ここまで，二群，三群の平均値差を求めるモデルを書いてきました。書き方はわかっているので，比較する群が四群，五群と増えても書き足していくだけでなんとかなりそうです。

しかし，たとえば四群に増えたときに $\delta_1, \delta_2, \delta_3$, 五群に増えたときに $...\delta_3, \delta_4$ と手作業で増やしていると，N 群までふえたときは $...\delta_{N-2}, \delta_{N_1}$ と何行も書き足していかなければなりません。しかもその度にコンパイルしているようでは，とても面倒なことはすぐに想像がつきます。

そこで，群の数が変わっても同じコードで対応できるように，一般化を目指してコードを改良しましょう。そのためには，何群の比較なのかを示す群の数を，data ブロックで外部から取り込むようにすれば良いでしょう。その変数を使って差分の数，効果の数を計算してやれば良いのです。たとえば比較する群の数を G とすると，全体平均 μ と，$G-1$ 個の差分をパラメータとして推定すれば良いことになります。

その考え方を使って書いたコードは，たとえば code:5.5 のようになります。

code : 5.5 群の数を一般化したコード

```
data{
  int<lower=0> Lv;              // 水準数
  int<lower=0> N;               // 各群のサンプルサイズ
  array[Lv,N] real X;           // 変数の値
}

parameters{
  real gm;                      // 全体平均
  array[Lv-1] real raw_delta;   // 全体からの差。水準数マイナス1個
  real<lower=0> sig;            // 誤差の分散
}

transformed parameters{
  array[Lv] real delta;           // 差の大きさを作り直す
  array[Lv] real mu;              // 再構成される群ごとの平均

  for(i in 1:(Lv-1)){
    delta[i] = raw_delta[i];    // ほとんどコピー
  }

  delta[Lv] = 0 - sum(raw_delta); // 総和が0になるように最後だけ書き換える

  for(i in 1:Lv){
    mu[i] = gm + delta[i];        // 群ごとに再構成
  }

}

```

```
29  model{
30    // Likelihood
31    for(l in 1:Lv){
32        for(i in 1:N){
33            X[l,i] ~ normal(mu[l],sig);
34
35        }
36    }
37
38    // Prior
39    gm ~ uniform(0,100);
40    raw_delta ~ normal(0,100);
41    sig ~ cauchy(0,5);
42  }
```

■コード解説

data ブロック　群の数 Lv, 各群のサンプルサイズ N, 各データ点 X を入力します。ここで X が**二元配列**になっていることに注目してください。X[1,1] は第一群の第 1 番目のデータ，X[1,2] は第一群の第 2 番目のデータ，X[2,1] は第二群の第 1 番目のデータ，同様に X[i,j] は第 i 群の第 j 番目のデータを表すことになります。この形式は二元配列 (2 次元の変数セット) になっていると言います[*6]。宣言の時に，上で取り込んだ変数 Lv や N を使うことができます。最大 Lv 個の群，各群 N 人のデータがあるという前提です。

parameters ブロック　全体平均 gm と，差分，誤差の SD をパラメータとしています。差分はここでは raw_delta とし，上で宣言した Lv をつかって Lv-1 個の要素を持つ配列にしています。raw_delta って変な名前，と思うかもしれませんが，この後これを変換して δ を作りますので，作る前の生 (raw) の δ という名前にしてみました[*7]。

transformed parameters ブロック　以後の話をわかりやすくするために，各群の平均 mu を構成することにします。mu の数は水準数と同じ Lv です。そして第 i 群の平均 mu[i] を作るために，mu[i]=gm+delta[i] という書き方をしたいので，水準数と同じ数だけ delta[i] を作りたいと思います。もとの raw_delta が 1,2,...,Lv-1 個ありますので，delta も Lv-1 番目まではそれと同じものをコピーします (for 文で繰り返し代入していってます。)。そして最後に Lv 番目の要素を，0 からこれまでの要素をすべて足した (sum(raw_delta)) ものを引くことで作っています。ちなみに sum は stan の持っている総和の関数で，配列要素のすべてを足し合わせるというものです。こうして delta が Lv 個できあがりましたので，各水準の平均値を mu[i]=gm+delta[i] という数式で一般的に書くことができるようになりました。

model ブロック　尤度のところに，群それぞれについて巡回する for 文，その中で 1,2,3...N 人まで巡回する for 文を組み込んで二重にぐるぐると回しています。事前分布は設定の通りです。

このコードを使って，実際に推定してみましょう。データは表 5.1 のものを使います。

推定してみましょう (R コードは 5.6, 結果は 6)。各群の平均値，全体平均からの偏差などが，事後分布と

[*6] 行列でもいいじゃないか，と思われるかもしれません。それでも結構です。Stan は行列の時，real ではなく matrix で宣言することができます。しかしこのように real 型にしておいて，後ろのカッコで配列の次元を表現すると，何次元でも拡張することができるので今回はそのようにしました。

[*7] 変数名は任意ですので，著者の命名法がわかりにくいなあと思ったら，自分で好きな変数名にしてもらって構いませんよ。その場合は，以後すべての変数を読み替えてくださいね。

表 5.1 独立した三群から得られたスコア

groupA	6	6	5	5
groupB	7	4	7	6
groupC	4	3	4	6

して出力されているのがわかると思います。これをみて，たとえば差の事後分布の 50% 確信区間でみると δ_2, δ_3 はその区間に 0 を挟んでないので，一定の効果はあるだろうなどと判断できるわけです。もちろん生成量をつかって，ある程度の差がある確率がどのくらいとか，優越率などデータレベルの仮説を立てることも可能です。

code : 5.6 三群の平均値の比較

```
Example <- matrix(c(6, 6, 5, 5, 7, 4, 7, 6, 4, 3, 4, 6),
                  ncol = 4, byrow = T)
dataSet <- list(Lv = 3, N = 4, X = Example)
```

MCMC の結果 6

```
# A tibble: 10 × 7
   name              EAP       MED       MAP        SD       L95       U95
   <chr>         <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!>
 1 delta[1]          0.257     0.256     0.227     0.569    -0.880     1.394
 2 delta[2]          0.749     0.754     0.804     0.566    -0.387     1.878
 3 delta[3]         -1.005    -1.009    -1.114     0.576    -2.158     0.162
 4 gm                5.251     5.251     5.242     0.400     4.452     6.039
 5 mu[1]             5.507     5.507     5.562     0.699     4.114     6.902
 6 mu[2]             5.999     6.001     5.986     0.697     4.603     7.386
 7 mu[3]             4.246     4.238     4.182     0.695     2.860     5.647
 8 raw_delta[1]      0.257     0.256     0.227     0.569    -0.880     1.394
 9 raw_delta[2]      0.749     0.754     0.804     0.566    -0.387     1.878
10 sig               1.339     1.261     1.134     0.391     0.823     2.301
```

5.3.1 データサイズの一般化

さてこのモデルは，比較する群の数が増えても外部からデータとして群の数を与えることができますので，毎回コンパイルする必要がありません。やったね！これで完璧，と言いたいところですが，1 つ気になるのはサンプルサイズです。各群のサンプルサイズ N をデータとして与えるようになっていますが，たとえば A 群は 10 人，B 群は 12 人，C 群は 14 人と言ったように，群ごとにサンプルサイズが異なるとこのコードでは対応できません。実際に実験をやる場合，各群のサイズを整えて人を集めたいところですが，調査研究などでは群ごとに同じ人数を与えるような調整ができないこともあり，そういう意味ではこのコードではうまく対応できそうにありません。

そこで，群の人数が異なる場合でも対応できるように，さらにこのコードを拡張していきたいと思います。そのためにまず，データを**整然データ (tidy data)** の形に整形しましょう。

先ほどのサンプルデータ (表 5.1) は，3×4 の長方形をしていました。これはデータのサイズが整っているからできることで，サイズが変わってしまうと表の中に欠損ができてしまうことになります。また，たとえば B 群

の 3 番目の人の値を見る時，われわれは 2 行 3 列目のデータにさっと目が行きますが，機械的には群の情報が行名に，人の群内整理番号が列名にあるので，1 つのデータを見る時に行・列それぞれを参照することになります (図 5.3 左)。ここで「どの群か」「群の中の何番目の人か」という情報はいずれも変数で，人によって変わるものですから，データの持つ情報の 1 つなのです。データの持つ情報，変数なのに参照先が変わっていることになります。

このデータの情報をまったく損なうことなく，縦長に並べ直したのが図 5.3 の右側です。これは 1 つの行がそれぞれの観測に対応しています。このような形式にすると，何行目かということを指定するだけで，どの群の何番目の人なのかという情報が手に入ります。またデータの中に欠損があっても，整然データの形にする時にこれは観測に含まれないので，データの一部が欠けることがないという利点があります。

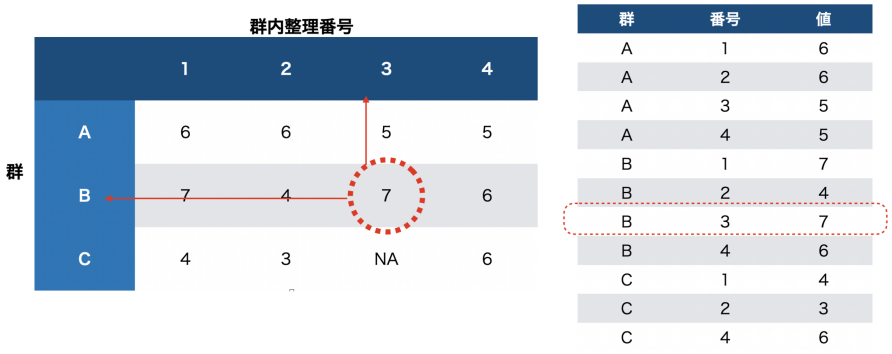

図 5.3 一般的なデータと整然データ

このようなデータ形式にして，これを与える形でデータ分析をするとデータサイズに依存しないコードが書けます。具体的には code:5.7 のようにします。

code : 5.7 整然データに対応したコード

```
data{
  int<lower=0> Lv;                          // 水準数
  int<lower=0> L;                           // データ数
  array[L] int<lower=0,upper=Lv> idx;       // データのID
  array[L] real X;                          // 変数の値
}
...(中略)...
model{
  // Likelihood
  for(l in 1:L){
    X[l] ~ normal(mu[idx[l]],sig);
  }

  // Prior
  gm ~ uniform(0,100);
  raw_effect ~ uniform(-100,100);
  sig ~ uniform(0,100000);
}
```

parameters ブロックと transformed parameters ブロックは code:5.5 と同じなので省略しました。まず data ブロックをみてください。群の数を Lv で入力するのはそのままですが，次にデータの総数 L を撮るようにしています。図 5.3 の例で言えば，C 群 3 列目のデータがありませんから，データ長は全部で $L = 11$ になります。次に L 行のデータがどの群に属するのかを指示するインデックス変数 idx を L 個用意しています。L 行目のデータが第何群に属しているのかの数字が入ります。たとえば 1 行目は A 群なので 1，5 行目 (L=5) は B 群なので 2, と言った数字が入るようになっています。さいごにデータの値そのものである X も，データ長 L と同じだけの 1 次元配列を用意してやります。

技術的に注目すべきは model ブロックの尤度のところで，for 文がデータ長 L の反復をしているだけです。つまり 1,2,3...,L 行目のデータを順に参照していくのです。そして l 行目のデータがどの群に属するかは，変数 idx[l] が指し示してくれますから，mu[idx[l]] としてやることで指定できていることになります。idx[l] は，l 行目のデータが属している群の数字が代入されていますから，たとえば 1 行目 (l=1) の場合は mu[idx[1]] = mu[1] としていることと同じ，5 行目の場合は mu[idx[5]]=mu[2] としていることと同じ，ということになります。ここでは入れ子になった参照が行われています。ちょっとテクニカルですが，この技術を使うと表現力も広がりますので，仕組みをしっかり理解しておいてください。

5.4 パラメータリカバリ

さあ，最後にパラメータリカバリをして，このコードが正しく推定できるか，あるいはどの程度のサンプルサイズがあればどの程度の精度で推定できるのかを検証しておきましょう。

私たちはデータがどのように造られるか，というそのメカニズムの方からアプローチしているわけです。これはリバースエンジニアリングと呼ばれる考え方ですね。そして今から仮想データを作ろうという時も，そのデータ生成モデルをそのまま利用すればいいのです。仮想データの生成は，データ生成メカニズムをリバースエンジニアリングで明らかにしていくアプローチのちょうど正反対，リバース・リバース・エンジニアリングです。

具体的には，たとえば次のようなコード (code:5.8) でデータを作ることができるでしょう。

code : 5.8 リバースエンジニアリング

```
N <- 100
Lv <- 5
gm <- 50
sig <- 3

raw_effect <- runif(Lv - 1, -10, 10)
effect <- c(raw_effect, 0 - sum(raw_effect))
mu <- gm + effect

X <- rnorm(N * Lv, mu, sig)
dat <- data.frame(
  Idx = rep(1:Lv, N * Lv),
  value = X
)

modelC <- cmdstanr::cmdstan_model("BetweenAnova2.stan")
dataSet <- list(Lv = Lv, L = NROW(dat), idx = dat$Idx, X = dat$value)
```

■コード解説

1 行目 各群のサンプルサイズ N を設定します。各群共通のサイズになります。

2 行目 水準数です。ここでは 5 群の平均値の比較をすることにしました。

3 行目 全体平均の設定です。事前分布に [0,100] の一様分布を置いてますから，真ん中ぐらいにしてあげました。

4 行目 誤差の散らばりを設定します。

6 行目 水準数 -1 の効果を設定します。手入力でもいいのですが面倒なので，-10 から 10 までの範囲で一様乱数によって生成しました[*8]。

7 行目 水準数 Lv の効果にするため，先ほど作った水準数 -1 の `raw_effect` に `0-sum(raw_effect)` の計算結果をつけくわえています。関数 `c` は結合させる combine という意味です。

8 行目 各群の平均値です。全体平均に効果を足しています。全体平均は 1 つの数字，効果は水準数の要素を持つベクトルですから，計算ができないように思えますが，このような場合 R は自動的にサイズ合わせのため値の再利用を行います。つまり，$50 + (\delta_1, \delta_2, \delta_3, \delta_4, \delta_5)$ を $(50, 50, 50, 50, 50) + (\delta_1, \delta_2, \delta_3, \delta_4, \delta_5)$ と解釈してくれるのです。

10 行目 水準数 $\times$ 各群のサイズの乱数を発生させています。乱数は正規分布によるもので，平均や SD は上で指定した通りです。ここでも値の再利用が行われていて，ベクトル `mu` は $\mu_1, \mu_2, \mu_3, \mu_4, \mu_5, \mu_1, \mu_2, ...$ とくり返して当てはめられていきます。

11-14 行目 数値を `data.frame` 型にくみ上げていってます。`rep` 関数は，指定した回数だけ繰り返すもので，ここでは `1:Lv` すなわち 1,2,3,4,5 という数字をデータサイズ分繰り返していることになります。

16 行目 ここでは `cmdstanr` パッケージを使ってコンパイルしています。

17 行目 推定に使うデータセットです。データ長は `data.frame` の長さを返す関数 `NROW` を使って与えています。インデックス変数は `dat` オブジェクトの `Idx` 変数，データは同じオブジェクトの `value` 変数を指定しています。

いかがでしょうか。作ったデータ，オブジェクト，ベクトルなどの意味がわからない場合は，その都度 R でオブジェクト名を入力し，何が格納されているかを確認しながら進めると良いでしょう。ここで作った仮想データを，先ほどの Stan オブジェクトに与えて乱数を生成し，パラメータリカバリがどの程度の精度でできるかを考えてみてください。サンプルサイズや誤差の大きさなどを変えながら確認してみると良いでしょう。

5.5 課題

今回は，基本課題と発展課題の 2 つを用意します。基本課題は必須ですが，応用課題は提出者に加点されるだけで必須ではありません (提出しなかったからと言って減点されることもありません)。

■基本；整然データに対応した多群比較のコードを書く 最後に紹介した，整然データに対応した多群比較のコードを書き，パラメータリカバリのコードとデータを使って正しく動くかどうか確認してください。作った Stan ファイルや R コードを提出してください。

■発展；自分のデータを使って分析する 基礎実験 1 や基礎実験 2 など，他の授業でとったいくつかの群の平均値を比較するようなデータを用意し，今回のモデルを適用して平均値の比較を行なってください。とくにそのようなデータセットを持っていない場合は，こちらから提供するサンプルデータを使ってください。

[*8] 一様乱数の関数は，R では `runif` です。

第 6 章

確率的プログラミング；項目反応理論

さてここまで，線形モデルを中心に**モデリング**のステップを一段一段登ってきました。この後は，これまでの技術や考え方を応用し，より実践的なプログラミングへと進んでいきます。さまざまなモデルの確率的表現や，そのコーディング技法を学ぶことで，ご自身の研究にも応用できる技術的ヒントが得られるかもしれません。

今回は第??講，第??講および第??講で学んだ項目反応理論を，Stan を使って実装する例を見ていきたいと思います。

6.1 ロジスティックモデルの復習

6.1.1 パラメータモデル側から

ここで簡単に前期の復習をしておきます。

目に見えないものを測定するという意味では，心理尺度や学力検査は同じ技術であるというところから，測定モデルの話を導入しました。とくに学力検査で想定されている測定対象，学力は，正規分布していると考えられること，累積的な能力の蓄積の程度を測定していると考えられることから，正答率が累積正規分布をつかった潜在能力 (学力) の関数と考えられるというところからモデル化がすすみました。累積正規分布を直接扱うモデルもありますが，一般にはそれによく近似する**ロジスティック関数**，すなわち

$$f(x) = \frac{1}{1 + exp(-1.7x)}$$

を使って近似することで，項目の特徴を描写することが考えられたのでした。

x 軸が潜在能力 θ を表していると考えれば，縦軸は能力に応じた**通過率**になると考えられます。この関数を左右に動かす位置パラメータ，b_j を加えたモデルが **1 パラメータ・ロジスティックモデル (One Parameter Logistic model)** であり，次の式 6.1 で表現されるのでした。

$$p_j(\theta) = \frac{1}{1 + exp(-1.7(\theta - b_j))} \tag{6.1}$$

ここで $p_j(\theta)$ は項目 j に対する θ の通過率であり，b_j は**困難度 (difficulty)** 母数と言われるものです。

さらに項目を特徴づけるために，傾きの大きさを表すパラメータ，a_j を加えたモデルが **2 パラメータ・ロジスティックモデル (Two Parameter Logistic model)** です。モデル式は式 6.2 で表されます。

$$p_j(\theta) = \frac{1}{1 + exp(-1.7a_j(\theta - b_j))} \tag{6.2}$$

a_j はのことをとくに**識別力 (discriminant)** と呼ぶのでした。a_j の値によって傾きがどのように変わる

かを，図 6.1 の二段目をみて確認しておきましょう。

最後に，第??講では紹介していませんでしたが，3 つ目のパラメータである c_j，**当て推量母数 (guessing parameter)** を入れたモデルも紹介しておきます。モデルは式 6.3 のように表されます。

$$p_j(\theta) = c_j + \frac{1 - c_j}{1 + exp(-1.7a_j(\theta - b_j))} \tag{6.3}$$

ここには切片のように定数 c_j が入っていますから，関数のベースラインが上に上がるイメージです。これはつまり，学力 θ にかかわらず一定の正答率を有するということで，「適当に答えても当たる確率」ということから当て推量とよばれています。

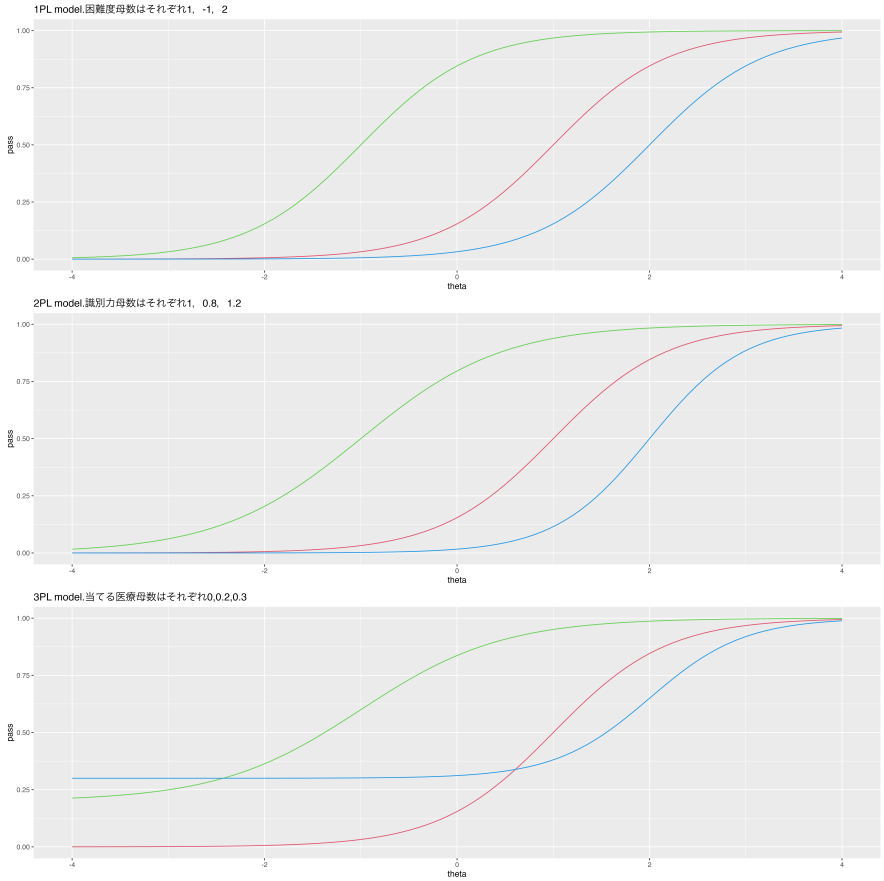

図 6.1　上から順に，1，2，3 パラメータロジスティックモデル。どこがどう変わっていくか確認しておこう。

6.1.2　確率モデル側から

さて今度は同じモデルをデータとの対応の観点から見てみましょう。テストのデータは 0/1 の二値であり，ベルヌーイ分布から出てきていると考えることができます。またロジスティックモデルはその言葉の通りロジスティック関数を使って，$-\infty$ から $+\infty$ まであり得る θ の値を，0 から 1 の範囲に入るように変換しているわけです。これは一般化線形モデルの文脈でいうところの，**ロジスティック回帰**分析であり，**リンク関数**がロジット関数，**逆リンク関数**がロジスティック関数になっているモデルだと考えることができます。

データ生成のモデル設計図として考えると，図 6.2 のようになります。パッケージを使っての分析の時は**最尤法**による推定結果でしたが，ここで被検者母数に標準正規分布を，項目母数に正規分布を事前分布として

おいただけで，ベイズ推定モデルに置き換えることができました。

これまで利用してきた技術の応用で考えることができますから，すぐにでも実装できそうですね！

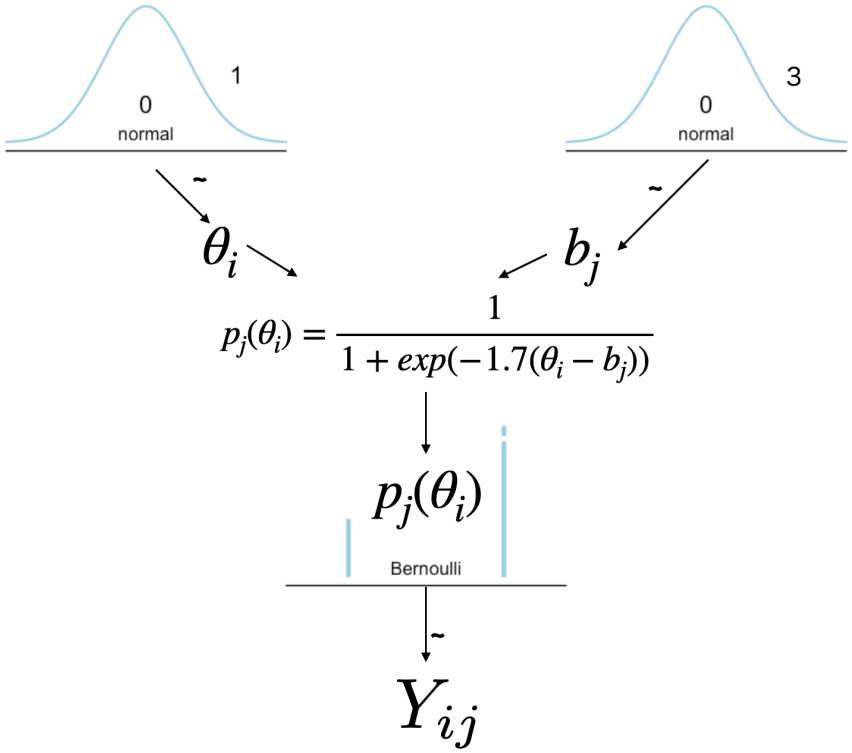

図 6.2　確率モデルとしての設計図。ロジスティック関数とベルヌーイ分布の組み合わせでテスト理論のモデルが表現できる。

6.2　ロジスティックモデルでの実装

それでは設計図をもとにモデルをコードにしていきましょう。

項目数が M，被検者数が N の $N \times M$ サイズの行列でデータが与えられていると考え，行列の各要素に対して通過率 p_{ij} を計算し，それをベルヌーイ分布に当てはめるという形で実装したのがコード 6.1 になります。

code：6.1　1PL モデル

```
1  data{
2    int<lower=0> M;
3    int<lower=0> N;
4    array[N,M] int<lower=0,upper=1> resp;
5  }
6
7  parameters{
8    array[M] real<lower=-5,upper=5> b;
9    array[N] real theta;
10 }
11
12 transformed parameters{
```

```
13    array[N,M] real<lower=0,upper=1> prob;
14    for(n in 1:N){
15      for(m in 1:M){
16        prob[n,m] = inv_logit(1.7*(theta[n]-b[m]));
17      }
18    }
19  }
20
21  model{
22    for(n in 1:N){
23      for(m in 1:M){
24        resp[n,m] ~ bernoulli(prob[n,m]);
25      }
26    }
27    //prior
28    b ~ normal(0,3);
29    theta ~ normal(0,1);
30  }
```

今回はわかりやすくするために，transformed parameters ブロックを使いましたが，直接 model ブロックに書き込んでも構いませんし，bernoulli_logit 関数を使えばさらに高速に安定した推定結果が得られます。

transformed parameters ブロックにパラメータを追加するだけで，2PL,3PL モデルに拡張することも容易にできます。2PL モデルに拡張した例がコード 6.2，3PL モデルに拡張した例がコード 6.3 です。

code : 6.2 2PL モデル

```
1  ...(前略)...
2  parameters{
3    array[M] real<lower=0> a;
4    array[M] real<lower=-5,upper=5> b;
5    array[N] real theta;
6  }
7
8  transformed parameters{
9    array[N,M] real<lower=0,upper=1> prob;
10   for(n in 1:N){
11     for(m in 1:M){
12       prob[n,m] = inv_logit(1.7*a[m]*(theta[n]-b[m]));
13     }
14   }
15 }
16 ...(後略)...
```

code : 6.3 3PL モデル

```
1  ...(前略)...
2  parameters{
3    array[M] real<lower=0> a;
4    array[M] real<lower=-5,upper=5> b;
5    array[M] real<lower=0,upper=1> c;
6    array[N] real theta;
```

```
 7  }
 8
 9  transformed parameters{
10    array[N,M] real<lower=0,upper=1> prob;
11    for(n in 1:N){
12      for(m in 1:M){
13        prob[n,m] = c[m] + (1-c[m])*inv_logit(1.7*a[m]*(theta[n]-b[m]));
14      }
15    }
16  }
17  ...(後略)...
```

第??講で使ったサンプルコードをつかって, このモデルで推定するためのコードがコード 6.4 です (cmdstanr での例)。

code : 6.4 IRT のコード

```
1  dat <- read_csv("IRTsample.csv")
2  model_1pl <- cmdstan_model("oneParameter.stan")
3  model_2pl <- cmdstan_model("twoParameters.stan")
4  model_3pl <- cmdstan_model("threeParameters.stan")
5
6  dataSet <- list(N = NROW(dat), M = NCOL(dat), resp = as.matrix(dat))
7  fit1 <- model_1pl$sample(data = dataSet, chains = 4, parallel_chains = 4)
8  fit2 <- model_2pl$sample(data = dataSet, chains = 4, parallel_chains = 4)
9  fit3 <- model_3pl$sample(data = dataSet, chains = 4, parallel_chains = 4)
```

分析結果を見てみましょう。**EAP 推定値**はそれぞれ表 6.1, 図 6.3 のようになりました。

表 6.1 それぞれのパラメータ推定値

	1PL	2PL		3PL		
Qid	b_j	a_j	b_j	a_j	b_j	c_j
1	0.727	0.347	1.716	2.865	1.567	0.221
2	-1.605	0.724	-2.140	1.818	-0.761	0.557
3	-1.175	1.124	-1.229	2.900	-0.626	0.338
4	-1.077	0.955	-1.222	2.033	-0.535	0.362
5	0.577	0.691	0.771	2.451	0.946	0.159
6	1.271	0.694	1.742	1.100	1.659	0.050
7	0.564	0.692	0.753	1.027	0.941	0.099
8	1.590	0.360	3.650	1.581	3.355	0.084
9	0.514	0.484	0.893	0.836	1.205	0.142
10	-0.864	1.214	-0.887	2.547	-0.395	0.298

6.3 整然データでの分析

このように, IRT モデルを簡単に実装できました。

ところで, このモデルをもう少し使いやすくするために, データを**整然データ (tidy data)** にすることを考

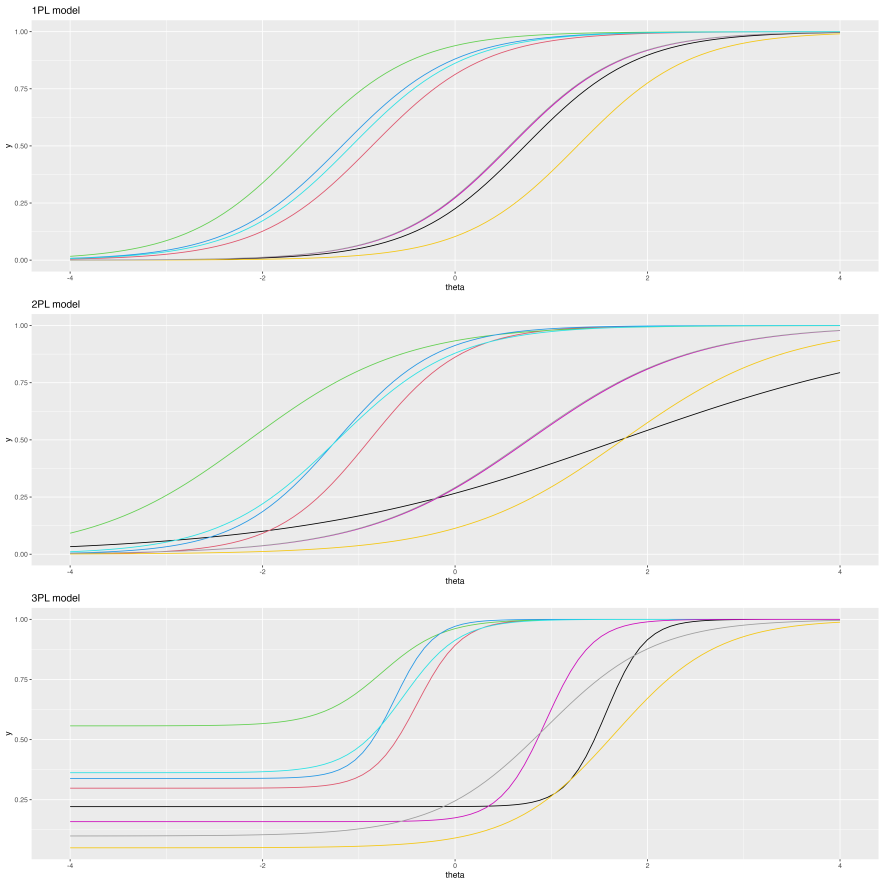

図 6.3　推定値を使って描いた各モデルの ICC

えてみましょう。というのも，今はデータを行列形で渡していますが，これだと欠損値があった場合にうまく機能しないからです。Stan はデータに欠損値を取ることができず，与えられるデータに `NA` が入っていることは許されません。

整然データは，1 行に 1 つの数字が入っているようなデータで (セクション 5.3.1，Pp.67 参照)，その行を見るだけですべての情報が手に入るようなデータです。行列型のデータは，人の ID を行ラベルで，変数の ID を列ラベルでみなければなりませんので，参照方向が 1 つに定められていません。具体的には次のような形のデータになります (**R の出力 6.1**)。この形式ですと，一行で誰の (`Pid`)，どの問いに対する (`Qid`) 反応か (`value`) ということがわかりますね。

R の出力 6.1: 整然データになったテストデータ

```
# A tibble: 20 × 3
     Pid   Qid value
   <int> <dbl> <dbl>
 1     1     1     0
 2     1     2     0
 3     1     3     1
 4     1     4     0
 5     1     5     0
 6     1     6     0
 7     1     7     0
 8     1     8     0
 9     1     9     0
10     1    10     0
11     2     1     1
12     2     2     1
13     2     3     0
```

もし欠損値があればその行を削除してしまえば，完全データになりますから，Stan に欠損値を与えなくて済むことになります。

もちろんこれに対応する形で，Stan のコードを書き換える必要があります。こんどはデータも長くなりますから，何行目に誰のど問いに対する反応が入っているかを，識別変数を使いながら指定することになります。具体的には次のようなコード例になるでしょう (コード 6.5)。

code : 6.5 Tidy Data に対応した 2PL モデル

```
data{
  int<lower=0> L;
  int<lower=0> N;
  int<lower=0> M;
  array[L] int<lower=0,upper=N> Pid;
  array[L] int<lower=0,upper=M> Qid;
  array[L] int<lower=0,upper=1> resp;
}

parameters{
  array[M] real<lower=0> a;
  array[M] real<lower=-5,upper=5> b;
  array[N] real theta;
}

transformed parameters{
  array[N,M] real<lower=0,upper=1> prob;
  for(n in 1:N){
    for(m in 1:M){
      prob[n,m] = inv_logit(1.7*a[m]*(theta[n]-b[m]));
    }
  }
}

```

```
25  model{
26    for(l in 1:L){
27      resp[l] ~ bernoulli(prob[Pid[l],Qid[l]]);
28    }
29    //prior
30    a ~ normal(0,3);
31    b ~ normal(0,3);
32    theta ~ normal(0,1);
33  }
```

■コード解説

data **ブロック**　データ長 L, 最大被検者数 N, 最大項目数 M, 被検者識別変数 Pid, 項目識別変数 Qid, L 行目の反応 resp としてデータを受け取ります。

parameters **ブロック**　これまでと同じです。

transformed parameters **ブロック**　ここも手を加える必要がありません。

model **ブロック**　L 行目の反応に対して, 識別変数で特定された確率をつかってモデルを当てはめます。

これの結果はこれまでのモデルと変わりませんが, 欠損値に対応できたはずですので, 元の完全データに技と欠損値を与えて改変し, 結果がどう変わるかを確認してみましょう。

code : 6.6　tidy データにして技と欠損値を与える

```
1  dat.tmp <- dat %>%
2    rowid_to_column("Pid") %>%
3    pivot_longer(-Pid) %>%
4    mutate(Qid = str_extract(name, pattern = "\\d+") %>% as.numeric()) %>%
5    dplyr::select(Pid,Qid,value)
6
7  # わざと欠損値を与える
8  dat.tmp$value[1] <- NA
9  dat.tmp$value[11:13] <- NA
```

■コード解説

1 行目　dat に入っている行列型･完全データを変形していきます。

2 行目　一行ごとに被検者の反応が入っていますので, 行番号を被検者識別変数 Pid として作ります。

3 行目　データを縦長にします。その時のキーとなるのは, 先ほど作った Pid で, この変数は除いて縦長にする関数が pivot_longer です。

4 行目　少し技巧的ですが, ここまでの段階で変数は Pid,name,value という 3 つになっています。なかでも name 変数には元データの変数名が入っていますので, これを加工して問題番号を取り出します。str_extract 関数の str とは string, すなわち文字列を扱う関数であるという意味です。str_extract は文字列から条件に合ったものを抜き出す extract というもので, 条件を pattern で指定しています。ここで\\d+ とあるのは**正規表現 (regular expression)** というもので, 文字列を一定の規則に従って特定の文字列を表現する方法です。ここでは数字だけを抜き出しています[*1]。

[*1] 少し丁寧にいうと, \d は正規表現で数字を意味する記号です。ただ, バックスラッシュ (\) がそのままでは正規表現の記号だと認識されず特殊文字だという宣言をしている (エスケープシーケンス, といいます) だけになってしまいます。そこで,「特殊文字

また，このようにして取り出せた数字は文字列としての数字ですので，これを as.numeric 関数に送ることで，数値であることを教えています。

5 行目 被検者識別変数，項目識別変数，反応の値だけにデータを限定しています。

7 行目 ここでこのデータセットの 1 行目に欠損値 NA を上書きしています。わざと欠損させたのです。

8 行目 同じく 11 行目から 13 行目までも欠損値に上書きしてしまいました。

こうしてできたデータセットは次のようになります (R の出力 6.2)。

R の出力 6.2: 整然データになったテストデータ

```
# A tibble: 20 × 3
     Pid   Qid value
   <int> <dbl> <dbl>
 1     1     1    NA
 2     1     2     0
 3     1     3     1
 4     1     4     0
 5     1     5     0
 6     1     6     0
 7     1     7     0
 8     1     8     0
 9     1     9     0
10     1    10     0
11     2     1    NA
12     2     2    NA
13     2     3    NA
14     2     4     0
15     2     5     0
16     2     6     0
17     2     7     0
18     2     8     0
19     2     9     0
20     2    10     1
```

確かに数カ所，欠けているところができましたね。これをそのまま渡すことができませんから，欠損のあるところは削除してデータセットを作り，推定してみましょう。

code : 6.7 データセットを作って推定する

```
1 # 欠損値を消したデータセットにする
2 dat.tmp <- na.omit(dat.tmp)
3
4 dataSet <- list(
5   L = NROW(dat.tmp), N = max(dat.tmp$Pid), M = max(dat.tmp$Qid),
6   Pid = dat.tmp$Pid, Qid = dat.tmp$Qid,
7   resp = dat.tmp$value
8 )
```

のバックスラッシュだよ」を表すために重ねて\\としています。また，+ は正規表現でいうところの「直前の文字が 1 回以上繰り返される」という意味です。今回は 10 という数字が出てくる可能性があり，\\d だけだと 1 しか抜き出せないので，数字が連なっていたら全体を取り出すようにしています。要するに，数字を全部取り出しましょう，が正規表現では\\d+ になるわけです。

```
 9
10  model_2pl_ver2 <- cmdstan_model("twoParameters2.stan")
11  fit2.2 <- model_2pl_ver2$sample(
12    data = dataSet,
13    chains = 4,
14    parallel_chains = 4
15  )
```

欠損があっても，データを整形して欠損を取り除いた形で分析し，問題なく推定できたと思います。欠損が含まれているから，そのデータはすべて使い物にならないと考えるのではなく，データの存在するところ・利用できるところは利用しつくすという有効活用ができたと思います。

最後に，推定結果を確認しておきましょう。1 人目の被検者は 1 つ，2 人目の被検者は 3 つの欠損がありました。つまり他の被検者は 10 問分の情報を持っているのに，この 2 人はそれぞれ 9 問，7 問分しか情報が得られなかったことになります。そのことが結果の推定値にどう変わるのかというと，出力 7 の通りです。

MCMC の結果 7

```
# A tibble: 3 × 7
  name           EAP       MED       MAP        SD       L95       U95
  <chr>    <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!>
1 theta[1]    -1.403    -1.392    -1.438     0.548    -2.530    -0.366
2 theta[2]    -0.774    -0.778    -0.809     0.606    -1.982     0.372
3 theta[3]    -0.656    -0.659    -0.620     0.508    -1.654     0.331
```

これを見ると，10 問分の情報を持っている 3 人目の被検者の能力値が $\theta_3 = -0.656$ と推定されていますが，その SD が 0.508 です。これに比べて，9 問しか情報のない被検者 1 の SD は 0.548，7 問しか情報のない被検者 2 の SD は 0.606 と，情報が少なくなると SD が大きくなっていくことがわかります。推定値の SD が大きいということは，幅が広い，すなわちわからないことがより多くあることを意味します。得られる情報が少なければ，絞り込みが難しくなるというのがデータにも表れていることがわかりますね。

6.4 課題

本講で扱った，1PL，2PL，3PL モデルそれぞれを実行する，Stan ファイルや R コードを提出してください。ただし，いずれのモデルも整然データに対応したコードになっている必要があります。データや R コードはシラバスのサイトを通じて提供されています。不明な点がありましたら，TA あるいは小杉まで連絡して指導を受けてください。

第 7 章

確率的プログラミング；変化点と折線回帰

さて今回は，今までの線形モデルとはちょっと違うモデリングになります。その名も変化点検出，そして折線回帰です。タイトルだけでも面白そうではありませんか？

さらに今回は次のようなデータを扱います (図 7.1)。何を隠そうこのデータは，私の体重の推移のデータなのです。私のモーニング･ルーティンとして，朝目が覚めるとトイレに行きます。そして体重計にのり，体重と体脂肪を測定します。その後顔を洗って[*1]，計測値を iPhone のアプリに書き込むというのがあります[*2]。このルーチンも早いもので 10 年近く続けていることになります (一時期やめていたこともありましたが，最古のレコードは 2012 年 06 月 08 日です)。ともかく，このデータを使って，いろいろ遊んでみようと思います。

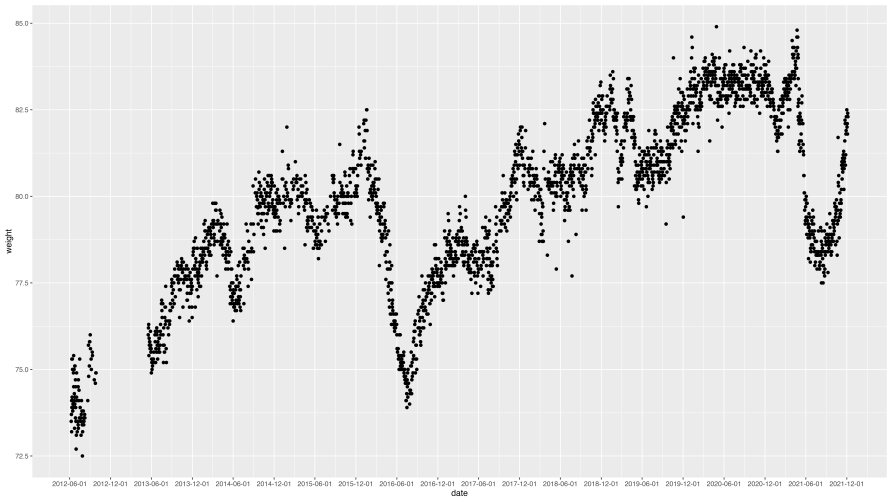

図 7.1　体重の時系列的推移

[*1] 体の内側の水分を出し，表皮に水分をつけないようにすることで，しっかりと肉の量を測ろうとしているのでこの順番になります。

[*2] ついでにいうと，書き込んだ結果はツイートします。フォロワーの中には私の体重の推移を楽しみにしている人がいるのです。おかしな世の中です。

7.1　混合分布モデルの応用

データが 10 年分というのはちょっと多いですから，少し時期を区切ることにします。2019 年から 2020 年の 2 年間に限定したデータが，図 7.2 になります。

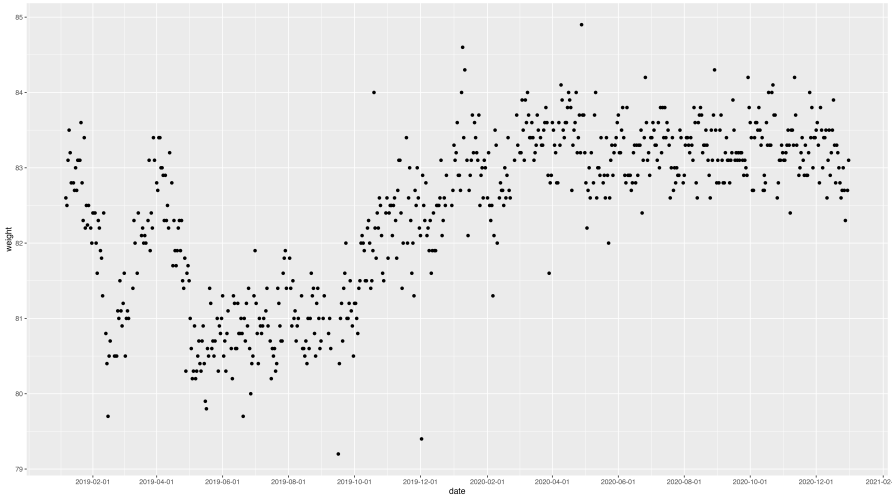

図 7.2　2019-2020 年のデータ

これをみると，うーん残念なことに，2020 年になると体重が増えてしまっているようですね。2019 年は 81kg 台をうろうろしていたようですが，2020 年になると 83kg 台になっているようです。毎回の体重の計測に偶然の測定誤差がついているとしても，19 年と 20 年とではそもそも体重が違ってしまっているようです。これは平均値の違う正規分布が混合している，混合分布モデルを考えてみることができそうですね。

第??講を参考に，混合分布モデルを考えてみましょう。筆者の体重には 2 つの状態があり，軽い方の状態なのか重い方の状態なのかは，確率 θ で変わると考えてみます。するとデータは，確率 θ で μ_1 を平均とする正規分布から，確率 $1-\theta$ で μ_2 を平均とする正規分布から得られるわけですから，確率モデルは次のようになります。

$$p(W_j) = \theta_j \times N(\mu_1, \sigma) + (1-\theta_j) \times N(\mu_2, \sigma)$$

ここで θ_j としたのは，観測時点 j ごとに θ が変わるという想定です。どちらのモードになるのかは一定の数字というより，毎回違っているように思えたのでそのようにしてみました。されてこを Stan で実装するにはどうすれば良いのでしょうか。そう，`log_sum_exp` 関数を使って，2 つの状態をベクトルで表記し，それを足し合わせる必要があるんでしたね。実際にコードにしてみたのがコード 7.1 です。

code：7.1　2 つの体重モードモデル

```
1  data{
2    int L;
3    array[L] real W;
4  }
5
6  parameters{
7    array[L] real<lower=0,upper=1> theta;
```

```
 8    ordered[2] mu;
 9    real<lower=0> sigma;
10  }
11
12  model{
13    for(l in 1:L){
14      target += log_sum_exp(
15        log(theta[l]) + normal_lpdf(W[l]|mu[1],sigma),
16        log1m(theta[l]) + normal_lpdf(W[l]|mu[2],sigma)
17      );
18    }
19
20    mu ~ normal(80,10);
21    sigma ~ cauchy(0,5);
22  }
```

これを使って，体重のデータを分析してみましょう。コード 7.2 を実行し，図 7.3 のような結果を得ます。

code : 7.2 混合分布モデルのコード

```
1  dat1 <- dat %>%
2    dplyr::filter(date > "2019/01/01") %>%
3    dplyr::filter(date < "2021/01/01")
4
5  model <- cmdstanr::cmdstan_model("changePoint1.stan")
6  dataSet <- list(L = NROW(dat1), W = dat1$weight)
7  fit <- model$sample(
8    data = dataSet,
9    chains = 4,
10   parallel_chains = 4,
11   seed = 12345)
```

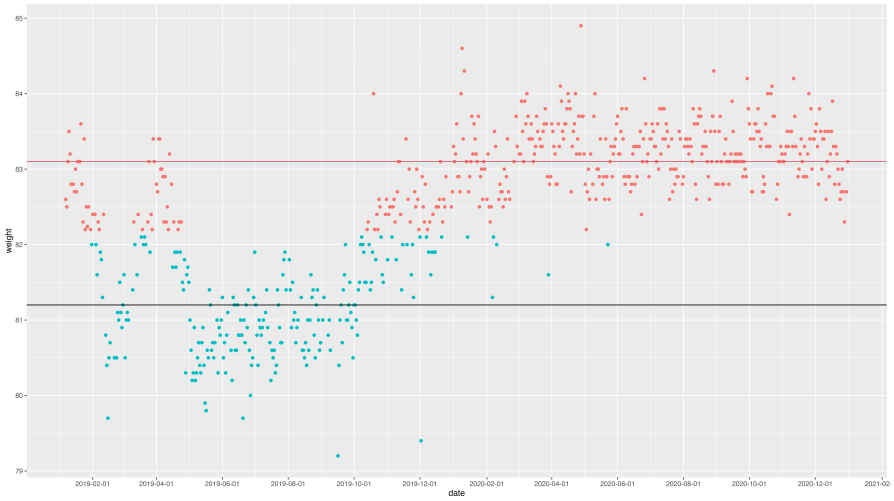

図 7.3 2 つの体重のモード

結果の図を見ると，どうやら 2019 年の初期にも 2 回ぐらい平均値が高い，「重いモード」が混在してお

り[*3]，2019 年の 10 月ごろからそちらの比率が増えています。もっとも，2020 年の 1 月にも「軽いモード」に入っている点はあるようですね。

とまあ，このような分析結果になったわけですが，これをみるとデータが 82kg ぐらいのラインを超えたかどうかで分割されているな，というのがわかります。当然，平均値の違う 2 つの分布を混ぜたわけですから，平均値の違いによって分かれるわけです。それにしても 2019 年の初期はどうしたんでしょうね。1 月や 4 月は重いモード，2-3 月と 5 月以降は軽いモードと，コロコロ入れ替わっています。ここでは 2 つのモードのどちらからデータが出てきているのか，ということだけを考えているのでこれでいいのですが，体重というのはそもそも時系列的な変化をするものですから，どこかで重くなった，どこかで軽くなった，というような時系列的なつながりについての情報が，うまくモデル化されていないように思えます。そこで，横軸が時間的な連続であるということに注意して，今度は 2021 年 1 月から 11 月までのデータを見てみましょう。その区間を取り出したのが図 7.4 になります。

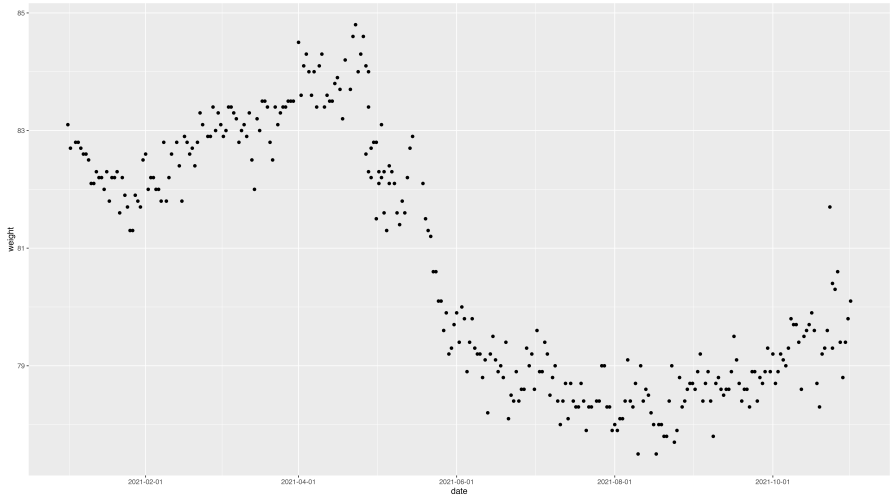

図 7.4 大きく変化したデータ

これは 2 つのモードがあるというより，途中で大きく変化したというべきではないでしょうか。とくに 2021 年 5 月ごろから 4kg ほど体重がぐぐぐっと下がることがあり，6 月以降は下がった体重のレベルに落ち着いているようです。このように，横軸が時系列だと考えると，5 月に何か変化があったのではないか，ということが推察されます。これをモデルで表現してみましょう。

7.2 変化点検出

何かのきっかけでデータの様相がガラリと変わってしまった，その変化したところを変化点と呼び，ここでの課題はその変化点を見つけ出す**変化点検出 (Change point detection)** ということになります。たとえばセンサーが検出するデータの針が急に違うレベルに変化すると，測定している対象の状態がガラリと変わったのではないか，と考えることができますね。

他にもたとえば，心理学の応用領域として，科学捜査研究所が担当する**ポリグラフ検査**というのがあります[*4]が，これなども針が大きく触れたことで変化をみることになります。

[*3] 誰がデブモードやねん

[*4] 皮膚電気活動や心拍，呼吸など複数の整理指標を同時に測定するので**ポリ**グラフであり，熟練の検査官が反応パターンから被疑者の特別な反応を検出するものです。嘘発見器と呼ばれることがありますが，正確には**虚偽検出**と読んだほうが良いでしょう。

変化点検出はその名の通り「いつ」変化したのかを検出するものです。今回のデータも，だいたい 5 月ごろに大きな変化があったというのはわかるのですが，いつなのかを特定するのは難しいところです[*5]。これをデータとモデルから明らかにしようというのです。

ベイズ統計はわからないことを確率で表現し，データでその確率情報をアップデートしていくというものです。今回はいつ変化したのがわかりませんから，その変化した時期を τ とし，データ区間の中にその日があると考えて，その区間の一様分布を事前分布とします。その時期 τ を境に，データが出てくる分布の位置パラメータがずれると考えるのです。設計図は次のようになります (図 7.5)

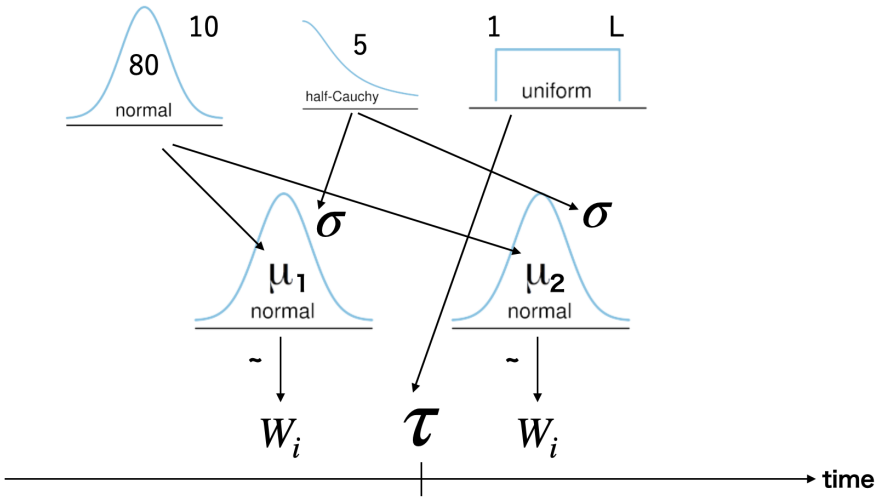

図 7.5 変化点検出のモデル設計図

設計図をもとに，コードにしてみたのがコード 7.3 になります。

code : 7.3 変化点検出のモデルコード

```
1  data{
2    int L;
3    array[L] real W;
4  }
5
6  parameters{
7    real<lower=1,upper=L> tau;
8    ordered[2] mu;
9    real<lower=0> sigma;
10 }
11
12 model{
13   for(l in 1:L){
14     if(l < tau){
15       W[l] ~ normal(mu[2],sigma);
```

最近では**隠匿情報検査 (concealed information test)** ということもあります。ちなみに科学捜査研究所，通称科捜研は，心理の他にも物理，化学，法医，文書 (筆跡鑑定) などの専門領域があり，各都道府県に 1 つずつ設置されています。

[*5] とはいえこのデータは筆者自身の体重変化ですので，何があったのかは実はわかっています。5 月 11 日，筆者が帰宅途中に自転車で転倒する事故を起こし，右肩の鎖骨を骨折しました。17 日に手術した後，利き手が使いにくいものですから食事の量が減り，結果的に体重 (筋肉?) が落ちたというのが真相です。

```
16      }else{
17        W[l] ~ normal(mu[1],sigma);
18      }
19    }
20
21    tau ~ uniform(1,L);
22    mu ~ normal(80,10);
23    sigma ~ cauchy(0,5);
24  }
```

日々のデータが順に 1 から L 行目まで並んでいるとします。パラメータ tau があり，1 行目のデータが変化点 tau より前にあるときは μ_2, σ の正規分布から，tau より後になれば，μ_1, σ の正規分布からデータが出てくると考えるのです。今回は後半の平均値が小さいことが明らかですから，μ のベクトルを ordered 型で宣言してあります。ベクトル要素の小さい順に並びますから，前半が μ_2，後半が μ_1 としています。

これを実行して，変化点がいつになるのかを推定してみましょう。推定結果は出力 8 のようになりました。

MCMC の結果 8

```
# A tibble: 4 × 7
  name        EAP       MED       MAP        SD       L95       U95
  <chr> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!>
1 mu[1]    82.770    82.769    82.762     0.062    82.648    82.893
2 mu[2]    78.891    78.892    78.896     0.060    78.775    79.008
3 sigma     0.750     0.749     0.747     0.031     0.693     0.812
4 tau     144.642   144.578   144.431     0.646   143.321   146.190
```

データの 145 行目がちょうどその変化点だといって，ほぼ間違いないようですね。145 行目はといいますと，データから 5 月 23 日だったことがわかります。

R の出力 7.1: いつでしょう

```
> dat2[145, ]
# A tibble: 1 × 3
  date                weight bodyFat
  <dttm>               <dbl>   <dbl>
1 2021-05-23 06:50:57   80.6      26
```

違いがわかるようにデータとモデルの推定値を合わせたのが，図 7.6 です。このようにして，ここで様相が変わったのだなということを，データから見出すことができました。

7.3 折線回帰

先ほどは，変化点を機に平均値が変わる，というモデルでした。では次のような分布の場合はどうなるでしょうか。今度は 2016 年に注目してみました (図 7.7)。この年はダイエットを志した時期で，最初の半年ぐらいでぐんぐん体重が落ちていってるのがわかります。そして残念なことに，8 月ごろでしょうか，ダイエットが終わりリバウンドが始まったのがよくみて取れますね[*6]。

[*6] このときのダイエット法は，1. ラーメンのスープは飲まない，2. 夕食と晩酌は分ける (食べながら飲むのではなく，食べ終わって，お酒だけ飲む),3. お酒を飲むときにおつまみは食べない，という 3 か条を守るというものでした。お酒が飲みたいので食事の量

図 7.6 検出された変化点と平均値の違い

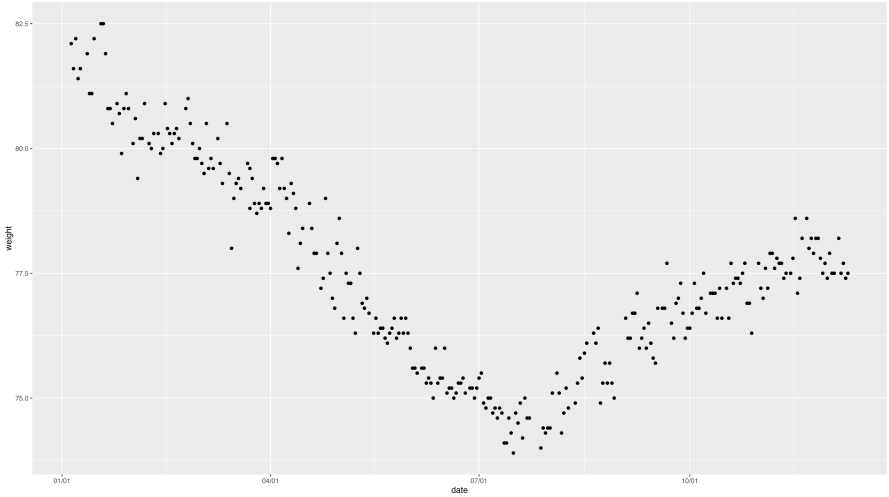

図 7.7 2016 年の変化

さてこのデータに対して，先ほどの変化点検出もできるのですが，変化点の前後で平均値が違う，というような簡単な話ではなさそうです。変化点の前はどんどん数字が減っていき，変化点のあとはどんどん数字が大きくなっていく，というのが実態です。回帰分析をしたら，変化点前は負の傾きが，変化点後は正の傾きが推定されそうな，そんなデータになっていますね。ということで，そのイメージができたのであれば，そのままモデルを描いてしまいましょう。設計図が少しごちゃっとしてしまいましたが，ポイントは傾きの係数を前半は負，後半は正に限定しているところでしょうか (図 7.8)。また，変化点は大体このあたり･･･ということでデータの 100 行目 (2016/04/29) から 250 行目 (2016/10/15) の間に置いてみました。

設計図をもとに，コードにしてみたのがコード 7.4 になります。

code：7.4 折線回帰のモデルコード

を減らし，すぐに飲むことにしたので体重が減り，食事の前に飲んでしまえばいいんじゃないかという裏技を見つけてダイエットが崩壊したのを覚えています。

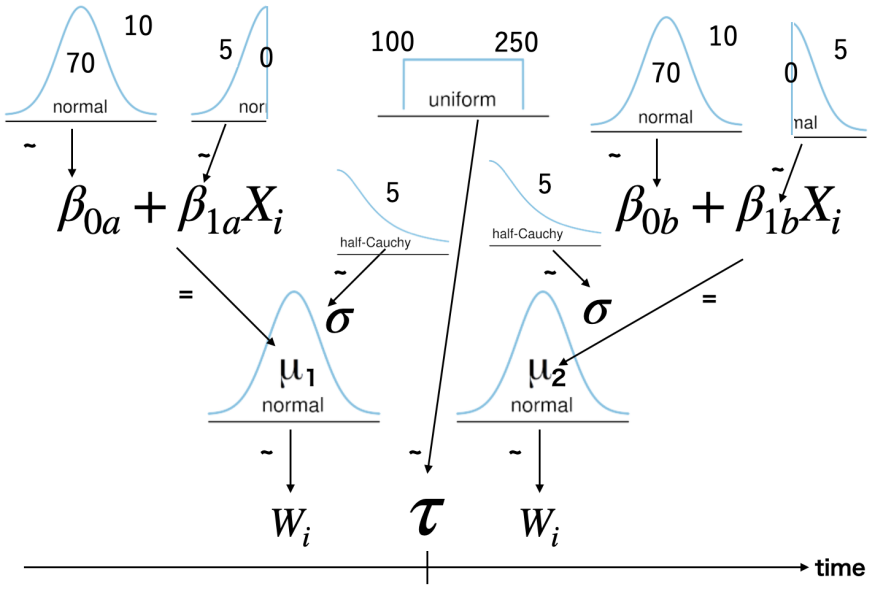

図 7.8 折線回帰モデル設計図

```
data{
  int L; // data length
  array[L] real W;
  array[L] real X;
}

parameters{
  real<lower=100,upper=250> tau;
  array[2] real beta0;
  real<upper=0> beta1a;
  real<lower=0> beta1b;
  real<lower=0> sigma;
}


model{
  for(l in 1:L){
    if( l < tau ){
      W[l] ~ normal( beta0[1] + (beta1a * X[l]),sigma);
    }else{
      W[l] ~ normal( beta0[2] + (beta1b * X[l]),sigma);
    }
  }

  beta0 ~ normal(70,10);
  beta1a ~ normal(0,5);
  beta1b ~ normal(0,5);
  sigma ~ cauchy(0,5);
}
```

回帰係数を正・負に限定するところは，パラメータの宣言で<lower=0>や<upper=0>としていることで表現しています。あとはデータが出てくる正規分布の平均に，線形モデルが入っているだけですね。これを実行すると，出力 9 のような結果が得られます。データの 187 行目 (2016/08/01) あたりが変化点ですかね。95% 区間で言うと 185 行目 (2016/07/30) から 188 行目 (2016/08/02) の間に変化点があると言えそうです。

MCMC の結果 9

```
# A tibble: 6 × 7
  name           EAP       MED       MAP        SD       L95       U95
  <chr>    <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!> <num:.3!>
1 beta0[1]    81.871    81.872    81.874     0.075    81.721    82.014
2 beta0[2]    70.548    70.551    70.549     0.380    69.793    71.274
3 beta1a      -0.043    -0.043    -0.043     0.001    -0.044    -0.041
4 beta1b       0.026     0.026     0.026     0.002     0.023     0.029
5 sigma        0.499     0.498     0.497     0.021     0.461     0.543
6 tau        187.135   187.326   187.460     0.832   185.057   188.215
```

そして回帰係数をプロットしてみました (図 7.9。なかなかデータにフィットしていそうです。

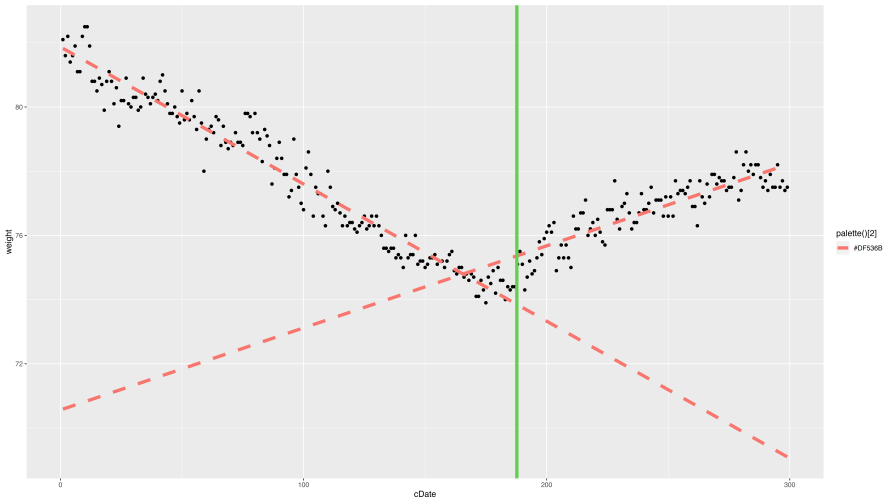

図 7.9 折線回帰モデル推定結果

このままでもよいのですが，変化点と折れたポイントが合致しないのはなんだか気持ち悪いですね。これを合わせることを考えてみたいと思います。変化点 τ のあるところで 2 つの回帰線は交わる，つまり同じ値になるはずですから，$\beta_{0a} + \beta_{1a}\tau = \beta_{0b} + \beta_{1b}\tau$ という式が成り立つはずです。ここから逆算して，

$$\beta_{0a} + \beta_{1a}\tau - \beta_{1b}\tau = \beta_{0b}$$

と考えることができますから，推定するパラメータを 1 つ減らすことができます。

code : 7.5 折線回帰のモデルコード 2

```
1  ...(前略)...
2  parameters{
3    real<lower=100,upper=250> tau;
```

```
 4    real beta0a;
 5    real<upper=0> beta1a;
 6    real<lower=0> beta1b;
 7    real<lower=0> sigma;
 8  }
 9
10  transformed parameters{
11    real beta0b;
12    beta0b = beta0a + ((beta1a-beta1b) * tau);
13  }
14  ...(後略)...
```

このコード 7.4 にあるように，beta0b を計算式から出すようにして，改めて推定をした結果が図 7.10 になります。これだと変化点が前にずれて，データの 169 行目 (2016/07/10) で心がポッキリ折れていることがわかります。

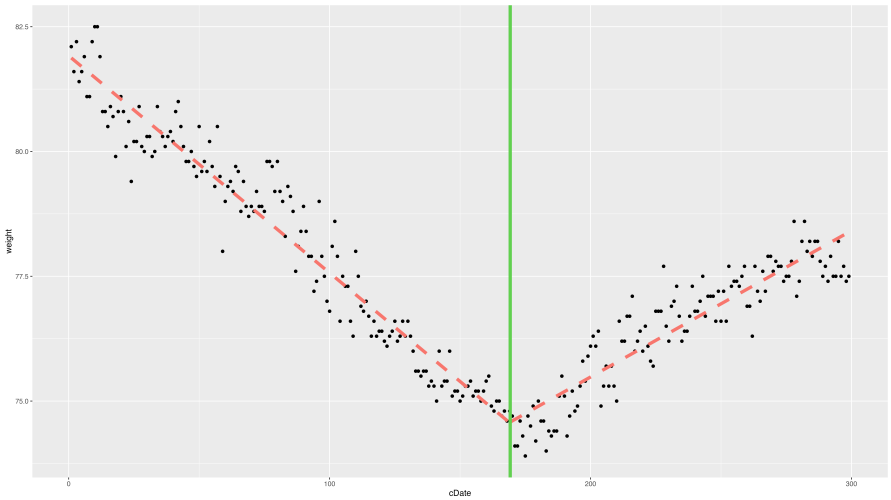

図 7.10　変化点を合わせた折線回帰モデル

このように，モデリングの力を借りるとわからなかった変化点がどこにあるのか，そしてその変化点の前後で予測モデルを変えるといったようなことが簡単に表現できます。ただの回帰分析であっても，データにあったモデルを考えることでさまざまな応用可能性が出てくるのではないでしょうか。

ところで，今回は時系列的なデータに対して回帰分析を行いました。この時の説明変数 X_i は日付，あるいは「何日目か」というデータだったわけです。しかし回帰分析には重要な仮定として，各データ点は (同じ分布から) 独立に得られていると言うものがありました。たとえば身長と体重の回帰分析，という話をするときに，各データ点にあたる個々人の身長・体重は，互いに影響し合わない独立だったはずであり，だからこそ尤度の計算の時に各データの尤度を掛け合わせていくことができたのでした。それに対し，体重のデータは明らかに違います。すなわち，今日の体重が明日の体重に影響している，それどころか今日の体重は明日の体重と大いに関わっているはずですし，明日明後日とその後の日々にも少なからず影響しているはずなのです。

このように考えると，独立していないデータ点，時系列的なデータに対しては異なる分析をする必要がありそうです。次回はこの時系列的なデータを扱うモデルを紹介していくことになります。

7.4 課題

2015 年の 1 月 1 日から 2015 年の 12 月 1 日の間にも，体重の減少と増加が切り替わったところがありそうです。変化点を合わせた折線回帰を実行し，何月何日に変化点があったのか推定するモデルを分析する R/Stan コードを提出してください。結果の解釈などを，スクリプトのコメントアウトや別添ファイルなどで提供してもらえるとなお良いです。もちろん Rmd ファイルでの提出であれば完璧です。なお提出されたコード単体でバグがなく動くことが確認できないものは，未提出扱いになります。コードの書き方などわからないところがあれば，曜日別 TA か小杉までメールで連絡し，指導を受けてください。

第 8 章

確率的プログラミング；状態空間モデル

前回は体重の変化データを例に，データの平均点が変わるモデル，変化点を検出したり変化点を境に傾きが変わる線形モデルをみてきました。

同じように，今回も時系列的なデータを扱います。前回の最後に案内したように，時系列データに普通の回帰直線をそのまま当てはめることは適切ではありません。どのような注意点が必要なのかについて，少し考えてみましょう。

8.1 時系列データの特徴

時系列的なデータはどのようなときに得られるでしょうか。前回導入した時のように，誰か 1 人が日々の生活を記録し続けている，それも立派な時系列データです。心理学には**日誌法**と呼ばれるデータ収集法があります。一言で行ってしまえば日記なのですが，日々多くの出来事や感情が到来するのを後から振り返って考えるのでは，正しく思い出せなかったり記憶が歪んでしまうこともあります。日々の状態を細かく記録し積み重ねることは，数字になっていなくても貴重なデータなのです。また最近は，**経験サンプリング**と呼ばれる調査法もあります。これは決まった間隔で質問紙がスマートフォンなどに送られてきて，定期的に心情を心理尺度に反映させていく，というものです。こうしたことができるようになったのには，私たちの周りに電子的なデバイスがたくさんあることが理由の 1 つです。さらに最近ではウェアラブル端末といって，身につける電子端末がありますね。Apple Watch など携帯電話と同じような機能を持ったこれらのデバイスは，GPS 機能がついていたり，万歩計による歩数の計測，血圧・脈拍のリアルタイムな計測が行われていたりします[*1]。これを蓄積したデータとすると，かなりの時系列的なデータが手に入ることになります。

あるいはまた，社会的なデータも時系列的に得られるものが多いです。一番わかりやすいのは株価の指標などでしょうか。日経平均株価など，時事刻々と変化する株価がグラフになって表されているのをみなさんもみたことがあると思います。また，Twitter などの SNS で，今どのようなキーワードが使われているかといったことが，リアルタイムに計測されます。これらの指標は個々人の特徴というより，社会全体のうねりのようなものを表現していることになりますが，これを研究することも立派な社会心理学的テーマになり得ます。時系列データは，人文社会科学領域ではこれまで経済学やマーケティングなどが専門的に扱ってきたところがありますが，これからは心理学の領域でもこうしたデータを分析することが流行してくるかもしれません[*2]。

さて時系列的なデータはどういった特徴があるかと言うと，端的に言えば**自己相関 (auto correlation)** がある，ということでしょう。すなわち，ある時点のデータはその前の時点と相関している，ということです。体重のデータの場合でもそうですが，明日の体重は突然ランダムな値になるのではなく，今日の体重に応じ

[*1] こうしたデータは**ライフログ (Life-log)** と呼ばれることもあります。

[*2] もちろん生理心理学や動物心理学では，古くからこうしたデータを扱ってきています。

て変化するはずだからです。また，データの変動に周期的なパターンが存在することもあります。脳波などのデータは基本的に波ですから，大きな波，小さな波がパターンを描いています。株価など社会的なデータも，季節による一定の変動など全体的な揺らぎがある上で，目的とする意味のある変化を見つけ出さなければならないという課題に取り組んでいます。たとえば周波数のデータの平均はゼロになりますから，基本的な記述統計では太刀打ちできないところがあります。回帰分析もその前提として，各データが独立に得られていると言うものがありますから，自己相関するデータは当然この仮定に違反していることになります。こうしたデータを分析するためには，周波数の大きさをスペクトル解析するとか，多次元の行列であるテンソルを使ってさまざまな要素を分解する，といった特殊な応用数学を駆使する必要があります。

今回はこうしたさまざまなモデルの中でも，**状態空間モデル (State Space Model)** を紹介します。

8.2 状態空間モデル

時系列的なデータは，自己相関すなわち前の時点の状態が，次の時点に影響しているという特徴があり，これをうまく表現するのが状態空間モデルです。状態空間モデルはこれまでの**モデリング**の技法を使うと簡単に実装できます。というか，これまでの時系列的なデータ解析は，自己相関を減らすために一時点前のデータとの差分をとってそれをモデルにする，時点ごとの変化をスムージングするなど，さまざまな工夫の上に成り立っていました。状態空間モデルはそれを非常にすっきり表現してくれているので，それまでの分析ノウハウがなくとも誰でも簡単に利用できるモデルだといえるかもしれません。

状態空間モデルは，まず観測されたデータと，その背後にある**状態**を区別します。体重変化のデータの例で考えてみましょう。計測された体重は，当然体の重さを反映したものですが，測定の時に多少の誤差が生じるでしょう。例に使っている筆者のデータでも，毎朝全裸で計測しているわけではありませんから，季節の変わり目で薄いパジャマからジャージにしたり，冬用の厚手のパジャマに変えたりすると，それだけで計測された数字には違いができます。本来知りたいのは，体の重さ，肉の重さそのものなはずですね。ここでいう体重そのもの，計測誤差のない肉の重さが，ここでいう状態のことになります。

そして t 時点の状態 μ_t は，次の時点の状態に影響します。服の重さなどを取り除いた t 時点の肉の状態に，食事や運動といった変化が加わって明日 $t+1$ 時点の肉の状態になるわけですから，$\mu_{t+1} = f(\mu_t)$ と考えることができるわけです。

もし翌日の体重が，今日の体重にちょっとした誤差がつくような変動しかしない，と言うのであれば $\mu_{t+1} \sim N(\mu_t, \tau)$ と表すことができるでしょう。あるいは，運動量 P_t の関数だというのであれば，$\mu_{t+1} \sim N(\mu_t + \beta_1 P_t, \tau)$ のように回帰分析の確率モデルのように表現すれば良いでしょう。もちろん説明変数が増えたり，一次関数ではないものを考えても構いません。

一方，計測値 W_t は，この μ_t に誤差 σ がついて得られる，すなわち $W_t \sim N(\mu_t, \sigma)$ という関係になります。これらの関係を表現したのが，図 8.1 になります。このようにして，状態が時間 $t, t+1, t+2, \cdots$ を通じてつながっており，状態に応じて観測値が得られると考えます。観測値を手に入れる際には偶然誤差 σ が生じますし，状態の変化は確率的に変わるのであれば幅 τ で，系統的に変わるのであればそれをモデリングしてやれば良いことになります。

この発想に基づいて，設計図を書いてみましょう。ここでは体重の変化に特別な傾向を考えず，ただ幅 τ で確率的にノイズが加わる**ホワイトノイズモデル (white-noise model)** で考えてみることにします。少しイメージしにくいかもしれませんが，たとえば次のように表現できます (図 8.2)。

それではこれをコードにしてみましょう。少し長くなりましたが，ホワイトノイズのモデルはコード 8.1 のようになります。

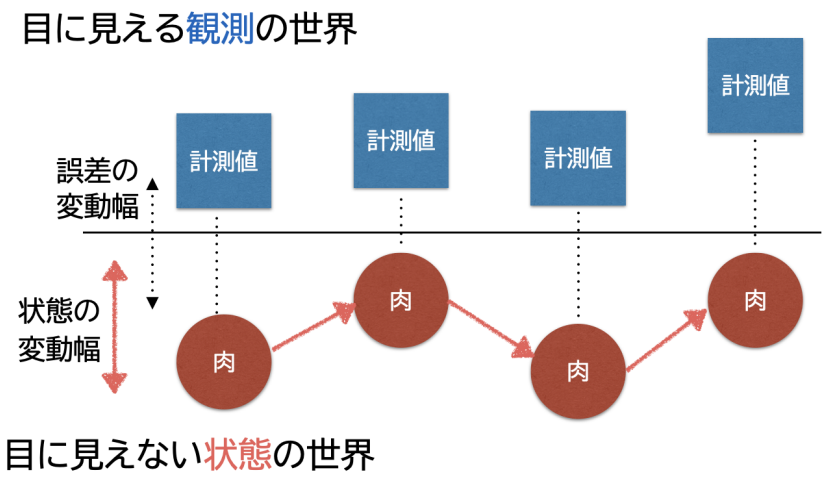

図 8.1 状態空間モデルのイメージ (体重変化の例)

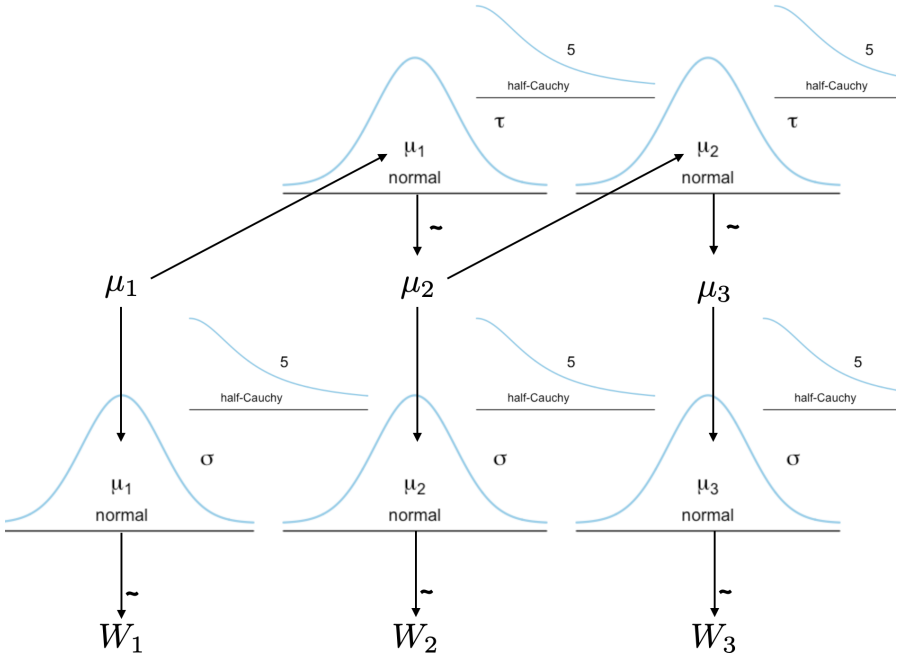

図 8.2 状態空間モデルの設計図

code : 8.1 ホワイトノイズモデル

```
1  data{
2   int L;
3   array[L] real W;
4  }
5
6  parameters{
7    real muZero;
8    array[L] real mu;
```

```
 9    real<lower=0> sig;
10    real<lower=0> tau;
11  }
12
13  model{
14    mu[1] ~ normal(muZero, tau);
15
16    for(l in 1:L){
17      W[l] ~ normal(mu[l],sig);
18    }
19
20    for(i in 2:L){
21      mu[i] ~ normal(mu[i-1],tau);
22    }
23
24    muZero ~ normal(80,10);
25    sig ~ cauchy(0,5);
26    tau ~ cauchy(0,5);
27  }
```

■コード解説

data ブロック　データ長 L, 各時点の体重 W がデータになります。

parameters ブロック　データ長と同じだけの状態を表す mu と, 状態の変動幅 tau, 状態に付加される計測誤差の変動幅 sig を宣言しています。それに加えて, 最初の状態 μ_0 をパラメータとしました。これは 2 時点目の状態 μ_2 は $\mu_2 \sim N(\mu_1, \tau)$ で, 3 時点目は $\mu_3 \sim N(\mu_2, \tau)$ で･･･と表現できるのに対し, 最初の点は $\mu_1 \sim N(\mu_0, \tau)$ となって, 計測される前の状態を考えなければならないからです。わからないものは確率で表現する, というベイズの流儀に則って, これはパラメータとして推定してやることにします。

model ブロック　まず 1 時点目の状態は, 0 時点目の状態から出てくるものですので, $\mu_1 \sim N(\mu_0, \tau)$ をモデル化しました。次に計測された値 W_l は状態 μ_l から出てくるものですから, 1 時点目からデータ長 L まで順に $W_l \sim N(\mu_l, \tau)$ として尤度を書きます。次に状態のモデルですが, 2 時点目以降は前の時点からの影響で表現できるので, $\mu_i \sim N(\mu_{i-1}, \tau)$ としています。最後に事前分布として, これまでの経験から $\mu_0 \sim N(80, 10)$ とし, 変化の幅はいずれも SD ですから半コーシー分布で表現しました。

これを使って, 体重のデータを分析してみましょう。データは 2020 年からのものを使います。図 8.3 をみると, 2021 年まではあまり変化がなく, 21 年初頭に少し体重が下がって, また上がって, その後 5 月から謎の下降[*3], そしてその反動[*4], とおそらく線形モデルでは表現できないような複雑な動きをしています。

さて, コード 8.2 を実行し, 図 8.4 のような結果を得ます。複雑な動きについても, モデルがしっかりと予測しているのがみて取れますね。状態空間のパワフルな表現力をお楽しみいただけましたでしょうか。

code : 8.2　状態空間モデルのコード

```
1  dat1 <- dat %>%
2    filter(date > "2020/01/01") %>%
```

[*3] 自転車事故による鎖骨骨折と, それに伴ってお箸持ちにくくて食事量が減る, 運動も減るといったのが原因でしょう。
[*4] 悔しいです。

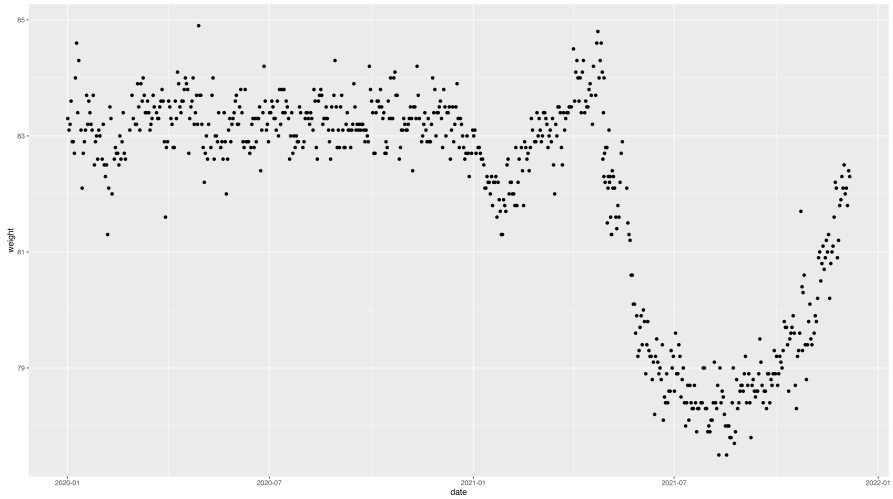

図 8.3 2020 年の体重変化

```
  mutate(date = as.Date(date))

model <- cmdstan_model("StateSpace.stan")
dataSet <- list(L = NROW(dat1), W = dat1$weight)
fit1 <- model$sample(
  data = dataSet,
  chains = 4,
  parallel_chains = 4
)

fit1.stanfit <- fit1$output_files() %>% rstan::read_stan_csv()
fit1.df <- fit1.stanfit %>% as.data.frame() %>%
  as_tibble() %>%
  rowid_to_column("iter") %>%
  pivot_longer(-iter, names_to = "Varname") %>%
  group_by(Varname) %>%
  summarise(
    EAP = mean(value),
    MED = median(value),
    MAP = map_estimation(value),
    SD = sd(value),
    L95 = quantile(value, probs = 0.025),
    L50 = quantile(value, probs = 0.25),
    U50 = quantile(value, probs = 0.75),
    U95 = quantile(value, probs = 0.975)
  )

Est1 <- fit1.df %>%
  dplyr::filter(str_detect(Varname, "mu")) %>%
  dplyr::mutate(ID = str_extract(Varname, pattern = "\\d+") %>%
  as.numeric()) %>%
  arrange(ID)

```

```
36  g <- dat1 %>%
37    rowid_to_column("ID") %>%
38    left_join(Est1, by = "ID") %>%
39    ggplot(aes(x = ID, y = weight, ymin = U95, ymax = L95)) +
40    geom_point() +
41    geom_point(aes(x = ID, y = MAP), color = palette()[2]) +
42    geom_ribbon(fill = palette()[3], alpha = 0.2)
43  plot(g)
```

■コード解説

1-3 行目　データファイルから該当する日程だけ抜き出します。日付を Date 形式に変換しています。

5-11 行目　cmdstanr でコンパイルしてサンプリングするところです。

13 行目　cmdstanr で作ったファイルを stanfit オブジェクトに変換しています。rstan パッケージを使っている人はこの行を実行する必要がありません。

14-28 行目　MCMC サンプルを集計し，**EAP**，**MED**，**MAP** 推定値，SD, 確信区間などを出しています。

30-34 行目　推定された状態 μ_i を取り出しています。

36-43 行目　もとのデータと推定値を合体させ, プロットしています。

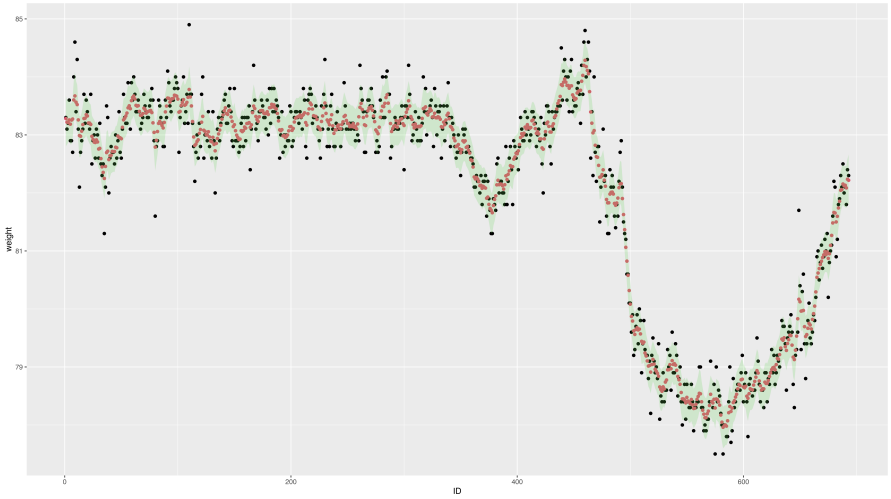

図 8.4　2020 年の体重変化をモデルで追跡してみた結果

8.3　欠損値の補間

ところで, データをよくみてみると, 実は毎日のデータになっていないところがあることに気づきます。いや, 気づかないかもしれないですね。次のコード 8.3 を実行してみてください。

code : 8.3　日付は連続かな?

```
1  dat1 %>%
2    mutate(lag = lag(date)) %>%
3    mutate(date = as.Date(date), lag = as.Date(lag)) %>%
```

```
4    mutate(FLG = date - lag) %>%
5    dplyr::filter(FLG > 1)
```

■コード解説

1 行目 先ほどの推定に使ったデータファイルです。

2 行目 lag 関数で，データを 1 行ずらした列を作ります。

3 行目 体重を記録した日の変数 date と，先ほど作った一行ずらした変数 lag はいずれも日付に関する変数ですので，日付型に変更します。

4 行目 日付変数の引き算をして，一行下のデータと何日ずれていたのか算出しています。

5 行目 1 日以上ずれている日をフィルタリングで抜き出しています。

これをみると，ちょくちょく抜けていて，時には 4 日も空いていたりすることがわかります[*5]。ともかく，これでは正確にデータを分析できているとは言えません。

R の出力 8.1: 抜けている日

```
# A tibble: 21 × 5
   date       weight bodyFat lag        FLG
   <date>      <dbl>   <dbl> <date>     <drtn>
 1 2020-01-13   83.1    26.8 2020-01-11 2 days
 2 2020-02-01   82.6    26.7 2020-01-30 2 days
 3 2020-02-12   82.6    26.7 2020-02-10 2 days
 4 2020-02-26   83.1    26.8 2020-02-22 4 days
 5 2020-02-28   83.7    27   2020-02-26 2 days
 6 2020-03-02   83.2    26.9 2020-02-29 2 days
 7 2020-03-21   83.4    26.9 2020-03-19 2 days
 8 2020-05-09   82.8    26.8 2020-05-07 2 days
 9 2020-05-18   82.8    26.7 2020-05-16 2 days
10 2020-07-31   82.8    27.7 2020-07-29 2 days
# … with 11 more rows
```

しかし計測はできていなくても，出張中にも状態の変化は続きいているはずで，次の計測時点はその日の状態の関数になっているはずです (図 8.5)。

この欠損した状態のデータをどのように考えれば良いでしょうか。実はこのことのヒントはすでに，最初のコードの中に入っています。状態空間モデルの 1 時点目，μ_1 は μ_0 から影響されているはずですが，データから μ_0 はわかりません。しかしわからないことは確率で表現するのがベイジアンのやり方です。今回も，欠損している測定値はわからないものとして確率で表現し，推定してやれば良いのです！このようにして間を埋めることをとくに**補間 (interpolating)** と言います。

補間をするためには，R コードの方でも Stan コードの方でもすこし工夫が必要です。まず R のほうで，データを加工するところから見てみましょう。Stan には欠損値 NA を与えることはできませんから，欠損しているところには特別なあり得ない数字，そうですね，999 という数字でも入れておきましょう。

code : 8.4 連続したデータを作り欠損値に借りの値を代入する

[*5] 2020 年 2 月 22 日から 24 日は岡山県にベイズ塾合宿 (出張) でした。コロナがまだ豪華客船の中だけに抑え込まれている時期で，ギリギリ出張できた時期です。このあと大学の卒業式がなくなったり，前期オンライン授業になったりと言う，波乱の 2020 年度が始まったのでした。

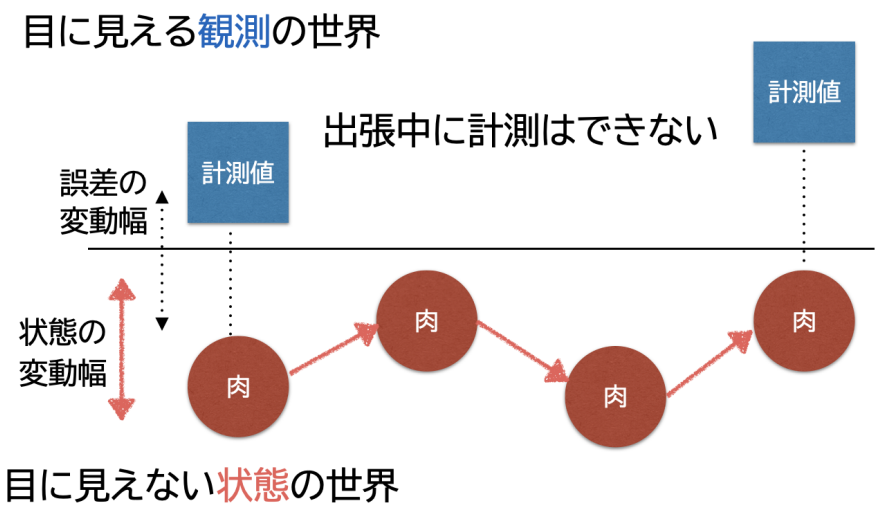

図 8.5　計測できない時点も変化は続く

```
1  fullDays <-
2    data.frame(date = as.Date("2020/01/01"):as.Date("2021/12/01")) %>%
3    mutate(date = as.Date(date, origin = "1970-01-01")) %>%
4    left_join(dat1, by = "date") %>%
5    tidyr::replace_na(list(weight = 999, bodyFat = 999))
```

■コード解説

2 行目　2020 年 1 月 1 日から 2021 年 12 月 1 日までにデータを限定します。この期間，連続した日付を date 変数に追加します。

3 行目　日付変数を日付型に変更しています。

4 行目　もとのデータを日付変数に結合しています。left_join 関数は左にもとのデータセットを置き，右から dat1 を引っ付けていきます。結合する際は，変数 date をキーにし，キー変数が同じものは同じ行に合わせるという湯やり方です。このやり方だと，該当する行がない場合 (計測データセットの日付がない場合)，もとのデータセットは残して，体重や体脂肪変数は NA になります。

5 行目　replace_na 関数で，欠損のところに 999 という数字を入れています。

このようにすることで，次のようなデータセットができました (出力 8.2)。

R の出力 8.2: 抜けている日を無くした完全データセット

```
> fullDays
          date weight bodyFat
1   2020-01-01   83.3   26.70
2   2020-01-02   83.1   26.80
3   2020-01-03   83.2   26.90
4   2020-01-04   83.6   27.00
5   2020-01-05   82.9   28.70
6   2020-01-06   82.9   26.80
7   2020-01-07   82.7   26.70
8   2020-01-08   84.0   27.10
9   2020-01-09   84.6   27.30
10  2020-01-10   83.4   26.90
11  2020-01-11   84.3   27.20
12  2020-01-12  999.0  999.00
13  2020-01-13   83.1   26.80
14  2020-01-14   82.1   65.00
15  2020-01-15   82.7   26.70
```

これをみると，2020 年 1 月 12 日は欠損だったのですが，データ上は 999 という数字が入って完全データのように見えます[*6]。このデータを Stan に与えてやり，体重が 999 というあり得ない数字の場合は別の処理をする，という 0 過剰ポアソンの技術を応用します (セクション??，Pp.??)。

code : 8.5 欠損値補間のコード

```
1  data{
2   int L;
3   array[L] real W;
4   int<lower=0> Nmiss;
5  }
6
7  parameters{
8    real muZero;
9    array[L] real mu;
10   array[Nmiss] real<lower=0> Miss_W;
11   real<lower=0> sig;
12   real<lower=0> tau;
13 }
14
15 model{
16   mu[1] ~ normal(muZero, tau);
17
18   {
19     int j = 0;
20     for(l in 1:L){
21       if( W[l] != 999){
22           // こっちは尤度
23           W[l] ~ normal(mu[l],sig);
```

[*6] ちなみに 2020/01/11 から 12 にかけては，犬会という研究会があったため関西に出張していました。宿に泊まったので，12 日の朝のルーティンができなかったのです。

```
24        }else{
25          j = j + 1;
26          // こっちはパラメータ
27          Miss_W[j] ~ normal(mu[l], sig);
28        }
29      }
30    }
31
32    for(i in 2:L){
33      mu[i] ~ normal(mu[i-1],tau);
34    }
35
36    muZero ~ normal(80,10);
37    sig ~ cauchy(0,5);
38    tau ~ cauchy(0,5);
39  }
```

data **ブロック**　データ長 L, 各時点の体重 W がデータです。加えて, 推定すべき欠損値の数もデータとして受け取ります。

parameters **ブロック**　データ長と同じだけの状態を表す `mu` と, 状態の変動幅 `tau`, 状態に付加される計測誤差の変動幅 `sig` を宣言しています。また最初の状態 μ_0 を表すパラメタと, 欠損の数だけある推定用パラメータ `Miss_W` を用意しました。

model **ブロック**　先ほど同様, 0 時点目の状態から出てくるものですので, $\mu_1 \sim N(\mu_0, \tau)$ をモデル化しています。次に各回のデータについて, W_l が 999 でなければ (同じでない, という論理式は`!=`で表現します), 普通の状態空間モデルのように尤度として計算します。そうではない, すなわちデータ $W_l = 999$ である, つまり欠損値であるはずということなら, $W_{miss} \sim N(\mu_l, \sigma)$ で推定してやります。ここで特殊な変数 j が出てきています。これは「何番目の欠損値か」を表す変数です。ブロックの中の一部だけで変数を宣言するため, まず `for` 文全体を中括弧で括り, はじめに $j = 0$ という数字を宣言しています。ここで欠損値推定のシーンになると, j のカウターを 1 つ増加させ, j 番目の欠損値の推定をさせる, というやり方をしています。欠損値の数と行番号が合致しないので, こうした工夫が必要なのですね。その後のコードは先ほどと同じです。

このようにして, 推定してみましょう。Stan に与えるデータセットは次のようにして作ります。

code : 8.6　Stan に与えるデータセット

```
1  dataSet <- list(L = NROW(fullDays), W = fullDays$weight,
2                  Nmiss = sum(fullDays$weight == 999))
```

欠損値の数は, オブジェクト `fullDays` の `weight` 変数が 999 かどうかを判定させ (==でイコールかどうかという論理判断になります), 条件が合致した数を数え上げるという方法をとっています。

コードが走り出すと, 推定は順調に進むと思います。が, 結果の出方が少しややこしいです。状態 μ_i はすべての日程について推定されますが, これに加えて欠損があったデータも推定されていきますので, 描画する際は「実測値なのか推定値なのか」,「補間した欠損の推定値は何番目の欠損値だったか」を判断しながら結合していき, データにする必要があるからです。Stan の中で作ったカウント変数 j を R の方でも考えてやらないといけないわけですね。

code : 8.7　プロット用にデータを整形する関数とプロットのコード

```r
Est2 <- fit2.df %>%
  dplyr::filter(str_detect(Varname, "mu")) %>%
  dplyr::filter(!str_detect(Varname, "muZero")) %>%
  dplyr::mutate(ID = str_extract(Varname, pattern = "\\d+") %>%
  as.numeric()) %>%
  arrange(ID) %>%
  select(ID, MAP, U95, L95)

Est2miss <- fit2.df %>%
  dplyr::filter(str_detect(Varname, "Miss_W")) %>%
  dplyr::mutate(ID = str_extract(Varname, pattern = "\\d+") %>%
  as.numeric()) %>%
  arrange(ID) %>%
  select(ID, MAP, U95, L95)

### plot用の関数を準備
plotFunction <- function(fullDays, Est, MissEst) {
  tmp <- fullDays %>%
    rowid_to_column("ID") %>%
    left_join(Est, by = "ID") %>%
    rowwise() %>%
    mutate(FLG = if (weight != 999) {1} else {2})
  misJ <- 1
  tmp$weight2 <- NA
  tmp$weight2U <- NA
  tmp$weight2L <- NA
  for (i in 1:NROW(tmp)) {
    if (tmp$FLG[i] == 2) {
      tmp$weight2[i] <- MissEst$MAP[misJ]
      tmp$weight2U[i] <- MissEst$U95[misJ]
      tmp$weight2L[i] <- MissEst$L95[misJ]
      misJ <- misJ + 1
    } else {
      tmp$weight2[i] <- tmp$weight[i]
      tmp$weight2U[i] <- tmp$weight[i]
      tmp$weight2L[i] <- tmp$weight[i]
    }
  }
  return(tmp)
}

plot.tmp <- plotFunction(fullDays, Est2, Est2miss)
g <- ggplot(data = plot.tmp) +
  geom_point(aes(x = date, y = weight2)) +
  geom_errorbar(aes(x = date, y = weight2, ymin = weight2L,
                    ymax = weight2U, color = palette()[2])) +
  geom_point(aes(x = date, y = MAP, color = palette()[3])) +
  geom_errorbar(aes(x = date, y = MAP, ymin = L95,
                    ymax = U95, color = palette()[4])) +
  scale_x_date(date_breaks = "1␣month",
               limits = as.Date(c("2020-01-01", "2020-05-01"))) +
```

```
52    theme(legend.position = "none")
```

■プログラム解説

Est2 を作るブロック MCMC サンプルをデータフレーム化したものの中から，状態 μ に関するデータだけを抜き出したものを作る。

Est2miss を作るブロック 欠損値を補間するための推定値を，MCMC サンプルから取り出した欠損値補間だけのデータセットを作る。

plot 用の関数 プロットはこの後にも行いますので，もう関数を作ってまとめてしまうことにします。もとのデータセット `fullDays` と，状態の推定値データセット `Est`，欠損値の推定値データセット `MissEst` を引数にとる関数です。

関数 3 行目 まずもとデータに行番号を `ID` として変数化します。

関数 4 行目 状態の推定値は，すべての行について推定しているはずですので，行番号をキーに結合 `left_join` します。

関数 5 行目 もとの観測値が実測値なのか欠損値なのかを表現する変数 `FLG` を用意しておきます。これらはすべて，一時的なオブジェクト `tmp` に納められます。

関数 6 行目 欠損値のインデックスを表す変数 `misJ` をつくり，カウントを 1 に設定しておきます。

関数 7-9 行目 先ほどの一次オブジェクトにいまから体重の推定値を入れていきますが，この推定値は実測値で得られている時は不要で `NA` になるはずですので，いったんすべてに `NA` を代入しています。

関数 10-21 行目 オブジェクト `tmp` を一行ずつ見ていって，もし `FLG` 変数が欠損値であることを教えてくれたら，`misJ` 番目の推定値を代入します。代入が終わったらカウンタを 1 つ追加しておきます。欠損値でなければ，もとの体重をそのまま移し入れます。ちなみに U や L がついているのは 95% 区間の上限と下限なのですが，実測値の場合は推定ではありませんのですべて同じ数字になります。

関数 22 行目 作ったオブジェクト `tmp` を戻り値として返します。

関数の結果を受けとる `plot.tmp` は描画用の一時データオブジェクトです。ここに全期間のデータと，今回の状態推定値，欠損推定値を入れておきます。

描画する x 軸は全日程です。まずは体重 W_j を `geom_point` と `geom_errorbar` でプロットします。欠損の場合は推定値が入っています。次のポイントとエラーバーの geom は，状態 μ のものになります。

さあどうでしょう。コードはともかく，図 8.5 を見てみましょう。実際に観測しているのは黒点，状態の推定値は緑点と赤いエラーバーで表示されています。青いバーは欠損した時の推定値で，値がないのですから当然触れ幅は大きいですが，おそらくこの体重が測定できなかった日はこれぐらいだったんじゃないか，という値が推定されているわけです。数字で見ると，たとえば 2020 年 1 月 12 日の MAP 推定値は 83.4kg，95% 区間でいうなら [82.6,84.2] です。前日が 84.3kg で翌日が 83.1kg ですから，そんなもんかなあという気もします。

ところで，測定していないはずの日も補間できるよ，ということを考えてみると，これはすごいことだと思いませんか。そうです，未来の値ももちろん「測定していない日」ですから，このモデルを使うと未来予測ができるのです！すごい！

早速同じコードで，日程を未来に飛ばしてみましょう。データは 2021 年 12 月 06 日までしかありません[*7]

[*7] この原稿は 2021 年 12 月 10 日 17 時に執筆しています。

図 8.6 欠損していても推定できる

から，2021 年 12 月 31 日までデータを伸ばし，推定してみようではありませんか。コードは省略しますが，実行した結果は次のような図 8.7 として得られます。

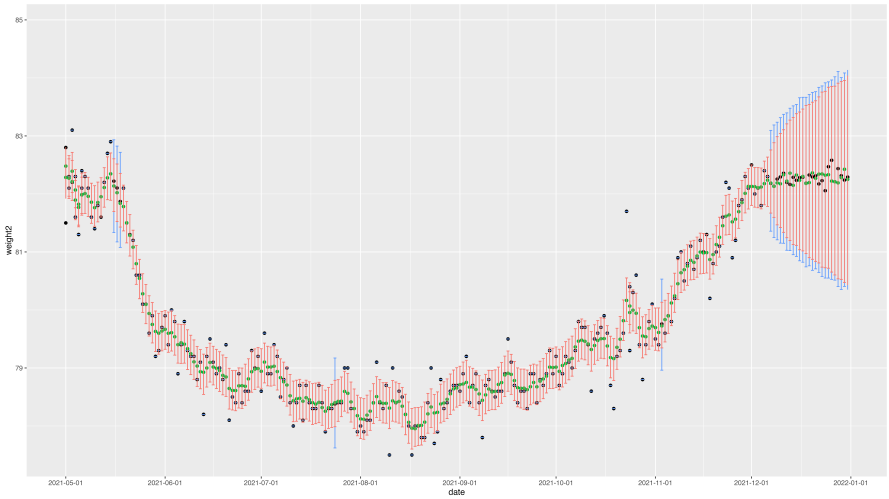

図 8.7 未来もある意味欠損値

これをみると，データが得られなくなった次の日から，確信区間が日を追うごとに広がっていく様がみて取れます。当然そうですね。μ_{t+1} は μ_t にプラスかマイナスか，いずれとも言えない振幅幅で揺れ動くのですから，先に行けば行くほどその可能性は不確かなものになっていくのです。ちなみに描画用のデータから 2021 年最後の数日間の予測値を見てみると，次のようになっています。

code : 8.8 予想される 2021 年末の体重 (一部略)

```
1  > plot.tmp %>% tail
2  # A tibble: 6 × 11
3  # Rowwise:
4     ID date       weight bodyFat   MAP   U95   L95   FLG weight2
5    <dbl> <date>    <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
```

```
 6    738 2021-12-26    999      NA  82.2  83.8  80.5     2    82.1
 7    739 2021-12-27    999      NA  82.3  83.9  80.4     2    82.1
 8    740 2021-12-28    999      NA  82.2  83.9  80.4     2    82.2
 9    741 2021-12-29    999      NA  82.2  84.0  80.4     2    82.0
10    742 2021-12-30    999      NA  82.3  84.0  80.3     2    82.3
11    743 2021-12-31    999      NA  82.3  84.0  80.3     2    82.2d
```

未来の状態 μ は 82.2 のあたりでほぼ固定，それに伴って想定される体重 weight2 も 82.2 程度ですが，幅がどんどんと広くなっていってます。可能性として，80kg に落ちているかもしれませんし，84kg になっているかもしれない，というのが現時点での予測になります。

予測ができるとは言え，なんだかほとんど変わり映えしない数字で面白くないですね。それは当然，「明日の体重は今日の体重 + 誤差だろう」という，とくに何らメカニズムや仮定を入れなかったものですから，ほぼ横一線の推定にしかならないわけです。

では少し欲張って，構造を入れみましょうか。ここまでは，μ_t は μ_{t-1} にのみ影響される，というモデルになっていましたが，もう 1 日前の影響が入っていることを考えます。たとえば体重が増加傾向にあるとか，減少傾向にあるといった，変化を考えてみることにしましょう。すなわち，$\mu_t - \mu_{t-1} = \mu_{t-1} - \mu_{t-2} + \varepsilon$ です。これは昨日から今日への変化 $(\mu_t - \mu_{t-1})$ は，一昨日から昨日への変化 $(\mu_{t-1} - \mu_{t-2})$ と同じようなものだろう (誤差 ε はあるけど)，と考えていることになります。このようなモデルを **2 階差分のトレンド**と呼びます。

この式を変形すると，

$$\begin{aligned}\mu_t &= \mu_{t-1} + \mu_{t-1} - \mu_{t-2} + \varepsilon \\ &= 2\mu_{t-1} - \mu_{t-2} + \varepsilon\end{aligned}$$

となることはすぐにわかりますね。これを確率モデルで表現するなら，

$$\mu_t \sim N(2\mu_{t-1} - \mu_{t-2}, \tau)$$

と考えていることと同じですから，そのように Stan のコードも変形します (コード 8.9)。

code : 8.9 2 階差分トレンドのコード

```
1  ...(前略)...
2  model{
3    mu[1] ~ normal(muZero, tau);
4    mu[2] ~ normal(mu[1] , tau);
5
6    {
7      int j = 0;
8      for(l in 1:L){
9        if( W[l] != 999){
10           // こっちは尤度
11           W[l] ~ normal(mu[l],sig);
12       }else{
13         j = j + 1;
14         // こっちはパラメータ
15         Miss_W[j] ~ normal(2*mu[l-1]-mu[l-2], sig);
16       }
17     }
18   }
19
20   for(i in 3:L){
```

```
21      //2階差分
22      mu[i] ~ normal(2*mu[i-1]-mu[i-2],tau);
23    }
24
25    muZero ~ normal(80,10);
26    sig ~ cauchy(0,5);
27    tau ~ cauchy(0,5);
28  }
```

コードは 2 階差分になっているので，状態は 3 時点目からしか推定できません。最初の 2 行で μ_1, μ_2 をメカニズムに沿って推定し，変化が取れるようになってから，は 2 階差分のコードで推定します。for 文が 3 から始まっていることに注意してください。

このコードと年末までのデータ，そしてプロット関数を使って推定した結果が次の図 8.8 になります。

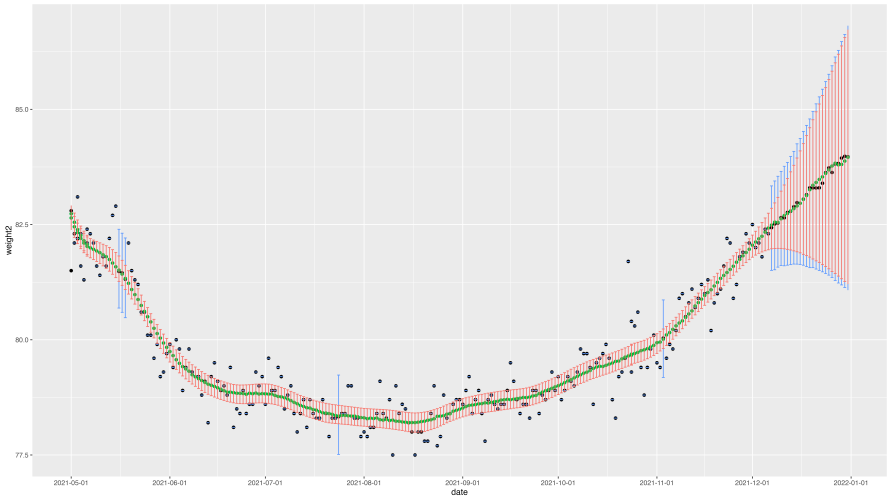

図 8.8　2 階差分トレンドの予測

これを見ると，それまでの増加傾向を反映して，年末には 84kg，最悪の場合 86.6kg まで増えている可能性があることがわかります[*8][*9]。

code：8.10　2 階差分トレンドで予想される 2021 年末の体重 (一部略)

```
1  > plot.tmp %>% tail()
2  # A tibble: 6 × 11
3  # Rowwise:
4     ID date       weight bodyFat   MAP   U95   L95   FLG weight2
5  <dbl> <date>     <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
6    738 2021-12-26    999      NA  83.9  85.6  81.5     2    83.6
7    739 2021-12-27    999      NA  83.9  85.8  81.5     2    83.8
8    740 2021-12-28    999      NA  84.0  85.9  81.4     2    83.8
9    741 2021-12-29    999      NA  84.1  86.1  81.4     2    83.9
10   742 2021-12-30    999      NA  84.2  86.3  81.3     2    84.1
```

[*8] なんてこった，まだ執筆中は答えが出ていませんが，この予測通りになったらとんでもないことですよ。だって私は 20 代 66kg だったんです。あの頃から 20kg も増えている自分の未来なんて想像したくないですよ。またダイエット始めるかなぁ。

[*9] 2021/01/15 追記。2021/12/31,6:07AM の記録は 82.3kg，体脂肪率 26.6% でした。ちゃんと確信区間の中に入ってますね！

```
11    743 2021-12-31    999       NA  84.2  86.5  81.2     2    84.0
```

今回は 2 階差分のトレンドを入れての予測となりましたが，この他にも周期的に影響してくる季節項をいれるとか，たとえば「週末は体重が増加するだろう」と考えて週末効果の項を入れるとか，さまざまな工夫を思いつくことができるのではないでしょうか。

8.4　状態空間モデルの展開

最後になりましたが，状態空間モデルは表面に現れている値の背後に潜在変数を仮定し，それを滑らかに繋ぎ合わせた**スムージング (smoothing)** の技術であるとも考えることができます。時系列を通じて，その前後の「状態」を，柔らかなバネで結合し，データ全体を通じて変化する関数として相互に調整させあっている，と考えることもできるでしょう。

また，時間というのは過去から未来へ進む，1 次元の道のりです。ところで私たちは 2 次元の地図を見たり三次元の空間を見たりしていますよね。たとえばこの状態空間が，時系列的一次元ではなく，X 軸と Y 軸に広がる網のような二次元結合していることを考えると，空間的な統計分析が可能になります。たとえば地価というのは，駅からの距離や大型商業施設，コンビニエンスストアなどの利便性に応じて変化しますが，おそらくそれらは連綿と繋がっていて，徐々に変化していくものであるはずです。であれば徐々に変化する要素をつなぎ合わせて，面の状態空間モデルを作ってやることもできるわけです。さらにその地価の変動を考える，つまり二次元平面 × 時系列に潜在変数をつなげてスムージングしたモデルを作れば，時系列的な地価の変動を考えたり予測したりできるかもしれません。

心理学はこれまで，調査法で一時点を切り出してその構造を把握する，というアプローチが多くありました。時系列を扱うとしても，プレ・ポストの変化を見るとか，あまり長期にわたって追いかけるようなことはしてきませんでした。しかし冒頭でお話ししたように，これからの心理学データは時間的にも空間的にも広がるものを計測し，利用できるようになるかもしれません。そのためにはより深い人間の観察や推察によるモデルの作り込みと，状態空間モデルとベイズ推定のような数理モデルと強力な推定法のタッグが，必要になってくるでしょう。

8.5　課題

体重変化のデータには，体脂肪の記録もあります。体重 × 体脂肪率から，筆者の筋肉量を計算できます。2020 年以降のデータを使って，筋肉量の変化を状態空間モデルで推定してみてください。2 階差分トレンドモデルで，未来のデータも予測できればなお結構です。これらのモデルを分析する R/Stan コードを提出してください。結果の解釈などを，スクリプトのコメントアウトや別添ファイルなどで提供してもらえるとなお良いです。もちろん Rmd ファイルでの提出であれば完璧です。なお提出されたコード単体でバグがなく動くことが確認できないものは，未提出扱いになります。コードの書き方などわからないところがあれば，曜日別 TA か小杉までメールで連絡し，指導を受けてください。

付録 A

よくある質問とミスの例

A.1 Frequently Miss and Comments

Rmd でレポートを提出したのに，なんだか中身の問題じゃないのに突き返された，中身を見てくれよ！と思う人もいるかもしれません。テキストでは R や Rmd での課題を提出するよう求めているところがありますが，その際よく見られる学生さんのミスとその対応についてのコメントをここにまとめておきます。

A.1.1 FMC1；そもそもファイルの書式が違う

Rmd で提出してください，R で提出してください，という指示に対して，違うものが提出されてくることがあります。書式があっていない，というのは些細なことのように思えるかもしれませんが，学術論文は書式に捉われず内容に集中するためにも，書式は整えられたものである必要があります。学会誌に掲載されている論文も，みなさんが書く卒業論文も，レポートに至るまで，書式や指示に沿ったものを準備する必要があります。書式があっていない場合は，門前払いになっても文句が言えないのがアカデミックの世界です。

ということですので，R や Rmd で提出してください，という指示があれば，R や Rmd で提出してください。ファイルの種類は拡張子で分類され[*1]，R ファイルは `.R`，Rmd ファイルは `.Rmd` という拡張子になっています。たまに `.Rproj` というファイルを提出してくる人がいますが，これは R プロジェクトのファイルで，これには R スクリプトも文書も含まれておらず，みなさんの計算環境情報が少し書いてあるだけです。また，`.Rmd` だけ入っていれば良いかというとそうではありません。まれに `Filename.Rmd.R` というようなファイルを送ってくる人がいます。これはファイル名にピリオドが含まれているだけで，ファイルの種類を識別する拡張子は `.R` ですから Rmd ファイルではありません[*2]。そもそもファイル名には 2 バイト文字はもちろん，!や?などの記号を含めるべきではありません。ピリオドも当然記号の一種ですから，ファイル名にするのは不適切です[*3]。

ではどうやって適切なファイル形式にするか，ということですが，最も素直な方法としては RStudio で新しくファイルを開くときに，R markdown 形式 (略して Rmd) を選ぶようにしましょう。もし間違って，R script 形式 (`.R` 形式) で開いてしまった，というときは，そのファイルを破棄して新しく作り直すのが一番ですが，エディタペインの右下にあるファイル種類表示 (図 A.1) をクリックして修正することもできます。

[*1] 拡張子についてはセクション C.5,Pp.130 参照

[*2] 最近の OS は拡張子を表示しない設定になっているものも少なくないので，このようなミスが生じます。

[*3] 長いファイル名などの場合，空白を入れるのも適切ではありません。できなくはないのですが，やるべきではないのです。どうしても空白を入れたければ，アンダースコアなどで区切ると良いでしょう。

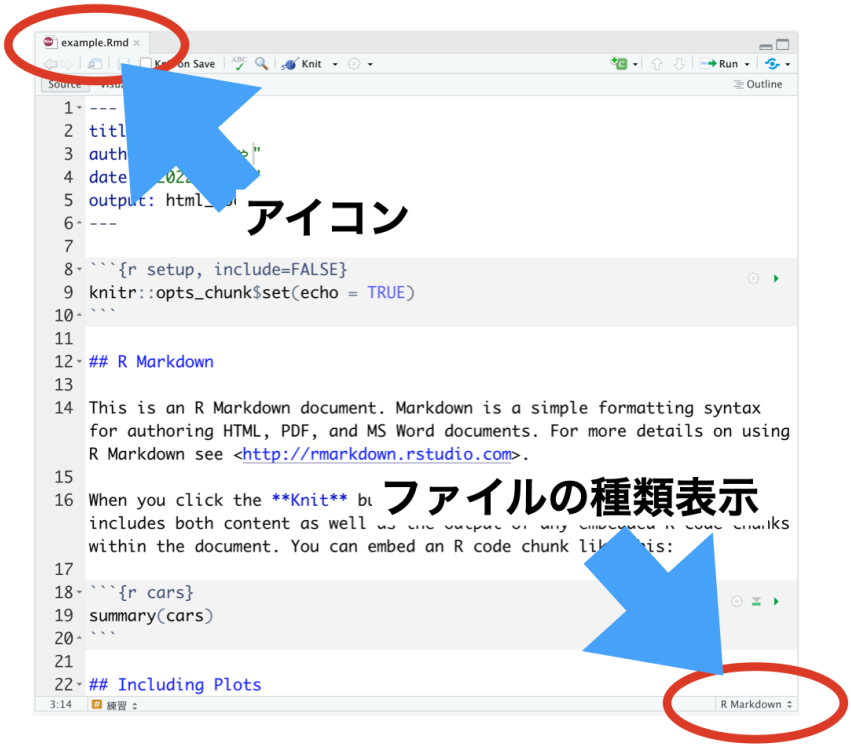

図 A.1 ファイルの種類を判別する方法

A.1.2 FMC2；やっぱりファイルの形式が違う

Rmd 形式のファイルは，拡張子だけで決まるものではありません。図 A.2 にあるように，Rmd ファイルは冒頭の 6 行 (正確には---で囲まれた領域) が YMAL と呼ばれるところで，文書全体の設定をしています。その下に，R のコードを実行する部分 (**チャンク (chunk)**) や文章の領域があります。文章のところは#記号で見出しを作ったりできます。

この YMAL 部分が壊れている，あるいはチャンクが正しく記述されていない場合，拡張子が.Rmd であっても適切な Rmd ファイルにはなっていません。YMAL 部分の書き方はよくわからない，という人も多いと思いますので，RStudio で Rmd ファイルを作ったときの状態をなるべく変更しないように注意すると良いでしょう[*4]。

チャンクは R のコードを書くところで，バッククォーテーション 3 つでくくるのが決まりです。チャンク領域が始まるところに{r}とかいて「ここが R で計算するところですよ」というのを指定するわけです[*5]。このバッククォーテーション 3 つ (```) が全角だったり (｀)，ダブルクォーテーションだったり (") すると，機械は正しくチャンクであると認識しません。フォントなどの見せかけ上は，微妙な違いのように見えますが，機械にとって違う文字列は違う意味を持ちますので注意してください[*6]。RStudio で編集する場合，チャンクの領域はや

[*4] Rmd ファイルを新しく開くときに，文書タイトルや著者名，日付，出力ファイル形式などを設定することができるウィンドウが開きますので，そこに必要な情報を書くと自動的にそれを使った YMAL が生成されます。また，本文として幾つかのサンプルコードが最初から含まれていますが，これらに関してはサンプルをそのまま使うことはないので，全て削除してしまって結構です。

[*5] ほかにも Python や Julia など他の計算言語を混ぜることもできます。

[*6] 記号の名称や入力については，セクション E,Pp.141 を参照してください。

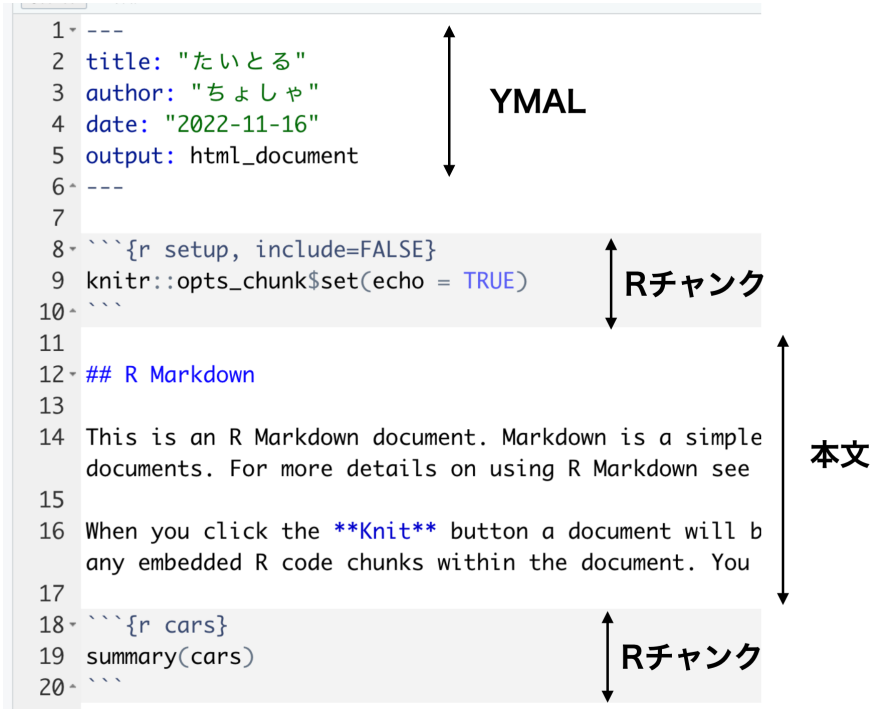

図 A.2　Rmd ファイルの中身における書式

や灰色がかった強調表示がされますので，どこにチャンクがあるかわかると思います。もし強調表示されていないようであれば，チャンクとして認識されていない可能性を疑った方が良いでしょう。

チャンクはバッククォーテーション 3 つで開き，おわったら同じくバッククォーテーション 3 つで領域を閉じます。閉じなければずっと R の計算領域が続くと解釈されますし，最後までチャンクが閉じられないと Rmd ファイルとして正しくない書式ということになります。チャンクが正しく入力できているかについて，常に注意を払っておく必要があります。また，自分でバッククォーテーション 3 つを使ってチャンク領域を開いたり閉じたりするのが面倒だ，という人は RStudio のチャンク挿入ボタンから挿入すると間違いないでしょう。

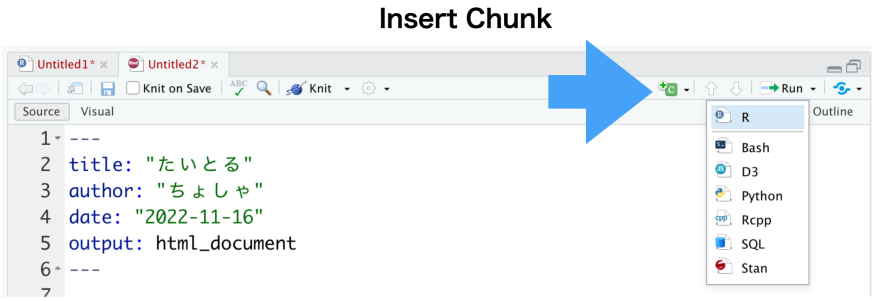

図 A.3　RStudio のチャンク挿入ボタン

A.1.3　FMC3；Rmd が knit できない

Rmd は文書作成に加えて，R での計算がセットになったファイル形式であり，文中の数字や分析結果は「書き写す」のではなく「その場で生成する」ものです。生成する，というのは Rmd ファイルを変換して，

HTML や DOCX，PDF 形式のファイルにすることを指します[*7]。このファイル変換を**ニット (knit)** といいます。編み物を編むようなイメージですね。このときに，タイトルをつけ，見出しのサイズを変え，R の計算をして結果を埋め込む作業をするわけです (図 A.4)。こうすることで，結果のコピペを避けること，再現性を担保することができるようになるわけです。すでに説明したように，チャンクを使って計算に必要な指示を Rmd

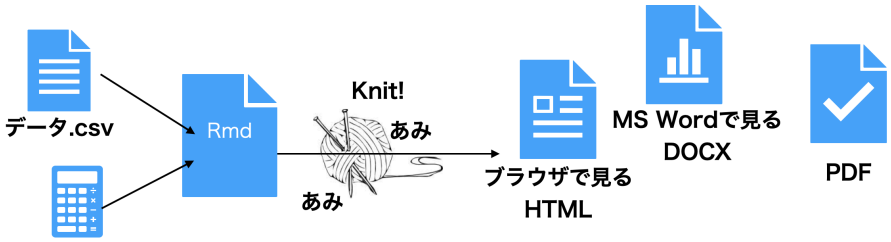

図 A.4 knit してレポートが完成する

ファイルに書きます。この指示はエラーのないコードである必要があります。当然ですが，スペルミスや間違った R コードは「実行できない」というエラーになるわけです。その場合，knit は中断され結果のファイルが出力されません。提出されたレポートは knit して出来上がったもののことを指しますから，knit できない Rmd ファイルを提出されてもレポートが提出されたことにはならないわけです。

knit できない Rmd ファイルになっていないか，というのはご自身の RStudio 環境で knit して確認できることです。提出前に一度，きちんと機能する Rmd ファイルになっているかどうか確認するようにしてください。以下で述べるように，自分の環境で knit できても，提出先の (私の) 環境で knit できないということもあり得ます。しかし，自分の環境で knit できないのに提出先でできる，ということはあり得ませんので，「knit できませんよ」というコメントをつけて返される前に自分で確認するようにしてください。

A.1.4 FMC4；外部環境を参照してしまう

さて，R で行う作業の中には，csv ファイルを読み込むといった「外部のファイルを使う」指示もありえます。例えば次のようなコードです。

code：A.1 Rmd 中の R コードが外部環境を参照してしまう

```
1  dat <- read.csv("social_effects.csv")
2  dat <- read.table("clipboard")
```

このコードでは，上の行は `social_effects.csv` というファイルを，下の行はクリップボードの内容を読み込むようになっています。ファイルを読み込む指示は，ファイルがなければ当然エラーになりますから注意が必要です。レポート等で Rmd ファイルを提出するとき，Rmd ファイルの他に必要な読み込むべきファイルがあれば一緒に提出するようにしてください。また，クリップボードの内容は再現できません。クリップボードとは，コピーアンドペーストのコピーを行ったとき，PC の内部で一時的にコピー内容を覚えておく場所のことです。つまり，みなさんがみなさんの環境で，クリップボード上にデータを一旦保持している場合は，このコードでエラーがしょうじることはありません。しかし，レポート提出先の (私の) 環境で，事前にデータのコピー作業を行なっているわけではないのですから，提出先でエラーが発生します。

[*7] HTML は Hyper Text Markup Language の略で，ブラウザで開くファイル形式です。DOCX は Microsoft Word のファイル形式です。PDF は Portable Document Format の略で，データを紙に印刷した状態のようにサイズを固定して出力したファイル形式であり，OS がもつビューワーや Adobe Acrobat Reader などで見ることができます。

そもそも Rmd ファイルは，作業の再現性を担保するためのファイルになっているわけですから，クリップボードの利用のような「自分の環境だけで可能な記録されない作業」をそのファイルに含めるのは適切ではありません。Rmd ファイルは Rmd ファイルだけで分析作業が完結するように，必要な記録は全て記載されている必要があります。同様に，分析作業に関係のない冗長な指示や無駄な指示は Rmd ファイルに含めるべきではありません。

A.1.5 FMC5；提出先の環境を変更してしまう

R はさまざまなパッケージを使って分析環境を拡張することができます。パッケージは **CRAN** を通じてインターネット経由で配布されますから，ネット環境があれば誰でも最新のパッケージをとってくることができます。みなさんが Rmd ファイルの中で使う R の関数の中には，パッケージの関数も含まれているでしょう。パッケージがなければ関数が動きませんから，パッケージをインストールする作業も Rmd に書いておきたいと思うかもしれません。しかし，これは推奨できません。

パッケージのインストールは，実行環境の準備にあたる作業です。Rmd ファイルを使ってコードのやり取りをするとき，提出先の環境で分析することになりますが，提出先の環境にどのようなパッケージをいつ入れるかは，提出先の環境の管理人が判断すべき問題です。提出する Rmd ファイルにパッケージをインストールする関数，すなわち `install.packages()` 関数が含まれているというのは，相手の環境を勝手に操作してしまうことと同じであり，セキュリティ的にも適切な発想ではありません。

環境の準備は提出先の管理人が管理すべき問題であり，またすでにパッケージが入っている場合は無駄な上書き作業をさせることにもなります。また `install.package` 関数は CRAN のサーバを参照したりしますから，適切な設定がなければ R チャンク実行時にエラーが発生します。いずれにせよ，R チャンクの中に `install.package` 関数を含めないようにしましょう。

以上が Rmd ファイルや R ファイルでレポートを提出するときの留意点です。その他にも R や RStudio を使うときによくある質問がありますので，それらも一問一答型で紹介しておきましょう。

A.2 Frequently Asked Questions；よくある質問と答え

Q. A.2.1: テキストを参考にパッケージをインストールしようとしたところ，エラーが発生しました。

A. 質問ではなくて報告ですね。お返事としては，「わかりました。エラーが発生して大変ですね。」としか言いようがありません。どの環境で，何をしようとして，どのようなエラーが出たのか，明示してください。メールのやり取りで指示が明確になるように，テキストを準備しています。何章，何ページの，どの文章を参考にしたのかも教えてください。

Q. A.2.2: 添付のようなエラーが発生しました (図 A.5)。対応策を教えてください。

A. エラーの発生画面を送ってこられていますが, この画面に写っていない上の方でエラーが発生していますから, これではどのエラーなのかわかりません。スクリーンショットを送ってこられることはよくありますが, ほとんどの場合, 適切な箇所が写っていません。複数枚添付してこられる人もいますが, 画面を拡大しながら読むのも難しいので, **R コードそのものを送ってきてください**。そうすると, 何行目のどこにエラーがあるかが明確になり, 対応に関係ない無駄なやり取り (ex.「もう少し上を写してください」「違う, もっと上」) が減ります。

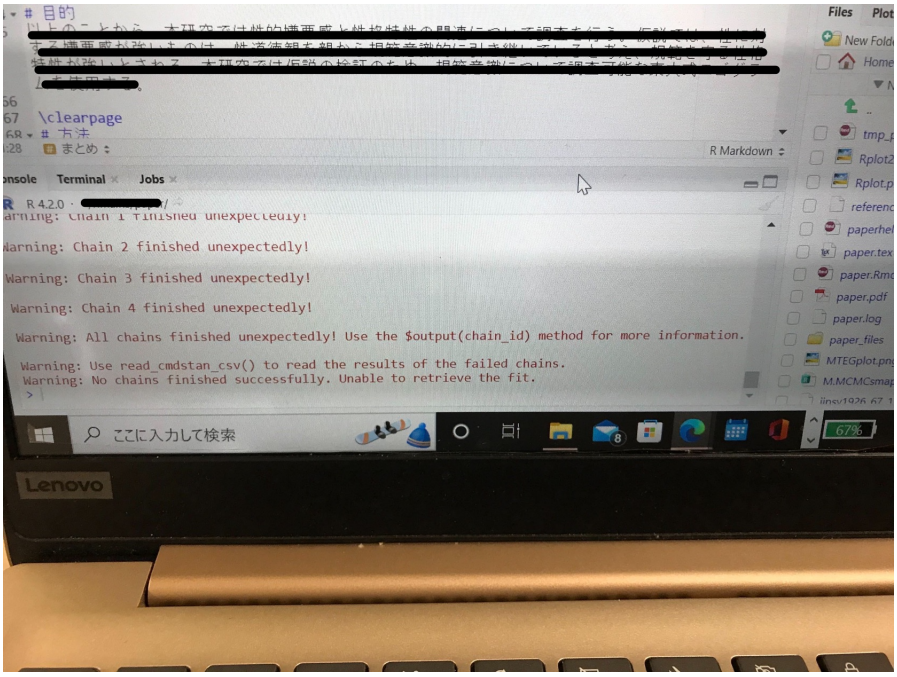

図 A.5 添付されてくるスクリーンショットの一例

Q. A.2.3: ファイルを読み込もうとすると次のようなエラーが出ます。どうすれば良いですか。

`'xxxxx'` に不正なマルチバイト文字があります

A. 文字化けの原因は, たとえば UTF-8 形式で提供されているファイルを, Windows 標準の CP932 形式で読み込もうとする, という文字コードの不一致です。ですからそれをオプションで指定してあげれば問題解決です。`read.csv` 関数を使っている場合, オプションの指定は `read.csv("foo.csv",fileEncoding="UTF-8")` のようにします。`tidyverse` の `read_csv` 関数を使っている場合, locale オプションで, locale 関数の encoding を明示的に UTF-8 とします。`read_csv("foo.csv", locale=locale(encoding = "UTF-8"))` のようにします。

Q. A.2.4: ファイルを読み込もうとすると次のようなエラーが出ます。どうすれば良いですか。

`'xxxxx'` に不正なマルチバイト文字があります

A. 文字化けの原因は，たとえば UTF-8 形式で提供されているファイルを，Windows 標準の CP932 形式で読み込もうとする，という文字コードの不一致です。ですからそれをオプションで指定してあげれば問題解決です。`read.csv` 関数を使っている場合，オプションの指定は `read.csv("foo.csv",fileEncoding="UTF-8")` のようにします。`tidyverse` の `read_csv` 関数を使っている場合，locale オプションで，locale 関数の encoding を明示的に UTF-8 とします。`read_csv("foo.csv", locale=locale(encoding = "UTF-8"))` のようにします。

Q. A.2.5: パッケージを読み込もうとすると次のようなエラーが出ます。どうすれば良いですか。

`library(tidyverse)` でエラー；`'tidyverse'` というパッケージはありません。

A. パッケージがないので，インストールしてください。

Q. A.2.6: パッケージをインストールしようとすると次のようなエラーが出ます。どうすれば良いですか。

```
Warning in install.packages:
    'lib = "C:/Program Files/R/R-4.0.5/library"' is not writable
```

A. ユーザがファイルに書き込みをする権限を持っていないので，(パッケージファイルをドライブに) 書き込みできないというエラーです。RStudio を実行する時に，「管理者として実行」を選びましょう。

Q. A.2.7: パッケージをインストールしようとすると次のようなエラーが出ます。どうすれば良いですか。

```
Warning in install.packages:
    ディレクトリ 'C:〜(任意の文字列)〜\????' を作成できません。理由は
```
```'Invalid argument' です。```

A. ユーザ名が全角文字を含んでいるため，文字化けして R 側から操作ができません。解決する方法は二つあります。

ユーザを作り直す方法 ユーザ名に全角文字を含まない，新しいユーザを作成します。設定＞アカウントから新しいアカウントを作りましょう。

インストールフォルダを指定する R のコンソールで `.libPaths()` と実行すると，パッケージをインストールするフォルダが出てきますが，ここを変更する方法です。次の 2 ステップで対応できます。

- まず C ドライブのすぐ下にインストール先のフォルダを作ります。`myLib` としましょう。
- 次に R のコンソールで `.libPaths("C:/myLib")` 書いて実行します。

これでインストール先が変わりますので，書き込み・インストールができるようになります。老婆心ながら付け加えますと，フォルダ名はなんでも構いませんが，全角文字ではいけません。また場所も C ドライブのすぐ下である必要はありませんが，全角文字や空白を含むフォルダの下に入れてしまってはいけません。OndDrive のようなクラウドを指定するのも良くありません (探しに行った時にオフラインだとまたエラーになります)。
```

Q. A.2.8: ファイルが読み込めません

```
file(filename, "r", encoding = encoding) でエラー:
  コネクションを開くことができません
 追加情報:  警告メッセージ:
file(filename, "r", encoding = encoding) で:
  ファイル 'foo.csv' を開くことができません: No such file or directory
```

A. 指定された場所にファイルがないので，読み込むことができないエラーです。確認すべきは，「プロジェクトを開いているか」，「プロジェクトフォルダの中に当該ファイルはあるか」，「ファイル名のミススペルはないか」です。プロジェクトってなんだという人は，RStudio の基本に立ち戻り，プロジェクトでファイルやフォルダを管理するようにしてください。プロジェクト管理については，Pp.??にもその説明があります。

プロジェクトによる管理とは要するに，R が今見ているフォルダの場所を固定する方法です。プロジェクトを開くと，プロジェクトフォルダが「今見ているフォルダ (ワーキングディレクトリ)」になりますので，その中のファイルを参照することになります。プロジェクトフォルダの中に当該ファイルがないと読み込むことができませんので[a]，当該ファイルをプロジェクトフォルダ内に移してきてください。

Rmarkdown や Quarto ファイルの場合も，必ずプロジェクトフォルダの中に置くようにしてください。これらは Knit するときはファイルの置かれた場所を WorkingDirectory にしますので，プロジェクトフォルダの中にないとパスが通らない問題が頻発します。

[a] フォルダの位置を相対的・絶対的に指定してやれば，どこにおいていても読み込むことはできますが，この問題で悩んでいる人は相対・絶対パスの指定というところでさらに疑問が深まることになると思いますので，気にしないでくれて結構です。相対・絶対パスが知りたい人は，付録 C,133 でファイル場所とは何かを再確認してください)。

Q. A.2.9: ANOVA 君が読み込めません

```
file(filename, "r", encoding = encoding) でエラー:
  コネクションを開くことができません
 追加情報:  警告メッセージ:
file(filename, "r", encoding = encoding) で:
  ファイル 'http:/riseki.php.xdomain.jp/index.php?plugin=attach&refer=
  ANOVA 君&openfile=anovakun_486.txt' を開くことができません: Invalid argument
```

A. ファイルを読み込みにいく先が URL，すなわちインターネット上になっています。ANOVA 君のファイルを一度手元の PC のフォルダにダウンロードし，そのローカルのファイルの位置を指定して読み込むようにしてください。

Q. A.2.10: 効果量として Hedges の g を算出してください。という指示について実行すると g ではなく d が出てしまうようなのですが、どうすればよいでしょうか。コードは次の通りです。

```
cohen.d(value ~ condition, data = dat, hedges.crrection = T)
```

A. Hedges の補正 (correction) のオプションが通っていません。スペルミスです。オプションのスペルが間違っているので無視されたので, 関数名通り Cohen の d が算出されます。

Q. A.2.11: 描画の際に次のような注意が出てきます。これはどういう意味ですか。

```
Removed 671 rows containing missing values (geom_point).
```

A.「671 件の行で欠損値が含まれています」ということです。つまりデータセットの中に欠損値 (観測されていない, 数値が入っていない行) が 671 件あったので, それは表示できませんでしたよ, という意味です。警告が嫌だということであれば, データセットの中で欠けているものを除外する必要があります。R の関数では, `na.omit` で欠損値を除外することができます。

Q. A.2.12: lm 関数を実行するコードでオブジェクトがないと言われます。

```
result <- lm( weight - height, data = dat)
```

A. 従属変数と独立変数とをつなげるのはチルダ (~) という記号です。そこがハイフン (-) になっています。スペルミスの一種です。

Q. A.2.13: lm 関数を実行するコードでオブジェクトがないと言われます。

```
result <- lm( weight - height )
```

A. データを与えていないので, `weight` というオブジェクトを探しに行って, 見つからないというエラーです。`data` オプションでデータを与えてあげてください。

Q. A.2.14: 因子分析の Robust 法での RMSEA の p 値って超えてたらまずいですか。Standard(DWLS) の方だけで報告はだいじょぶでしょうか。

A. 非常に専門的な質問をしておられますが，まず，「因子分析」「Robust 法」「RMSEA」など，個々の用語の意味はわかっているでしょうか。キーワードによる検索で，「ロバスト法とは」「Robust の意味」などは答えが出てくると思いますが，これらを分析法の体系的な文脈の中に位置付けないと，答えられない種類の問題です。この例をもとに説明すると，Robust を辞書で引くと「壮健」「たくましいこと」などと出てきますが，もちろんそういう意味ではありません。ロバスト法を検索すると，「統計学の分野でロバスト推定法というやり方がある」「観測値に外れ値が含まれている可能性を考え，その影響を抑えることを目的とした手法」などの説明が出てきます。ロバスト推定法は因子分析に限らず，回帰分析など他の手法でも使われる考え方なので，このような一般的な解説になります。ですがここで知りたいのは「因子分析におけるロバスト推定」ですから，因子分析でロバスト推定するとはどういうことか，因子分析の観測値における外れ値とは何か，因子分析における推定とは何を推定するのか，そもそも因子分析とは何を目的としているのか，なぜ自分は因子分析方を使うのか，さらに何故因子分析の中のロバスト推定を使いたいと思っているのか，といったことがわかっていないと，この質問に正しく回答することができないのです。このように，複合的な要素について一回で質問しても，適切な答えに辿り着けないことがあります。聞くことは恥ずかしいことではありません。知らないことを知っているふりをすることが恥ずかしいことであり，知らないまま「みんなそうやっているから」「テキスト/ネットに書いてあったから」と看過してしまうことこそ恥ずべきことなのです。ちなみに，質問に対する回答を間違えることも恥ずかしいことではありませんから，「正しく答えられなければ恥をかく (から質問しない)」というのも同様に恥ずべきことです。こうした恥ずかしさ (保身) から，「専門用語を使ってそれっぽく質問してわかった感じになろう」というのは，かえって遠回りになります。実は回答よりも，質問の仕方でその人の理解度が明らかになってしまっています。このように書くと，なんでも反射的に「わかりません，教えてください」という人もいますが，それも適切な質問方法ではありません。教員がアドバイスできるのは，「答え」ではなく「理解」が欲しい人に対してだけなのです。質問する場合は，自分は何がわかっていて何がわかっていないのか，何が知りたいのかを明確に言語化して質問するようにしてください。

付録 B

標準正規分布から尺度値を求める計算方法

Likert 法では，態度が標準正規分布すると仮定するのでした。標準正規分布をカテゴリの相対度数で分割し，あるカテゴリ c の上限の確率点 z_c，下限の確率点 z_{c-1} の確率密度の差分を，相対度数 p_c で割ることで，尺度値 Z_c が下の式で得られます。

$$Z_c = \frac{(y_{z_{c-1}} - y_{z_c})}{p_c}$$

ここで y_{z_c} は標準正規分布をカテゴリで区分し，当該カテゴリ c までの累積確率点 z_c における確率密度，p_c はカテゴリ c の相対頻度です。図 B に記号の対応関係を示しましたので，確認してください。

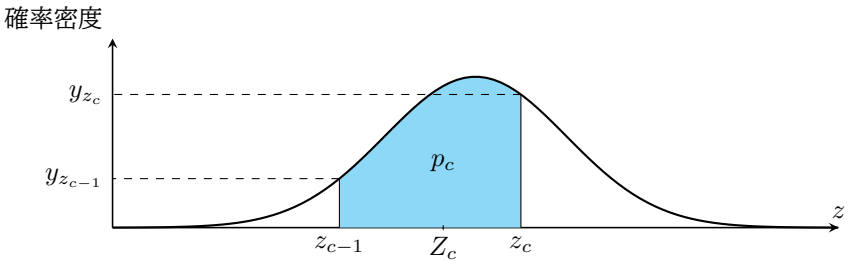

図 B.1　標準正規分布と対応する記号の確認

尺度値 Z_c を求める計算が確率密度 $y_{z_{c-1}}, y_{z_c}$ と，相対度数 p_c で算出されるというのは一見奇妙です。どうしてこのようになるのかを見ていきましょう。

まず標準正規分布の確率密度の式を確認しておきます。確率点 z における確率密度 y は次の式で算出できます[*1]。

$$y_z = f(z) = \frac{1}{\sqrt{2\pi}} e^{(-\frac{z^2}{2})}$$

この関数は確率密度の曲線を表しており，確率はその面積です。z_{c-1} から z_c までの面積 (確率) は，積分を使って

$$p_c = \int_{z_{c-1}}^{z_c} f(z)dz = \int_{z_{c-1}}^{z_c} \frac{1}{\sqrt{2\pi}} e^{(-\frac{z^2}{2})} dz$$

[*1] e^x を $\exp(x)$ と書くことも少なくありませんし，この e は自然対数の底で $e = 2.718...$ という実数です。この数字は微分しても変わらない，$(e^x)' = e^x$ という便利な特徴を持っています。

で表されます。

さて，ある確率点 z における密度の高さを $f(z)$ としたとき，z の微小な増分 Δz を考えると，区間の面積 $S = f(z)\Delta z$ を考えることができますから，逆にある点 z が知りたい時は

$$z = \frac{\sum Sz}{\sum S}$$

とすれば良いことになります。積分はこの微増分 Δz の極限を

$$\lim_{\Delta z \to 0} \Delta z = dz$$

と考えることですから，ここで考えたいのは

$$Z_c = \frac{\int_{z_{c-1}}^{z_c} f(z)zdz}{\int_{z_{c-1}}^{z_c} f(z)dz}$$

になります。分母は確率密度関数の積分ですから面積すなわち確率で，ここでは p_c であり，その面積を実データの相対度数で代えてもいいでしょう。

分子については少し式の変形が必要です。記号を見やすくするために，$a = z_{c-1}, b = z_c$ と書き換えておきましょう。

$$\begin{aligned}\int_a^b yzdz &= \int_a^b \frac{1}{\sqrt{2\pi}} e^{(-\frac{z^2}{2})} zdz \\ &= \frac{1}{\sqrt{2\pi}} \int_a^b e^{(-\frac{z^2}{2})} zdz\end{aligned}$$

ここで変数変換をして，$u = -\dfrac{z^2}{2}$ とすると

$$\begin{aligned}\frac{du}{dz} &= -z \\ du &= -zdz\end{aligned}$$

となり，積分の下限は $-a^2/2$, 上限は $-b^2/2$ になりますから，以下のように展開できます。

$$(\text{与式}) = -\frac{1}{\sqrt{2\pi}} \int_{-a^2/2}^{-b^2/2} e^u du$$

$\int e^u du$ は e^u なので

$$= -\frac{1}{\sqrt{2\pi}} \left[e^u\right]_{-a^2/2}^{-b^2/2}$$

$$= -\frac{1}{\sqrt{2\pi}} e^{(-\frac{b^2}{2})} - \left(-\frac{1}{\sqrt{2\pi}} e^{(-\frac{a^2}{2})}\right)$$
$$= \frac{1}{\sqrt{2\pi}} e^{(-\frac{a^2}{2})} - \frac{1}{\sqrt{2\pi}} e^{(-\frac{b^2}{2})}$$

$y = \dfrac{1}{\sqrt{2\pi}} e^{(-\frac{z^2}{2})}$ であることから，

$$= y_a - y_b$$
$$= y_{z_{c-1}} - y_{z_c}$$

以上のことから，リッカート法において標準正規分布をもとに尺度得点を決めるには，

$$Z_c = \frac{(y_{z_{c-1}} - y_{z_c})}{\int_{z_{c-1}}^{z_c} f(z)dz} = \frac{(y_{z_{c-1}} - y_{z_c})}{p_c}$$

とすれば良いことになります。

この計算に至る理論的背景は，より専門的には**系列範疇法 (Method of Successive Categories)** と呼ばれ，順序カテゴリに数値を付与する心理測定論，精神物理学理論からきています。詳しくは Guilford (1954 秋重訳 1959) や西村 (1977) も参照してください。

付録 C

電子計算機のイロハ

C.1　前置き

このセクションは心理統計ではなく，コンピュータについての四方山話をダラダラと書いています。そんなの聞かなくてもわかってるよ，という人もいるかもしれませんし，知らなくてもみなさんはきっとスマートフォンやタブレットを使えていることと思います。しかし知らずに使うことと，知ってて使うことには大きな違いがありますし，今後大学でレポートや論文を書いたり，それに必要な統計処理をするためにも，計算機の基本的な特徴を知っておくと，トラブルに出会した時に「あぁこれってひょっとして」というヒントが得られたり，納得できるようになるかもしれません。知らなければ「何だかわからないけどパソコンが壊れた」というか，「パソコン運が悪い」「自分はパソコンが苦手なのだ」と間違えた帰属をしてしまうことになります。

コンピュータ関係の授業で聞いたことがある話，これから聞く話もあると思いますが，もし苦手意識を持っている人がいたらこれを機に再入門するつもりで読んでください。

C.2　コンピュータの基礎

21 世紀に生きる私たちは身の回りをコンピュータに囲まれて生きています[*1]。それは携帯電話の形をしていたり，ノートパソコン，デスクトップパソコンの形をしていたりします。また時々テレビなどで報道されますが，気象予報や飛沫がどのように飛び散るかをシミュレーションする大型計算機「富嶽」などもコンピュータですね。これらは形は違いますが，いずれも電子計算機であり，電子計算機には次の 5 つの装置があります。

入力装置　キーボードやマウス，タッチパネルなどを使って情報を取り込むデバイス (装置)

出力装置　モニタやプリンタ，タッチパネルなどを経由して情報を出力するデバイス

演算装置　プログラムの命令に従って計算処理 (四則演算や論理演算) をする装置。一般に電子計算機の中央で一括して処理するので，Central Processing Unit(CPU) と呼ばれるものです。

制御装置　演算結果に従って他の装置に指示を出す装置のこと。演算装置とまとめて CPU に実装されています。

記憶装置　計算結果などの情報をいったん保持しておく装置のこと。コンピュータの内部にあって一時的な

[*1] 私は 1976 年生まれですが，生まれた頃は周りにコンピュータなんかありませんでした。小学生の頃にマイコン (マイクロコンピュータ，小さなコンピュータという意味でもありますが，My Computer，私のコンピュータという意味でもあります。つまり個人単位でコンピュータが使えるようになった，というだけでも大きな出来事だったのです。) という言葉が出てきて，なんかかっこいいなと思った記憶があります。私が 10 歳になったころ，ビデオゲーム (テレビゲームでやるゲームが家庭でできるようになり，それをこのように呼びました。) が身の回りに出てきました。ファミリーコンピュータ，とくに「スーパーマリオブラザーズ」によって日本中の子供たちが熱狂したのが 11 歳の頃，「ドラゴンクエスト」によって社会問題になったのが 12 歳の頃になります。ともかくこの頃は，コンピュータといってもゲーム機のような扱いでした。

計算に使われる一次記憶装置，ハードディスクドライブ (Hard Disk Drive,HDD) やソリッドステートドライブ (Solid State Drive,SSD) などの二次記憶装置など。

最後の記憶装置については，一次記憶装置，二次記憶装置と種類が分かれていますがこの区別は簡単で，電源を落とした時に記憶が消えてしまうのが一次記憶装置，電源を落としても記憶が消えないのが二次記憶装置です。たとえば $12+38=$ という計算をするとき，頭の中で「えーっと一の位が 2 と 8 だから 10 になって繰り上がるから･･･」と考えてから，ノートに $=50$ という答えを書くと思います。次の問題に進むと，先ほどの「1 繰り上がるから･･･」という情報は忘れてますよね。でもノートに書いた $12+38=50$ というのは残っています。このノートがいわば二次記憶装置であり，頭の中で一時的に保持していた情報が一次記憶装置ということになります。一次記憶装置は RAM(Random Access Memory) とも呼ばれます[*2]。

ところで，コンピュータがやっているのは計算だけです。私はこの資料を PC に向かって書いており，キーボード (入力装置) を叩きながら，画面 (出力装置) を見て文字を連ねています。これも「キーが押されたら文字を表示させ，その文字列を記録する」という処理を機械が淡々とこなしているに過ぎません。あるいはマウスやトラックパッドで，アイコンを指し示し，カチリと押す[*3]ことで選択し，押し込んだまま移動させ (ドラッグ)，離すことで別の場所に置いたりします (ドロップ)。トラックパッドの場合は 2 本指で同時に押したりしますし，タッチパネルの場合は二本指を広げたり (ピンチアウト)，逆に二本指を狭めたり (ピンチイン)，3 本以上の指でファサーっと触って場所を広げたりします。たとえば「ファイルを掴んでゴミ箱に捨てる (削除する)」というのが我々にとっての操作ですが，コンピュータの内部では実はこんなことをしていません。ファイルは (二次) 記憶装置に書き込まれた情報です。記憶装置は原稿用紙のように小さなマス目がたくさんあって，ファイルとはそのマス目の XXX 番目から YYY 番目までの情報，ということです。このファイルを削除するというのは，記憶装置のある場所 (アドレス) に「削除されたものなので画面に表示しない」という情報を書き込む，という操作をしているだけです。じゃあなぜ私たちは「ゴミ箱にドラッグ＆ドロップ」なんてするのでしょう？ それはそのほうがわかりやすいからですよね。「ファイルを削除する」というのは，「メモリアドレスの XXX 番地に別の情報を書き込む」という操作だと言われてもピンとこないので，コンピュータが人間にとってわかりやすい表現をして見せてくれているのです。このユーザにとってわかりやすい幻を見せてその気にさせてくれるというデザインのことを，ユーザーイリュージョンと言います。ともかくこういう「画面で見ながら操作する」ことをグラフィカル･ユーザ･インターフェイス (Graphical User Interface,GUI) といいますが，これのおかげでコンピュータの操作は随分楽になっています[*4]。

C.3 コンピュータの歴史

コンピュータの装置，すなわちハードウェアについての解説につづいて，ソフトの側面についても解説を加えようと思うのですが，そのためには少し歴史的な流れを説明したほうがわかりやすいかもしれません。

コンピュータの発展の歴史は，小型化の歴史でもあります。最初にできたコンピュータは ENIAC といいます。1946 年の話です。この ENIAC は 27 トン，広さにして倉庫 1 つ分 ($167m^2$，90 畳以上の広さ) が

[*2] 実はこのように人間を 1 つのコンピュータに喩えて，そこではどのように計算がされているのか，人間の記憶装置や演算装置，入出力装置の特徴はどうなっているのか，というのを研究するのも心理学の仕事です。記憶や演算，制御については認知心理学や学習心理学，入出力については知覚心理学や生理心理学が専門的に扱っています。そういう意味でもコンピュータの登場は心理学に大きな影響を与えているのですが，それはまた別の講釈で。

[*3] 押すときの音から，この操作をクリック click と言います。2 回続けて押すことをダブルクリックと言います。

[*4] 実は人間の意識もこのユーザーイリュージョンのようなもので，実際の体の動かし方や感覚情報の受け止め方，処理の仕方は別ですよね。すべての情報を意識に上げるのではなく，「私は XXX をしている」と身体がそれっぽい幻想を投影して見せてくれているのが意識の正体ではないか，という議論があります。興味のある人は Norretranders (1999 柴田訳 2002) を読んでみてください。

必要なもので，真空管を使った計算機でした。軍事的な計画のために開発されたオーダーメイドのものですから，一般人が触れるはずがないものです。時代が下がって真空管が半導体，IC チップになった頃，やっとサイズが小さくなって，家庭用・個人用のコンピュータというのができるかもという時代が来ました。Apple コンピュータの創始者，スティーブ・ジョブズとスティーブ・ウォズニアックが最初のパソコン (Personal Computer,PC)，Apple I を発売したのが 1976 年。爆発的に売れた Apple II が発売されたのが 1977 年です。AppleII はブラウン管表示装置とキーボードを持っていましたので，今の PC の原型とも言えるかもしれません[*5]。PC を作っている会社は Apple だけでなく，IBM や DEC などがありましたが，まだこの頃はパーソナルなレベルのものよりも大型計算機の開発が進んでいました。IBM が PC を作ったのは 1980 年で，この頃から小型化が進められていきます。

私事で恐縮ですが，1976 年に生まれた私が初めて PC を手にしたのは 1991 年，高校入学のお祝いで買ってもらった Fujitsu の FM-Towns という機体でした。この頃は「マルチメディア」という名前もなく，「ハイパーメディア」と読んで売り出していました。この機体は他の PC(NEC の PC-9801 や Sharp の X68000 シリーズが有名でした) とは違って，CD-ROM ドライブをつけていたことが画期的だった時代です。その後 1994 年に大学生になりましたが，この頃は連絡を取り合うツールはポケベルが主流であり，携帯電話 (や PHS) のような個人端末は高級品という時代でした。大学に入るとコンピュータを学ぶ授業があり，アカウントをもらったりするのですが，それは大学が持っている大型計算機に端末からアクセスするためのものでした。いろんな部屋にあるのは「端末」で，それほど機能の優れた PC ではなく，複雑な計算 (統計的な計算など) は大型計算機に仕事を依頼しその返信を待つ，というスタイルでした。関西私立のマンモス校でしたので学生数は非常に多かったのですが，多くの学生が一度にアクセスしても，大型計算機はものすごくものすごーく計算が早かったので，瞬時に回答をもたらしてくれるものでした[*6]。つまり，まだ「専門的な計算は大型計算機」という時代であり，パーソナルなコンピュータになるにはもう少し時間が必要でした。

どれぐらいの時間が必要だったかというと，実はその次の年なのです。1995 年，Windows 95 という OS が発売されました。これを機に日本でも PC がどんどん浸透していくことになります。Windows95 は物凄いんだぞ，と発売前からテレビでも散々とりあげられ，発売日には行列ができて真夜中のカウントダウンと同時に大フィーバー，という売れ行きでした。当時のそれは何が凄かったのでしょう？ コンピュータにはそれ動かす基本ソフトが必要です。HDD にデータを書き込み，キーボードからの入力をディスプレイに表示する，といったごく基本的な装置を統括し，メモリ番地をファイルという単位で扱うと言った基本的な操作は OS いうソフトウェアが担当します (図 C.1)[*7]。この OS，大型計算機は Unix と呼ばれるものを使っていましたが，各企業が個人向けにコンピュータを売り始めるときにも当然必要で，各社で開発もしていましたが，PC の共通規格をつくることで OS 部分は共有できるようになりました。そこを提供したのが Microsoft 社のビル・ゲイツです。どんなパーツで作られた PC であっても Windows という OS が共通のフィールドを用意してくれるので，ユーザは Windows で動くアプリケーションを選ぶだけで良い，ということになったのです。

そして Windows95 は，GUI，つまり「ファイルを掴んでポイ」といった直感的な操作で使えることも大きな特徴でした。大型計算機で使われている Linux は基本的に Command User Interface,CUI で，黒い画面にプログラムを書いて実行するといった手法で，初心者には人気がなかったのです。GUI については，その頃 Apple を追放されていたスティーブ・ジョブズが，NeXT という会社で GUI を備えた OS を開発していました。この NeXT はその後 Apple に買収され，ジョブズは Apple に戻って活躍することになります。その頃から巷では，Windows の GUI は Apple の OS を真似したものだと非難されていたのですが，商業的に

[*5] Mac の歴史については Isaacson (1995 井口訳 2011) が読み物としておもしろいですよ。

[*6] Time Sharing System,TSS，時分割システムとよばれる機構です。命令を小さな単位に分割し，それを順次捌いていくという方法でした。

[*7] 物理的な機構と OS との間に Basic Input/Output System,BIOS というのが入りますが。

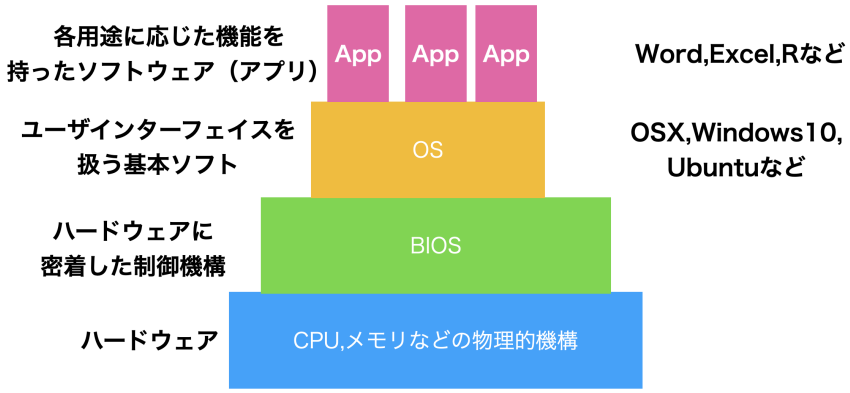

図 C.1 コンピュータのハードウェアとソフトウェアの関係

は Windows が大勝利，というわけです。ともかくこれを機に企業などはもちろん一般家庭でも PC を使うような時代になりました。

ちなみにインターネットが広まったのもこの頃です。私は大学 3 年生の時，大学の授業で初めてインターネットを介して世界の情報を得る，という経験をしました[*8]。その頃から徐々に，大型 PC のアカウントではなくインターネットで使えるメールアドレスというのを個々人が持つようになりました。携帯電話も廉価な PHS が広まり，メッセージだけでなく徐々に音声，写真，短い動画が遅れるようになっていきます。当初は当然画質・音質も今とは比べものにならないのですが，それでもインターネットを経由してつながるという経験は想像を超えたものでした。

皆さんはすでに，携帯電話やインターネットがある時代に生まれた世代だと思います。こんな話はすでに過去のものであり，不便な頃の話を聞かされても困る，と思うかもしれません。私の世代は，幸いにもこのようにちょうどコンピュータが使われはじめ，広がり，高性能になっていくにつれて育ってきていますので，そのぶん機械の根本的な理解に直結しやすい社会環境にあったのです。みなさんは，便利な時代ではありますが，言い換えると「初心者が苦労しないように補助輪をつけておく」「何もしなくてもできているような感覚が得られるようなイリュージョンを見せておく」という状態におかれているので，補助輪が対応できない道に進んだり，多くの人とは違う使い方をするときに，急にサポートが外れどこで困っているかわからなくなる，ということになりかねません。非常に大雑把な Historical Review ではありますが，理解の一助になればと思います。

ところで，先ほどサポートという言葉を使いましたが，このサポートを商売にしているのが Microsoft や Apple という IT 企業です。これらの会社は，ユーザが使いやすいようにソフトウェアを提供してくれますが，有償ですし想定外の使用をするユーザに対してはサポートをしてくれません。逆にいうと，「決められた路線を走らなければ面倒を見ない」ということでもあり，これはユーザに不自由を強いているとも言えます。また，どのような仕組みで動いているのかを尋ねても，企業秘密と言って答えてくれません。コンピュータは誰でも自由に使えるべきものであり，勝手にユーザの情報を盗んだりしていないか，と言ったチェックをするためにもオープンであるべきではないか。そういう考え方に基づく，フリーソフトウェアというソフトウェアのあり方があります。Unix という OS を PC 用にした Linux がそうであり，オフィスソフトの LibreOffice，統計環境の R などもこの精神に賛同するものです。フリーソフトウェアは自由であり，無償です。お金と秘密を払ってサポートを受けるのではなく，ユーザが相互に助け合ってオープンで自由な世界を広げていこうという活動です。急

[*8] 教職関係の科目で，教育技術として今後使われるだろうということで担当教員が実演してくれました。隣の部屋から電話線を延長し，モデムというパソコンの信号を音情報に帰る機器を繋げて，NASA の Web ページを Netscape というブラウザでみたのが初めての体験でした。

に何の話なんだ，と思うかもしれませんが，心理学ひいては科学的活動すべてにおいて重要な問題であることをご理解いただきたいと思います。

C.4 情報の単位

コンピュータを取り巻く世界の話はこれまでにして，ソフトの側面についての解説に入りましょう。

コンピュータは文字，音，絵，動画ファイルいずれについても，すべて 0 か 1 のデータとして管理します。0/1 の 1 つの単位を 1bit(ビット) と言います。1bit であれば Yes か No か，という二択の情報しか提供できませんが，これが 7 つあれば $2^7 = 128$ ですから，これで 128 種類の状態を表現できます。コンピュータは一般に，8bit で 1 つの単位として計算します。8bit のことを 1byte(バイト) と言います。この 1byte が 1024 集まったものを 1kb(キロバイト) といいます[*9]。1kb の次は，1024kb=1Mb(メガバイト) です。さらに 1024MB=1GB(ギガバイト) で，1024GB=1TB(テラバイト)，1024TB=1PB(ペタバイト)，1024PB=1EB(エクサバイト) と続きます。K,M,G,T,P,E といった名称は 1000 倍ごとに変わる大きさの桁をあらわしているのであって，「ギガが減る」というのは本来意味をなさない表現です[*10]。

このビット・バイトは情報に関する基本単位なのであちこちに使われます。記憶装置について使われるとき，一次記憶装置も二次記憶装置も同じ単位なので混同するかもしれませんが，2021 年現在では二次記憶装置の単位は GB から TB が使われます。USB フラッシュメモリーや外付け HDD などは数 TB の容量が一般的でしょう。これに対して，一次記憶装置は 4〜64GB ぐらいが相場かと思います。一次記憶装置は暗算の途中経過のように一時的に記憶する場所に過ぎないので十数 G でも問題ありませんが，二次記憶装置は結果の記録なので大きければ大きいほど余裕が持てますね。たとえば数 TB でもテレビドラマや映画を何本も記憶できるのですから，PB や EB なんて使うのかな，と侮ってはいけません[*11]。すぐにそれぐらいのサイズが必要な時代が来ることでしょう。

実際，それぞれ単位ではどれぐらいの情報が記録できるのでしょうか。1byte は 128 文字表現できるので，英語のアルファベット 26 文字に加え，数字や簡単な記号であれば 1byte で表現できます。たとえば A という文字は 01000001，B という文字は 01000010，小文字の a は 01100001，と言ったように 0/1 の文字列 8 個と一対一対応させて考えるのです。日本語は 1byte では足りませんので，一文字あたり 2byte が割り当てられます。また 1KB(=1024byte) は 500 文字ですから，原稿用紙一枚ぐらいになります。1MB(=1024KB) は文字だけだと新聞一紙 (朝刊の 40 ページ分) ぐらいで，昔の記録媒体であるフロッピーディスク一枚に保存できるのがちょうど 1MB でした[*12]。ちなみに「カメラ映像 + 音声」のオンラインビデオ会議を 1 時間やると 200〜300MB ぐらいの容量をやりとりしていることになります。ビデオ会議では文字 (チャット) だけでなく，画像 (動画) や音声も送り合いますね。実は画像や音声も，0/1 に置き換えています。音声の場合，1 秒間を短い間隔にくぎります。この区切りのことをサンプリング周波数といい，たとえば 44.1 kHz という単位は 1 秒間に $44.1 \times 1000 = 44100$ 点のデータの採取をします。この 6 データ点において，音の振動幅を区切ります。ビットレートと言いますが，たとえば 16bit で区切る場合は $2^16 = 65,536$ 段階で区

[*9] キロメートルやキログラムのように，キロは 1000 の単位を表す言葉ですが，コンピュータは 2 進数なので 1000 ちょうどではなく 1024 で 1 つ上の位に上がることになります。

[*10] 同様に「USB を紛失する」というのもよく聞くおかしな表現です。USB は Universal Serial Bus の略で，データ転送規格のことを指します。なくすことができるのはそれにつなげるメモリースティックなどです。

[*11] 私事ですが，私が 1991 年に生まれて初めて手した PC，FM-Towns はハードディスクが 40MB，RAM は 2MB で，CD-ROM がついていましたが，それは 360MB の容量でした。購入するときに，「ハードディスクが 40MB もあって何を記録するんです？ CD-ROM なんか情報が詰まり過ぎてますよ！」と店員さんに笑われたのを今でも覚えています。数年後，PC の動きが遅くなったので，2 万円で 2M の RAM を追加したのも良い思い出です。今でこそ，2MB なんて USB フラッシュメモリーでも売ってないほど微小なサイズですが。

[*12] 正確には 1.2MB 入る規格 (2DD) と 1.44MB 入る規格 (2HD) とがあります。

切り，その高さの音がある (1) かない (0) かで表すわけです。このように時間と音階を細かく区切り，その目に情報があるかないかを積み重ねてデータとするわけです。テキストよりも圧倒的に情報が多くなるのが分かりますね。画像も同様に，図面を細かい単位 (ピクセルなど) で分けて，そこの色合いを色々な段階で区切ります。色は R(赤)G(緑)B(青) の組み合わせで表現でき，それぞれを 8bit$= 2^8 = 256$ 段階で表現したりします。一点一点にその情報がありますから，図の情報も非常に多くなるのがわかると思います。動画はその画像が時系列的に細かく分割されたものと思ってください。このように分解しますので，テキスト，音声，図，動画の順にデータサイズが大きくなります。通信機器が最初ポケベル＝数 byte の情報しか送れなかったものから，徐々に絵文字，ショートメッセージ (音声)，写真[*13]，動画が送れるように発展していきました。今では町中のあちこちで，誰もが手軽に動画をモバイル端末で見られるようになっています。あらためて，すごい進歩ですね[*14][*15]。

ところで，1byte は 256 種の情報が記録できるので，英語のアルファベットや数字は 1byte あれば十分だが，日本語や中国語など，英語以外の言語は文字種が多いので，2byte で一文字を表すという話をしました。この 2 バイト文字も，たとえば「あ」とか「亜」という文字に 111000111000000110000010 とか 111001001011101010011100 という文字列を割り当てるのですが，言語ごとによってどの数字をどの文字に割り当てるかという対応表が変わってきます。これを文字コードと言います。日本語はかつて Shift-JIS というコードで変換していましたが，今は世界のあらゆる言語に対応している共通企画である，UTF-8 という文字コードで変換することが一般的です。ところがなぜか，日本の Windows OS だけ Shift-JIS をいまだに使い続けており，他の PC とファイルをやり取りするときに文字コードの変換エラー問題が起きます。ファイルを開いて文字化けをしたりとか，プログラムが実行される際に「ファイルにアクセスできない」というエラーが生じたりするのは，この文字コードの問題が大きいのです[*16]。受身的な対策法になってしまいますが，PC でつかうユーザ名やファイル名などは半角英数文字を使い，短くした方がこうしたエラーに出くわしにくくなります。逆に，全角文字やスペース (空白) などを含んだファイル名，やたらと長いファイル名を使っていると，こうした問題に出くわしやすくなるということです。

C.5 ファイルの種類と拡張子

ここまで述べてきたように，計算機というのは基本的に物理的実体 (記録装置，記憶装置) の上で 0/1 のデータをやり取りしているだけです。記録 (記憶) 装置上に置かれている情報のセットは「ファイル」という形で記録されています。スマートフォンやタブレットは，ユーザの利便性のためにファイルの存在を意識しなくても良いようになってはいますが，バックエンドでは実行されるアプリケーションもファイルですし，開かれる音声や動画もファイルです。とくにパソコンでは，どの媒体，どのアプリで使うどういうファイルかを識別するために，拡張子 (かくちょうし) と呼ばれる識別記号をファイルの後ろにつけています。拡張子はファイル名の背後にピリオドで区切って追記されています。ついてないように見えても，OS がそれを表示させない設定にして

[*13] できた当時は写真を撮ってメールができることをとくに「写メールする」と言い表したほどです。

[*14] 実は音でも画像でも，分割してそのままデータにしてしまうと膨大になりすぎるので，人間が気付きにくい周波数や色合いなどは削除して作ります。これを非可逆圧縮処理と言います。一度落としてしまった情報は戻らない，という意味です。ライブや生きている人間が処理している情報は，携帯の画面から得られるものの何億倍もの情報量なんですよ!

[*15] デジタル化のすごいところは，こうした文字，音，図版，動画といったものを bit という共通の単位に落とし込んだことです。こうすることですべて一元的 (bit という共通次元) で処理することができるようになったのです。メディアの違いが問題にならなくなり，0/1 の情報であれば複写も簡単にできてしまいます。情報化社会においては情報に特別性はなく，情報があるかないか，それを生み出せるかどうかこそが重要なのです。

[*16] Windows だけ世界標準から外れているので，早く修正して欲しいのですが，歴史的な経緯からユーザ数が多くて切り替えられないでいることと，こうした違いがあることをユーザに説明しない (素人は知らなくていい，と馬鹿にされているようなものです) ので，問題が解決される日はまだ先になりそうです。

あるだけであることに注意してください。代表的な拡張子と，それに対応づけられているアプリケーションは次のようなものがあります。

.docx マイクロソフト社の文書作成アプリケーション，Word で使うファイル

.xlsx マイクロソフト社の表計算アプリケーション，Excel で使うファイル

.pdf Adobe 社が開発した Portable Document Format 形式。OS が違っても同じレイアウトで文書を表示できるのが利点で，PDF 形式を読むことができるアプリケーションは多数。

.txt シンプルな文字だけのテキストファイル。文字の飾りやレイアウトなどの情報がない最もプレーンな形式なので，OS が違っても文字コードさえ合っていれば読むことができる。

.mp3 音楽，音声のファイル。音声データには他の種類もあります。

.jpg 画像のファイル形式の一種。

.png 画像のファイル形式の一種。

.csv comma/character separated variables ファイル。変数をカンマ (,)，あるいはタブ，半角スペースなど文字コードで区切ったファイルという意味で，中身は.txt と同じく装飾のない文字/数字だけであり，文字コードさえ間違えなければ OS を問わずに読み書きできる。データのやり取りはこの形式で行われることが多い。

.zip 圧縮ファイルの一種。1 つまたは複数のファイルをパックして圧縮してあるもの。ファイルの冗長な部分をうまくまとめてコンパクトにまとめ上げるため，ファイルサイズが小さくなるし，複数のファイルもひとまとめにできる。また圧縮の際にパスワードをかけることもできるため，メールなどに添付する場合はこの形式にまとめられることが多い[*17]。可逆圧縮であり，圧縮されたファイルは展開する (解凍するともいう) ことでパッキングを開封できる。zip ファイルの圧縮/展開は各種 OS が標準的に対応している。

`.txt` や `.csv` といった形式は，「装飾のない，文字だけの」ファイルです。こうした種類のことを ASCII ファイルと言います。メモ帳などのエディタと呼ばれるプログラムで読み書きできます。逆に `.docx` など特定の会社が提供するアプリケーションに対応しているファイルは，アプリの中でのさまざまな操作・装飾を暗号化して保存しており，メモ帳などで読んでも意味がわかりません。対応しないアプリでは開くこともできません。こうした形式は ASCII ファイルに対してバイナリファイルと言います。

OS は拡張子を見てファイルの種類を判別し，そのファイルを開くのに適したアプリケーションを自動的に起動し，開いてくれます[*18]。`.csv` ファイルは Excel などのアプリケーションで開くことも当然できますが，その際文字コードのエラーが生じたり，保存するときに文字コードを変えたりして，形式・内容が気づかずに変わっていることがあります。Windows も良かれと思ってやっていることなのですが，処理が徹底してないのか，かえって不便になってしまっています[*19]。

[*17] **PPAP** というとピコ太郎の楽曲を思い出す人もいるかもしれませんが，圧縮ファイルの文脈では“「Password つき zip ファイルを送ります。Password は次のメールで送ります」Angoka(暗号化) Protocol の略です。つまりメールでパスワード付きのファイルを送り，そのファイルを開くためのパスワードをまたメールで送るという，日本でよく見られるおかしな風習です。おかしな，というのは，メールがハッキングされていたらパスワードもどうせバレるわけで，同じメールに書いてあるのはもちろん馬鹿馬鹿しいですが，すぐ次のメールに書いてあるのも同じぐらいに馬鹿馬鹿しいことです。情報セキュリティ対策手法のつもりで行われる慣習が広まっていますが，PPAP の標語のもと，馬鹿馬鹿しいのでやめましょうという風潮になってきました。

[*18] 見たことのない拡張子の場合は，どのアプリケーションで開くべきか尋ねてくるでしょう。

[*19] 実際，この授業での課題データを UTF-8 形式の csv ファイルで提供しても，Excel で開いたばっかりに文字化けして分析できなくなる，という相談がこれまで多く寄せられています。根本的な解決策として，Windows を使うのをやめることをお勧めします。

C.6 クラウドとは

すでに述べたように，計算機は基本的に 0/1 データのやりとりであり，それを保存してあるのがファイルとよばれるものです。ファイルは HDD や USB メモリ，SSD に保存することができます。

ところで，最近はこうした手元の物理的実体にファイルを置くことに加えて，クラウドに保存することも少なくありません。クラウドとは雲という意味で，インターネットの向こう側のどこか，ということを意味します。ですが，基本的にはインターネットで繋いだその先にも電子計算機があるのです。たとえばパソコン A とパソコン B をケーブルで繋ぐと，パソコン A からパソコン B のファイルにアクセスできます[*20]。このケーブルをどんどん伸ばすと，遠く離れていてもこの操作ができます。このケーブル網を世界レベルに広げているのがインターネットです[*21]。

ここで覚えておいて欲しいのは，当たり前のようですが，ネットといっても基本的には電子計算機と電子計算機を繋げている実体がどこかにあって，ファイルのやりとりをしているだけだということです。ブラウザはウェブサイトの情報を書いたファイルを取り込んでホームページを見せてくれていますし[*22]，Youtube は動画のファイル，Instagram は画像のファイルへのアクセスをして見せてくれているのです。最近ではクラウドサー

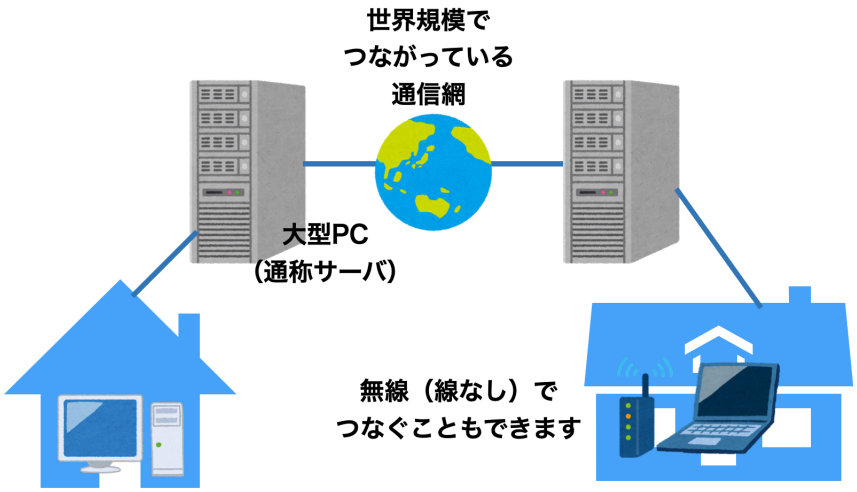

図 C.2 インターネットの世界

ビス，というのがよくあります。中でも Dropbox とか OneDrive，iCloud などが有名です。これらは，ファイルをインターネットを介した別の大型電子計算機に保存 (アップロード) し，インターネットを介して読み込み (ダウンロード) して使う，というものです。手元の電子計算機の中に記録されているファイルのことを「ローカル」，サーバなど遠隔地にある大型機に記録されているファイルのことを「サーバ」「クラウド」と呼んで区別しますが，このローカルとサーバのファイルを常に同じものに同期しておく便利なシステムです。こうしたクラウドサービスを使うと，たとえば大学で作業して保存したファイルを，USB メモリに保存して自宅のパソコンにコ

[*20] もちろんファイルの情報をどのように信号に変えて送受信するか，アクセス権限はどうするかといった細々したことを調整しなければなりませんが，そのあたりの仕事をやってくれるのが OS のありがたいところです。

[*21] インターネットは軍事通信網として始まりましたが，それを電話回線や企業内通信網，大学間通信網などと接続しあって世界中に広げています。網と網が相互に (inter) 繋がっているので inter net であり，大学内・企業内のネットワークのことはイントラ (intra) ネットと言います。ちなみに一般用語としてのインターネットは Internet と大文字で書き始めます。

[*22] ホームページとは本来ブラウザが最初に開くページのことを指し，企業や個人が情報発信しているページのことはウェブサイトというのが適切です。

ピーする，という操作をしなくても，大学でクラウドサービス上のフォルダ内に保存しただけで，自宅のパソコンにそのファイルがコピーされているため[*23]，意識せずにそのまま作業を続けられるということです。自動的にバックアップをとってくれているとも言えるので便利ですね。

もちろん注意すべきこともあります。「他の人に見られたら困るファイル」，「アプリケーションの実行ファイル」などは通信網の向こうではなく，手元 (ローカル) に置いておきましょう。個人情報・機密情報などを，クラウドドライブに保存すると，悪意を持った人が大型計算機に攻撃を仕掛けて情報を盗んでいく可能性があるからです。情報化の怖いところは，取られても気づかない (コピーすればいいだけで元ファイルになんの影響もない) ところにあります。加えて，取られたものをばら撒かれる＝インターネットを介して誰でもアクセスし保存できるようにされると，すべてを回収できなくなるのも問題です。失言が記録されて拡散されると大変なことになるのは，みなさんもこの時代に生きる人間のマナーとして色々見聞するところと思います。また，クラウドサービスで自動的に同期されるといっても，アプリケーションの実行ファイルなどはローカルに保存するべきです。アプリケーションは実行に際してさまざまな関連ファイルにアクセスしますので，1 つでも場所が違うところがあるとエラーになって動かなくなります。同じ OS でも，です。インターネットからとってきたソフトウェアをうっかり OneDrive に保存してしまうと，アクセスできないエラーで起動しない，ということもありますので注意してください[*24]。

余談ですが，R などプログラミングを推している時に最も頻発する質問が「ファイルがありません/読み込めません」です。その理由は大きく 2 つあって，1 つはスペルミスです。ファイル名の大文字・小文字の違いということもありますし，拡張子のまえに半角スペースが入っている，というよくみてみないと気づかないようなものもあります[*25]。2 つめの問題は，パスが通らないことです。デスクトップに置いたのに見つからない，ということもよくあります。教員や TA を悩ませるタイプの問題として，同じファイル名・フォルダ名が別の場所にあることがあります。フォルダ A の中にファイルを置いたのに見つかりません，という質問に対応していて，確かにフォルダ A を開いているのにな，と思ったら別の場所のフォルダ A だった，ということがあるのです。パスは隅々まで同じでなければ意味がないので，自分がどこに何を置いたのかをしっかり把握しておきましょう[*26]

C.7 ファイルの位置の指定

ここでファイルとそのパスについての話をしておきたいと思います。

計算機が情報を 0/1 で管理し，それらがファイルとなってどこかに保存されている，ということでした。私たちは Finder や Exploer などファイルブラウザをつかって，実行したいファイル，参照したいファイルを探していきますね。ファイルはまとめてフォルダの中に含まれていますし，フォルダの中にフォルダがあるといった，階層状態になっていることも少なくありません。ちなみに

[*23] これはユーザが特段の指示をしなくても，アプリケーションがユーザの見えないところで (バックグラウンドで) アップロード，ダウンロードの作業を進めているからです。パソコンはシャットダウンしていなければ，裏でこうした作業を淡々とこなし続けてくれています。

[*24] マイクロソフト社は，これまたユーザのためを思ってやっているのかもしれませんが，デフォルトで OneDrive に保存させようとします。それでうまくインストールできなかったという相談も多々寄せられています。抜本的な解決策として，Windows を使わないことをお勧めします。

[*25] k これはたとえば，`sample.csv` というファイルをダウンロードしてね，と指示した時に，複数回同じ場所に保存してしまって，機械が，`sample (1).csv`，`sample (2).csv` などと名前をつける時に起こります。この `(1)` を削除してもその前に半角スペースが入ってたりするのです。まったく Windows は！

[*26] Windows は One-Drive にもデスクトップがあります。同じ環境をローカルと同時にクラウドにもあって常に同期していればいいのですが，必ずしもそうではないので (OneDrive の設定によります)，`C:\Users\Username\Desktop` と `C:\Users\Username\OneDrive\Desktop` のどちらに保存したのかが分かりにくくなっています。デスクトップに保存したのに，と言われてもどっちのデスクトップなのか，という問題が生じるわけです。さらに専修大学では 2023 年から仮想デスクトップ環境 (VDI) を使うようになりました。これで 3 つ目のデスクトップができたことになるわけです。なんてこった！まったく Windows は！

フォルダと同じ意味で**ディレクトリ**という言葉が使われることもあります。

C.7.1 相対パスと絶対パス

普段 PC を使っているときは気にすることがありませんが，R や RStudio などプログラミング言語をつかっているときは，「今どこで作業しているか」という現在地が重要になってきます。たとえば，RStudio で `C:\User\kosugitti\Document\kiso1\`というところでプロジェクトを開いているとします。スラッシュ (\) はフォルダ，コロン (:) はドライブを表す記号です。プロジェクトフォルダは，C ドライブの User フォルダの下にある，kosugitti フォルダの下にある，Document フォルダの下にある，kiso1 というフォルダということになります (プロジェクト名が `kiso1` だとそうなります。)。この kiso1 フォルダが現在地です。

このフォルダの中で，Rmd ファイルや R スクリプトファイルを使って，他のファイルを参照するようなコードを書くとしましょう。たとえば `script1.R` というファイルに `read_csv` 関数を書いたとします。読み込みたいファイルは，同じフォルダの中にある，`sample.csv` とだとします。このとき，`read_csv` の書き方は次のようになるでしょう。

code：C.1 相対パスで読み込む

```
1  dat <- read_csv("sample.csv")
```

しかし別の書き方もあります。たとえば code:C.2 のような書き方でも問題ありません。

code：C.2 絶対パスで読み込む

```
1  dat <- read_csv("C:\User\kosugitti\Document\kiso1\sample.csv")
```

後者 code:C.2 の書き方は，ファイルの場所を全部書いてありますから，確実にその場所が特定できます。それに比べて前者 code:C.1 の書き方は，なぜファイルを書いただけでいいのでしょうか。これは，このコードを実行している現在地と同じフォルダの中に `sample.csv` ファイルがあるからです。プログラムは，命令を受けるとファイルを探しにいきますが，現在地と同じフォルダの中を探すことになっているのです。この現在地，すなわち現在作業しているフォルダのことを**作業フォルダ (working directory)** と言います。

では作業フォルダと別のフォルダの中にファイルがあれば，アクセスできないのでしょうか。そんなことはありません。code:C.2 の書き方を使えば，作業フォルダがどこにあっても位置を特定できますから，作業フォルダを問わずに書くことができます。ちょっと長くて面倒ですが，確実にある場所を指定しているからです。この書き方のことを**絶対パス**による指定といいます。一方，code:C.1 の書き方は，今の作業フォルダから見た場所，という相対的な書き方になっています。この書き方のことを**相対パス**による指定，といいます。相対パス指定で，違うフォルダにアクセスする場合には，次のようにします (code:C.3)。ここでは，`Document` フォルダの中に，kiso1,kiso2 フォルダがあり，kiso1 フォルダの中で作業している時に kiso2 フォルダの `sample2.csv` ファイルを読み込む例を書いています。

code：C.3 絶対パスで読み込む

```
1  # 絶対パス指定
2  dat <- read_csv("C:\User\kosugitti\Document\kiso2\sample2.csv")
3  # 相対パス指定
4  dat <- read_csv("..\kiso2\sample2.csv")
```

絶対パスはそのままなのですが，相対パスは..\という記号になっていますね。このピリオドを 2 つ打つ方法で，「ひとつ上のレベルの」という意味になります。このように，現在地からの相対的な位置関係で，ファイルを指定することもできます。

絶対パスと相対パスのどちらが良いのか，というのは一概には決められません。絶対パスは，PC が変わったりフォルダの構造が変わったりすると役に立ちませんから，使い勝手が悪いと言えなくもないですが，確実に指定できる方法です。相対パスは，PC が変わったりしても「現在地から相対的に見てどこか」という話ですから，たとえばこの例で `kosugitti` フォルダごと別の場所に移しても (たとえば `D:\Univ\Classes\kosugitti\kiso1` のように)，コードはエラーなく動きます。kiso1 フォルダ，kiso2 フォルダの相対的な位置関係が変わらなければいいのですから。バックアップを取ったり，複数の環境で同期しながら作業する場合などは相対パスの方がいいでしょうね。

いずれにせよ，現在どこで作業しているかということ，すなわち作業フォルダの場所を，意外と意識しておかなければならないということには注意が必要です。ファイルをどこに置いたか，どんなファイルを置いたか，自分はどこにいるのか，これが変わってくると「ファイルが見つかりません」というエラーになるのです。言い方を変えると，**ファイルが見つかりませんエラーの原因は，この 3 つのどれかである**ことがほとんどです。

C.8 ファイルのバージョン管理

これからみなさんは大学生活の中で，たくさんのファイルを生み出していくことになるでしょう。たとえば 4 年生の時に卒業論文を書くことになりますが，データファイル，分析ファイル，図を書いたファイル，引用文献リストを書いたファイル，卒論本文などなど，1 つの研究でも複数のファイルが作られることはよくあります。さらに，これらのファイルは日々加工されますから，その度に上書き保存することになります。いわば，ファイルがバージョンアップしていくのです。

卒論などの場合はとくに，「途中で保存しておく」ことが重要です。途中まで書いていた時に，横に置いていたマグカップが倒れて PC にコーヒーがかかり，変な音を立てて PC が壊れてしまった，ということがあるかもしれませんからね。紙に書いていた時代は，その手のハプニングがあってもせいぜい原稿用紙数枚がダメになっただけで，「ちくしょう，やりなおしかぁ」で済んだのですが，電子データの場合は電子の藻屑になると復元させることができません。ですから**バックアップは非常に大事**なのです[*27]。

バックアップの基本は，「別の場所に」「別のファイル名で」というものです。同じ名前で上書きすると元に戻ることができませんから，面倒でもコツコツと違う名前をつけましょう。そうするとよくあるのが，`soturon1.docx`,`soturon2.docx`,`soturon3.docx`,`soturon3`(**修正**)`.docx`,`soturon3`(**最新**)`.docx`,`soturon3`(**最新**)(**修正**)`.docx`,`soturon3`(**最新**)(**修正**)(**提出版**)`.docx`,`soturon3`(**最新**)(**修正**)(**提出版** 2)`.docx`.... というようになっていくやつで，「どれが最新版だっけ...」と書いてる本人でも探すのに苦労することになります。

この問題の解決策として，`soturon1001.docx`，`soturon1005.docx` のように日付を入れるというものがあります。10 月 1 日分，10 月 5 日分，としていけば「いつまで戻れば良いか」もわかるのでいいやり方ですね。日付の数字をファイルの先頭につけておくと，並べ替えも簡単です。この日付をつけて保存するというのを習慣化し，1. 昨日までのファイルを開いて今日の日付で別名保存する，2. 作業を進めて，時々上書き保存，最後にも上書き保存，3.PC に USB メモリをさして，バックアップ保存して作業終了，というルーティンを作っておくと，確実に記録が残って良いでしょう。

ただし，この方法の問題は，ファイルサイズが大きくなりすぎることです。図表などを含めたファイルが数百 MB になることは少なくありません。それを次々複製していくわけですから，大容量の USB メモリでも限界が来るかもしれません。これは「丸ごとコピー」していることが原因で，たとえば昨日は 10 行目まで書いた，今

[*27] ちなみに私の経験上，レポートが電子の藻屑になったので助けてください！と言われることがよくあり，4 年間の学生生活の間では平均 10–15 名に 1 人の割合で発生することのようです。

日は 11 行目から 14 行目まで書いた，というのであれば，この増えた 4 行分 (差分) だけを追加保存すればいいのに･･･と思いませんか。

こうしたバックアップやバージョン管理をやってくれる仕組みとして，Git というものがあります。Git は作業フォルダの中身の差分だけを記録し，必要であれば過去のバージョンに戻すこともできるシステムです。毎回上書き保存 (commit する，といいます) のたびに「どこを変更したか」というメモを付けて保存しておけば，そのメモを見ながら「ここの時点まで戻ろう」という使い方をすることができます。ファイル名は変更する必要なく，同じファイル名で進めていけますから，同じようなファイルがたくさんあって訳がわからなくなるということもありません。さらに保存先をクラウドにした GitHub というものもあり，これを使うとクラウド上に追記していくことができます。この GitHub は IT 企業などでプログラムをチームで進めていく時にも使われている技術で，全体のプログラムに個別の機能を複数人が追加，管理者が必要なものだけ取り入れる，というように使われています。国里ゼミや小杉ゼミでは，卒論を GitHub で管理し，学生が書いた分を commit し，それを hub 上にアップロード (push といいます) する，というようにします。教員の方は学生の進捗が管理できますし，どこがどう変わったかが分かりやすく，バージョン管理と同時にバックアップもできるという便利な仕組みです。GitHub は無料でアカウントを作ることができますから，興味があれば皆さんもチャレンジしてみてください[*28]。

さてここで，ひとつ注意をしておきます。卒論の原稿やプログラムは日々変化するものですからバージョン管理が必要ですが，データファイルはアップデートする必要がありません。いや，アップデートしてはおかしいのです。たとえば 100 人分のデータを取って分析をしていて，後で「やっぱりこのデータを削ろう」というのは，研究不正が疑われかねません。自分に都合の良いデータだけで議論し，都合の悪いデータは削除して統計的に有意な結果が出るように細工しよう，なんてことがあれば困るのです。データファイルはバージョンアップせず，それを加工，計算するプログラムがバージョンアップしていく。そしてその加工プロセスは誰でも見ることができるように公開されている。少なくとも，科学的な営みをする上では，そうしたやり方が必要なのです。自分だけのデータで自分だけの分析方法で，良い結果だけ示すというのは適切な方法ではありません。

Open Science Framework(https://osf.io/) はこうした「オープンな科学」にむけた取り組みの一種です。このアカウントは誰でも無料で作ることができ，ここにファイルをアップロードしたり，分析計画を事前に記録しておくことができます。何も難しいことではなくて，クラウド上のファイル置き場だ，というぐらいに思っていただければ結構です。ここにおかれたファイルは自動的にバージョン管理され，同じファイル名のものがアップデートされてもその記録 (ログ) が残ります。最近はこの OSF をつかって，論文化されたデータやプログラムを公開するという取り組みも進んでいます。

クラウド，バックアップ，オープンサイエンスといった新しい研究方法は日々生まれてきています。皆さんも便利な機能はどんどんキャッチアップしていきましょう！

C.9 おわりに

古臭い話をしてしまうのは，私が歳をとった証拠でしょうか。皆さんはこんな話を知らなくても，スマホやタブレットを使いこなしていることと思います。細かいことを知らなくても，ユーザとして利用するだけなら知らなくて良い話なのかもしれません。私はここにも書いたように，高校生の頃からコンピュータの発展と一緒に大人になってきましたから，学ぶともなく学んできたところがあります。皆さんは生まれた頃からコンピュータやがあったネイティブ・デジタル・カウボーイですから，苦労なんかする必要なかったわけです。

[*28] このテキストやシラバスも GitHub で管理していますし，公開されているサイトも GitHub 上のものです。これからは教科書も日々成長していくものになるかもしれません。

しかし細かい仕組みを知らないということは，問題が生じた時に「何か・誰かが，どこかでどうにかなって，今私が困っている」という状況になります。問題を特定できないと，解決することもできません。コンピュータは文房具に過ぎませんから，それを使いこなせないほうが格好悪いのです。しかも今後ますますコンピュータに囲まれた世界になっていくのは自明ですから，ここに学習コストをかけない方が勿体無い。幸い，わからないことに対して，自ら調べて学んだ利する時間と環境が用意されているのが大学という世界なのですから，今のうちにしっかり基礎固めをしておきましょう。

ここのくだらない懐古的エッセイのような文章が，何かの足しになれば幸いです。

付録 D

ギリシア文字一覧

ギリシア文字ってかっこいいけど，読み方わからない･･･という人のための一覧。ついでに TEX 表記も。

表 D.1　ギリシア文字とその読み方

読み方	大文字	小文字	英語表記	TEX 表記
アルファ	A	α	alpha	`\alpha`
ベータ	B	β	beta	`\beta`
ガンマ	Γ	γ	gamma	`\gamma`
デルタ	Δ	δ	delta	`\delta`
エプシロン	E	ε	epsilon	`\varepsilon`
ゼータ	Z	ζ	zeta	`\zeta`
イータ	H	η	eta	`\eta`
シータ	Θ	θ	theta	`\theta`
イオタ	I	ι	iota	`\iota`
カッパ	K	κ	kappa	`\kappa`
ラムダ	Λ	λ	lambda	`\lambda`
ミュー	M	μ	mu	`\mu`
ニュー	N	ν	nu	`\nu`
クサイ	Ξ	ξ	xi	`\xi`
オミクロン	O	o	omicron	`\mathrm{o}`
パイ	Π	π	pi	`\pi`
ロー	R	ρ	rho	`\rho`
シグマ	Σ	σ	sigma	`\sigma`
タウ	T	τ	tau	`\tau`
ウプシロン	U	υ	upsilon	`\upsilon`
ファイ	Φ	ϕ	phi	`\phi`
カイ	X	χ	chi	`\chi`
プサイ	Ψ	ψ	psi	`\psi`
オメガ	Ω	ω	omega	`\omega`

付録 E

記号の入力とキーボードの場所

プログラミングのミスでよくあるのが打ち間違い，スペルミスです。X と x，S と s など大文字と小文字で形が同じものや，l(エルの小文字) と 1(数字のイチ) の違いなどは，プログラミング用フォントにして違いがわかるようにするとか，文字列の意味から類推する (Normal とあればノーマルであって，ノーマ・イチではないと察する) など工夫が必要かもしれません。

また，理由はよくわからないのですが頻発するスペルミスは，データ (data) をデート (date) と書いてしまうことです。データはラテン語の datum(与えられたもの) の複数形なのですが，最近のデータサイエンスの文脈では data も単数形と考えるようです。ともかく，日付を表す date とは由来も意味もスペルも全て違うので，気をつけましょう。

さて，これらはまだ序の口。プログラミングのコードを読んでも，日本語の五十音に入っていない記号の違いがわからない，どこでそれが入力できるかわからない，質問しようにも読み方がわからないといったものもあります。ここではこれらをまとめて解説します。記号の上では微妙な違いですが，当然形が違うのでプログラミング上は違う文字として扱われますので，形の細部までよくみてください。なお，プログラミングでつ変わる時は言語に依存することもありますので，ご注意ください[*1]。

特に目立つのはハイフンとチルダ，アンダースコアの入力ミスです。それぞれ文字を書く領域における位置が違ったり，形が違ったりするのでよくみてください。

ハイフンは真ん中	A-B
アンダースコアは下	A_B
オーバーバーは上	A¯B
チルダはニョロ	A~B

これを踏まえて，そのほかの記号の名称や意味を表 E.1 で確認しましょう。

一般的な日本語キー配列の場合，1 つのキーに 4 つの文字・記号が割り当てられていますが，英語入力モードの場合はキーの左側，日本語入力モードはキーの右側を見ることになります。キーを押すと下の段の文字が入力されますが，シフトキーを押しながらキーを押すことで上の段の文字が入力されることになります (図 E.1)。

これを踏まえて，日本語キーについては E.2，US キーについては図 E.3 に代表的な記号とキー配列の位置を示しました。入力に困った場合は一度図を見て確認してください[*2]。

[*1] たとえばコメントアウトは C 言語では\\，R では#，TeX では % など，それぞれ異なります。

[*2] US キー配列はキートップに一種類の文字しかなく，美しい配置なのでおすすめです。

表 E.1 記号と読み方

記号	読み方	解説
:	コロン	英文中では「すなわち」などの意味。セミコロンと間違えないように
;	セミコロン	英文中では文章の区切り，接続詞のようにつかう
.	ピリオド	英文の終わりを意味する。日本語で言う句点
,	カンマ	英文の区切りを意味する。日本語で言う読点
@	アットマーク	メールアドレスに用いられることで有名
$	ドルマーク	米国の通貨単位。R では変数名指定のときにもちいる
/	スラッシュ	割り算の記号
*	アスタリスク	掛け算の記号
+	プラス	足し算の記号
-	マイナス	引き算の記号
^	ハット	累乗の計算の記号
=	イコール	等号。プログラミングでは==で一致しているかどうかの判定にも
!	エクスクラメーション	強調。プログラミングでは否定 (NOT) の意味になることも
_	アンダースコア，アンダーバー	位置に注意。ハイフンではなく文字領域の下の線 変数名をつなげる時 (ex. snake_case) に使ったりする
~	チルダ	R では独立変数と従属変数をつなぐときに使う ハイフンやオーバーライン，アンダースコアと間違えられる率高め
‾	オーバーライン，オーバーバー	アンダースコアの逆。滅多に使わないが。文字化けを直したときにみられる。
%	パーセント	プログラミングではコメントアウトの時などに使われたりする
&	アンパサンド	プログラミングにおける AND(論理積) の記号など
|	縦棒	プログラミングにおける OR(論理和)，条件付き確率の記号にも
\	バックスラッシュ	プログラミングではコメントアウトの時などに使われたりする
"	ダブルクォーテーション	文字列の開始･終了を表す。同じ記号で閉じる
'	シングルクォーテーション	文字列の開始･終了を表す。同じ記号で閉じる
`	バッククォーテーション	文字列の開始･終了を表す。同じ記号で閉じる
[]	大括弧	プログラミングでは配列を意味することがある
{}	中括弧	プログラミングではブロックの開始と終了を意味することがある
()	小括弧	数式のまとまりを表す

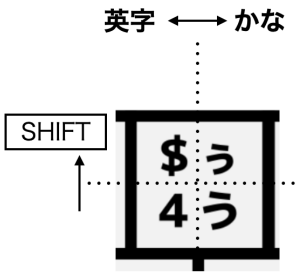

図 E.1 日本語キーで入力する場合

図 E.2　代表的な記号と日本語キー配列

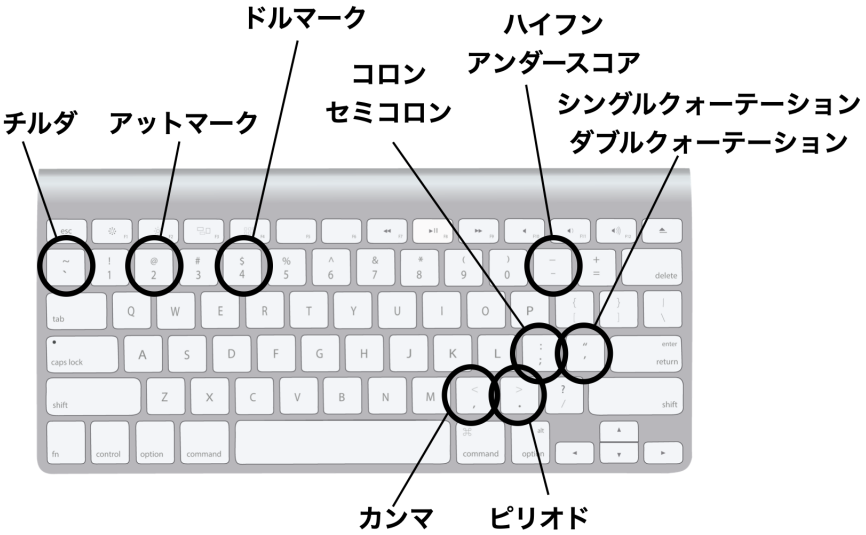

図 E.3　代表的な記号と US キー配列

付録 F

本講義に対応する詳細シラバス

F.1 プログラミングの基礎

F.1.1 授業内容

科目の中でのこのコマの位置づけ

本講義の主眼は「データから意味のある情報を取り出す」ことにある。すなわち，これまでのデータ駆動型 (Data Driven) な発想を逆転し，データがどのように生成されたと考えるかという観点から，データ生成モデル (Data generating Model) による理解を試みる。

モデルは数学的表現がなされ，モデルの形成やデータとの照合 (fitting) には計算機の利用が必須である。初回となる今回の目的は，プログラミングの基本的な考え方を身につけ，次回以降の本格的な運用に備えることである。プログラミングの基礎はコマンドによる命令であり，コマンドの書き間違えはエラーとなって帰ってくる。一見不親切に思えるが，即時反応により学習の効率は良く，コード補完機能などを用いることで簡単なミススペルは回避することができる。小さなものから大きくしていくこと，1 行ずつ実行することなど，基本的な姿勢についても理解する。

また，昨今では chatGPT など生成 AI を用いたプログラミング・アシスト機能が発展しており，積極的な活用をすることで非常に効率よくプログラミングできる。残念ながら，現状では「ハルシネーション (もっともらしい嘘)」の回答が与えられることも少なくない。あくまでも真偽の判断をするのは人間の側に残されており，参考にはできるものの正答が得られるとは限らないことに注意する。

コマ主題細目

パソコンの基礎　スマートフォンやタブレットは「ファイル」を意識しない OS になっており，PC とそれらの境界はますますボーダレス化していくが，プログラミングを行う際はこれらの概念を知っておくことが重要である。PC の基本的な構造 (BIOS/OS/アプリケーション)，ファイルの位置 (パス，作業ディレクトリなど) などについての基礎知識を再確認する。

プログラミングの基礎　プログラミングにあたって重要なことは「思った通りに」動くのではなく，「書いた通りに」動くことである。ミススペルや大文字小文字の違いにも注意が必要である。コードのスペルチェックや補完機能を活用し，また 1 行ずつ確認しながら進めるといったプログラミングに関わる基本的な心構えについて理解する。

いくつかのプログラミング言語　R はプログラミング言語の一種であると言っても良い。プログラミング言語には他にも Python や Basic，C 言語などがある。これらの基本的な関係について理解するとともに，コンパイラとインタプリタという実行形式の違いについても理解する。この違いは今後確率的プロ

グラミング言語を利用する際の知識として生きてくる。

R 言語の基本的な挙動 R および RStudio の最新版を導入し，改めて基本的な活用法を理解する。簡単な四則演算やオブジェクトへの代入，パッケージや関数の利用などを演習的に復習する。tidyverse という概念およびパイプ演算子をはじめとするプログラミング記法についても演習を通じて学ぶ。

生成 AI の効率的な使い方 ChatGPT をコード生成時の補助役として活用する方法について理解する。ChatGPT は人間と違って，些細な質問を繰り返しても決して感情的になることはないため，質問の敷居は低い。一方で，間違えた答えを恥じることもなく，淡々と同じ間違いを重ねることもある。役割や方針を適切に与えた上で，問題を小分割してスモールステップでプログラミングを重ねていくことが基本である。また，結果が正しいかどうかは，必ず実際に動かしてユーザ側で判断する必要がある。

キーワード

- プログラミング言語
- コンパイラ・インタプリタ
- オブジェクト指向プログラミング
- 生成 AI

F.1.2 授業情報

■コマの展開方法 講義/遠隔/演習

予習・復習課題

■予習 事前に環境の準備をしておく必要がある。環境の準備についてはいくつかの方策があり，これについては導入資料を参照しながら準備しておくこと。なお，環境準備中に問題が生じた場合はいち早く教員か TA に相談し，実行できるようにしておくこと。

■復習 基本関数の使い方についての演習課題を課す。

F.2 プログラミングの実際

F.2.1 授業内容

F.2.2 授業内容

科目の中でのこのコマの位置づけ

プログラミングの本質は代入，反復，条件分岐である。この 3 点について理解し，基本的な書き方を学ぶ。実際に R でコードを書きながら，その挙動について確認する。最終的な到達段階として，Fizz-Buzz 問題や行列計算ができるコードを書くこととする。

コマ主題細目

高級言語の基本的な働き 高級言語と呼ばれるプログラミング言語の基本的な働きは，代入，反復，条件分岐である。R では<-や=で代入を，`for` や `while` で反復を，`if` や `if_else` で条件分岐を行う。これらの表現は言語間を通じて共通なものが多いため，その基本的な振る舞いを確認することは技術の一般化に役立つ。

オブジェクトの型 R は変数を宣言せずに利用できるところが初心者向けの長所ではあるが，R の中にストックされるオブジェクトの種類や型について正しく理解しておくことが，今後のデバッグの上で役にたつ。変数としての数値型 (Int,Double,Complex)，文字型 (Char,Factor)，論理型 (logical) や，オブジェクトのベクトル，行列，配列，リスト，データフレームについて理解する。

関数を作る コマンド (文) は連なってスクリプトとなる。反復されるスクリプトは関数化すると効率的である。ここでは R を用いた関数の定義の仕方について学ぶ。引数やデフォルトの値について知ることは，ヘルプを見る時の役にも立つ。また関数化はスパゲティ・コードになることを避ける，カプセル化された表記方法への第一歩であり，オブジェクト指向プログラミングへの第一歩でもある。

キーワード

- オブジェクトの型
- 代入
- 反復
- 条件分岐
- 関数

F.2.3 授業情報

■コマの展開方法 講義/遠隔/演習

予習・復習課題

■予習 RStudio におけるプロジェクト単位での管理など，R/RStudio の基本的な使い方については以後も逐一支持することはないので，これまでのことをしっかり復習し，本講義の予習とする。

■復習 反復計算の練習課題，条件分岐の練習課題など，複数の課題にしっかり取り組むこと。

F.3 確率関数

F.3.1 授業内容

科目の中でのこのコマの位置づけ

心理統計は個人差や偶然性を表すために確率の考え方が必須である。しかし確率の概念は数学的に高度かつ抽象的であり，直感的に理解することが難しい。概念的には，空間を標準化したときの相対的な大きさを指標化したもの，すなわち面積の一種と考えるとわかりやすいし，複雑な性質よりも最低限守るべき公理の方からアプローチした方が理解しやすい。ただしこのとき，，確率変数とその実現値の違いに注意する。とはいえこのコースの目的は，数学的内容を座学で学ぶのではなく，あくまでもプログラミングによる演習を通じて具体的に理解することにある。本講ではまず概念的な理解の復習からすすめる。なお以後，小杉他 (2023) を副読本としながら授業を進めていく。

コマ主題細目

確率の基礎 高校数学までの範囲で学ぶ確率は，すべての場合の数を全体としたうち，該当する事象の生じる回数を相対頻度で表すものであった。一方我々は，天気予報で降水確率が 30% というように，

確率的な表現を日常的にも用いる。天気予報は未来の予測であるから,「起こりうるすべての未来」を書き出して生じる天気の割合を表現できるはずもない。こうした不確実なものであっても,確率という言葉で表現できるようにコルモゴロフが公理として最も基本的な枠組みを提供した。ここでは認識論に深く立ち入ることはせず,確率という数字が従うべき法則について確認する。

→ Kruschke (2014 前田・小杉監訳 2017) の P.78–83, 小杉他 (2023) の P.61–62.

R の確率関数 R には確率関数がいくつか用意されており,接頭語 `d,p,q,r` と関数名からなる。ここでは正規分布を例に, `d,p,q-norm` がそれぞれ何を表しているかを確認する。確率は確率密度関数で表される領域の面積であり,積分によって計算されることを近似計算で理解する。

→ 小杉他 (2023) の P.66–77.

確率変数の期待値と分散 確率変数の期待値 (加重平均) と分散について,定義式を確認する。また正規分布においては期待値が位置パラメータ μ に,分散がスケールパラメータ σ^2 に一致することを確認する。

→ 小杉他 (2023) の P.83–86.

キーワード

- 確率変数と確率変数の実現値
- `dnorm,pnorm,qnorm,rnorm`
- 確率分布関数

F.3.2 授業情報

■コマの展開方法 講義/遠隔/演習

予習・復習課題

■予習 一年時の「データ解析基礎」ですでに確率については学んでいる。テキストや関連資料を見て改めて確率の概念を復習しておくことが,このセクションの予習になる。

■復習 R の正規分布を例に,関数の形とパラメータの関係,期待値と分散について学んできたが,正規分布以外の確率分布関数についても定義や性質を調べつつ, R の関数をつかって描画・計算して理解を進めておくと良い。

F.4 乱数による近似

F.4.1 授業内容

科目の中でのこのコマの位置づけ

確率分布についての数学的位置付け,理解について確認したところで,より具体的かつ体験的に理解するために,乱数を用いることで近似できることを学ぶ。計算機である以上,完全な乱数ではなく疑似乱数に過ぎないが,人間の人為的な乱数生成よりも,より「ランダムな」数字を作ることができる。さらに計算機の乱数はすでに確率分布に従うものが準備されているから,試しながら理解を深めることができる。また今後のベイズ

推定に向けて，名も無分布からの乱数であっても同様のことができることを理解する。

コマ主題細目

計算機の乱数　計算機の擬似乱数は，所謂「ガチャ」のような振る舞いをする数字のことである。一様乱数を例にサイコロの出目を再現するプログラムを通じて，擬似乱数の特徴を学ぶ。とくに乱数の種 (seed) を固定することで，再現可能であることにも注意する。

乱数による形状の近似　ここまでは確率密度関数を使っての話であったが，乱数を使うことで確率変数の実現値を使った近似が可能である。正規乱数を例に，まずは乱数の数を増やしながら，ヒストグラムの形状が徐々に正規分布に近づいていくことを確認する。また，一様乱数など他の確率分布の例でも確認してみる。

乱数による値の近似　`p,q-norm` を駆使して計算した面積の問題も，乱数を使って近似することができる。乱数は確率変数の実現値であり，R のもつ統計関数を使った集計が，積分などの計算に対応するものであることがわかる。期待値と分散の計算を乱数によって近似する。近似の精度は生成する乱数の数に依存することを理解する。

→ 小杉他 (2023) の P.83–86.

事後分布からの乱数に向けて　正規乱数や一様乱数，そのほかにも F, t, χ^2 分布など，聞いたことのある確率分布であれば乱数を生成して計算することで，数学的 (解析的) アプローチを避けることができる。また，特殊な例に思えるかもしれないが，とくに有名な名前のついていない確率分布であったとしても，そこから乱数が生成できれば，ここまで学んできたことを利用して近似計算が可能なはずである。この後のベイズ統計学的アプローチの理解に向けて，この論点を先に押さえておきたい。

キーワード

- 疑似乱数
- 乱数の種
- 正規乱数，一様乱数
- 事後分布

F.4.2 授業情報

■コマの展開方法　講義/遠隔/演習

予習・復習課題

■予習　前時の R の確率関数の使い方を十分に復習しておくこと。とくに積分の計算によって確率分布の一部の面積が算出できることを理解する。

■復習　小杉他 (2023) のコラム P.87–90 にある正規分布の再生性について，テキストに沿って確認しておくと良い。確率分布同士の計算も，解析的にではなく近似的に分析することができることを知ると，自信につながる。

F.5 一致性，不偏性，有効性，サンプルサイズ

F.5.1 授業内容

科目の中でのこのコマの位置づけ

ここからは推測統計学における確率分布の利用について理解する。すでに推測統計の基礎的な発想は一年時に学んでいるが，結果を追従するための理解ではなく，仕組みを理解するための一歩進んだ考え方を学ぶ。改めて母集団と標本の関係について理解をした上で，確率分布や数理統計の技術がどのように使われているかを理解する。小杉他 (2023) の第 4 章を参考にする。なお，有効性の理解についてはテキストを参照するにとどめる。

コマ主題細目

推測統計の基礎 母集団，標本という関係，また母数と標本統計量の区別を改めて確認する。R の `sample` 関数を使って，全体から一部を取り出す練習をすることでイメージがしやすくなるだろう。また標本統計量は標本を取るたびに変わりうる値，すなわち確率変数であるから，標本統計量の従う分布すなわち標本分布についても理解する必要があることを確認する。

→ 小杉他 (2023) の P.119–120.

一致性の理解 標本から母数を推測するとき用いられる標本統計量を推定量というが，この推定量が持つべき望ましい 3 つの性質について，シミュレーションを通じて学ぶ。まず一致性について，サンプルサイズが大きくなればなるほど母数に近づく性質である。正規乱数のサンプルサイズを変えてこれを確認する。

→ 小杉他 (2023) の P.121–128.

不偏性の理解 標本分散と不偏分散の違いは，学んだ当初はその区別がわかりにくかったかもしれない。ここでシミュレーションを通じてその理解を深める好機である。不偏性は推定量の期待値が母数に一致することであるから，正規乱数の標本分散と不偏分散の計算を反復し，その平均を取ることでいずれが推定量として適切であるかを確認する。

→ 小杉他 (2023) の P.128–137.

サンプルサイズと中心極限定理 ここまで正規乱数の例が多かったが，中心極限定理によると母集団が正規分布でなくてもその標本平均の分布は正規分布に近づく。t 分布を例にこのことを確認すると同時に，心理学実験などサンプルサイズが小さい場合にどのような注意が必要かについて学ぶ。

→ 小杉他 (2023) の P.140–144.

キーワード

- 母集団と標本
- 一致性
- 不偏性
- 有効性

- 中心極限定理

F.5.2 授業情報

■コマの展開方法 講義/遠隔/演習

予習・復習課題

■予習 一年時の「データ解析基礎」のテキストを復習しておくこと。第 16 講 (後期第一回目) が本時に対応する。

■復習 テキストには正規分布以外の確率分布を使って例が多く載っている。参考にしながら一致性・不偏性について理解を深め, また有効性についても一読しておくこと。

F.6 信頼区間

F.6.1 授業内容

科目の中でのこのコマの位置づけ

標本統計量が確率変数であるということは, 心理学実験や調査などで得られるデータとその統計量をそのまま一般化できないことにつながる。正規母集団から得られた標本平均は幸いにして, 母平均の推定量として好ましい性質を有しているが, それにしても標本平均すなわち母平均と考えるのは極端であると言わざるを得ない。標本統計量が確率変数であり, 確率分布 (標本分布) の性質がわかっているのなら, それを使って区間推定することが考えられる。ここでは区間推定についてもシミュレーションを通じて理解する。

コマ主題細目

点推定と区間推定 点推定と区間推定の区別を確認した上で, シミュレーションによって点推定が母数と一致する割合, 区間推定が母数を含む割合を確認する。前者がゼロであることから, 区間推定の必要性が理解でき, かつどのように区間を設定すれば良いかという問題に気づく。

→ 小杉他 (2023) の P.145–146

母分散が既知の場合 すでに「データ解析基礎」において標準正規分布を利用した区間推定について, 95% 区間に対応する ± 1.96 という数字を使った例を学んだ。今回はシミュレーションを通じて, 標本平均 $\pm 1.96\dfrac{\sigma}{\sqrt{n}}$ が正しく母平均を含んでいる回数をチェックし, このことを確認する。

母分散が未知の場合 同じく, 母分散が未知の場合は t 分布を使って区間推定をするのであった。ここでもシミュレーションを通じて, 区間推定が正しく母平均を含んでいる回数をチェックし, このことを確認する。可視化することでこのことが理解を深める。

→ 小杉他 (2023) の P.146–156.

キーワード

- 点推定
- 区間推定

- t 分布

F.6.2 授業情報

■コマの展開方法 講義/遠隔/演習

予習・復習課題

■予習 一年時の「データ解析基礎」のテキストを復習しておくこと。第 17 講 (後期第二回目) が本時に対応する。

■復習 t 分布の数式について，テキストのコラムを一読しておく。余裕があれば，テキスト 4 章後半の相関係数の信頼区間などにも目を向けるとよい。

F.7 帰無仮説検定のシミュレーション

F.7.1 授業内容

科目の中でのこのコマの位置づけ

標本統計量が確率変数であるなら，「二群の差」といった標本の群間差も確率的に変動するし，これに意味があるかないかといった判断も確率的になる。この判断が誤ったものになる水準をコントロールしようというのが帰無仮説検定の考え方である。このことに注意しておけば，帰無仮説検定の手続きも「米印を探す単調作業」でないことは理解できるだろう。改めて帰無仮説検定の手順を確認するとともに，t 検定とシミュレーションを通じて検定の精度と意義の理解を深める。なお小杉他 (2023) の第 5 章を参考にする。

コマ主題細目

帰無仮説検定の論理 帰無仮説と対立仮説，判断の手続などを再確認する。また，用語としてのタイプ 1 エラー，タイプ 2 エラー，検定力，および帰無仮説が従う分布 (帰無分布) についても解説を加える。

→ 小杉他 (2023) の P.187–191.

R による検定の実際 二群の平均値差の検定を例にとる。データを乱数から生成し，どのような手順で検定が行われているか，そのアルゴリズムを復習する。

→ 小杉他 (2023) の P.191–196.

タイプ 1 エラー確率のシミュレーション 心理学の研究実践において，タイプ 1 エラー，タイプ 2 エラーがどれぐらいだったか，検定力がどれぐらいあったかを知る術はない。しかしシミュレーションすることによって，何がどの程度起こりうるかを体験することができる。これをもって，自らの研究実践に対する統計的指標の意義を考える一助にしてもらいたい。また，t 検定においては Welch の方法がデフォルトで用いられるが，等分散性の仮定が成立しない場合はどのような問題が起きるかも確認できる。

→ 小杉他 (2023) の P.196–203.

キーワード

- 帰無仮説と対立仮説

- 帰無分布
- サンプルサイズ, タイプ 1 エラー, タイプ 2 エラー, 検定力
- 等分散性の仮定

F.7.2 授業情報

■コマの展開方法 講義/遠隔/演習

予習・復習課題

■予習 一年時の「データ解析基礎」のテキストを復習しておくこと。第 18–20 講が本時に対応する。

■復習 サンプルサイズや二群それぞれの分散の値をさまざまに変化させて, シミュレーションの結果がどのように変わるかいろいろ試してみよう。

F.8 QRPs とサンプルサイズ設計

F.8.1 授業内容

科目の中でのこのコマの位置づけ

ここでは帰無仮説検定の仮定を逸脱した場合に何が起こるか, 悪しき研究実践の例を仮想的に体験することでその問題を理解する。帰無仮説検定はデータに対して厳格なルールを決めてから実践するべきもので, いわばスポーツのゲームのような側面がある。ラフプレーをするとゲームが成立しないように, 科学におけるラフプレーは真実を見誤るという問題を引き起こす。とくにサンプルサイズや帰無仮説を事後的に変えることで, 本来コントロールされているはずのものがそうならなくなってしまう。これらについて, 小杉他 (2023) の第 6 章を参考に話を進める。

コマ主題細目

心理学の再現性問題 2000 年ごろから指摘され始めた, 心理学の再現性問題を紹介する。その一因として挙げられる心理統計法の誤用と QRPs(問題ある研究実践) の存在を知り, このことが科学としての心理学の根幹を揺るがす大問題であることを共有したい。資料として池田・平石 (2016) を用いる。

QRPs の実践 QRPs を仮想空間上で実践して, どのような問題が生じるかを見ていく。サンプルサイズを徐々に増やしていく (「もう少し頑張ってデータを取ろう！」) ことがなぜ悪いのか, 有意なところだけ報告する (帰無仮説の事後的な変更) などによって, タイプ 1 エラーの確率が制御できない様子を確認する。

→ 小杉他 (2023) の P.6–8 および P.225–232.

非心分布をつかったサンプルサイズ設計 以上のことから, 帰無仮説検定を行う際は事前にすべての設定・ルールを定めておく必要があることがわかる。実践者が恣意的に決められるのはサンプルサイズであるから, サンプルサイズの設計が重要になってくる。ここで対立仮説が従う分布として非心分布を導入し, これに基づくサンプルサイズ設計の例をシミュレーションで確認する。

→ 小杉他 (2023) の P.6–8 および P.238–224.

非心分布をつかわないサンプルサイズ設計 二群の平均値差の検定のような, 比較的簡単な実験計画に

おいてはサンプルサイズ設計の理論も単純なほうである。とはいえ，非心分布など慣れない確率分布の利用に戸惑うこともあろう。そこで多少力技的ではあるが，乱数生成を繰り返すことで実際どの程度の差・効果がどの程度の割合 (確率) で検出できるかを検証するシミュレーションを行う。この方法は非心分布がわからないときや，複雑な実験計画にであっても応用が可能である。

キーワード

- 再現性問題
- QRPs
- 非心分布
- 例数設計

F.8.2 授業情報

■コマの展開方法 講義/遠隔/演習

予習・復習課題

■予習 一年時の「データ解析基礎」のテキストを復習しておくこと。第 19 講が本時に対応する。

■復習 分散分析の例数設計については，テキストを参考に自ら実践することが望ましい。

F.9 確率的プログラミング言語

F.9.1 授業内容

科目の中でのこのコマの位置づけ

これまでは R にビルトインされている確率関数を使って，シミュレーションや検定のロジックを確認してきた。ここまでは，確率を反復試行におけるある実現値の出現割合と考えてきたことになる。この考えに基づく確率の定義は頻度主義的ともいわれるが，同じ確率を使った推論でも他の方法が存在する。ベイズ的確率の考えがそれで，この場合の推論は事前分布と確率モデルから導出される事後分布を活用して行われる。心理統計でよく用いられるモデルは，一様分布を事前分布とし，確率モデルは正規分布を仮定することがほとんどであり，事後分布も解析的に導出することができるが，ここではより一般的な確率モデルに応用することを考える。確率的プログラミング言語を持ちることで，尤度と事前分布を設定することで事後分布発生器を作ることができ，いかなる事後分布からでも乱数を生成することができる。乱数によって確率分布が近似できることはすでに学んだとおりである。ここでは専門の確率的プログラミング言語 Stan を導入し，以後の学習に備える。

コマ主題細目

ベイズ推論の基礎 推定法の一種，ベイズ推定はベイズの定理に基づいている。改めてベイズの定理を解説し，条件付き確率や尤度，事前分布，事後分布といった用語についても改めて確認する。

→Kruschke (2014 前田・小杉監訳 2017) の Pp104–123 ほか枚挙に遑がない

事後分布の生成 マルコフ連鎖モンテカルロ法は，事後分布の生成と乱数の生成が合体した計算技術であ

る。事後分布の関数の形はわからなくとも，そこから乱数を生成することで形状を近似し，分析することができる。これらの基本的な仕組みを理解する。

→Kruschke (2014 前田・小杉監訳 2017) の Pp.147–194

確率的プログラミング言語 Stan　マルコフ連鎖モンテカルロ法の演習にあたって，確率的プログラミング言語 Stan を導入する。Stan は変数の宣言が必要であること，ブロックに分割されていること，行の終わりにセミコロンを入れることなど，R 言語よりも C 言語に近い書き方が必要である。RStudio には Stan ファイルのサンプルも入っているので，これらを活用して簡単なコードを書いてみる。

→Kruschke (2014 前田・小杉監訳 2017) の Pp.407–425.

事後分布による解析　Stan を R から利用すると，結果として得られるオブジェクトに非常に多くの情報を含んでいることになる。ここでは cmdstanr の出力結果を確認し，またコンパクトに出力をまとめる関数を作りながら，事後分布をどのように可視化，記述するかについて学ぶ。

キーワード

- ベイズ推定
- 事前分布，事後分布
- 確率的プログラミング言語
- Stan

F.9.2　授業情報

■コマの展開方法　講義/遠隔/演習

予習・復習課題

■予習　Stan の導入は大学の PC 教室を利用するが，そうでなければローカルに環境を構築する必要がある。ローカル環境での利用を希望するものは，事前に調べて用意しておく。

■復習　Stan のサンプルコード，可視化の関数やパッケージなど，演習の分量が多いため，時間内に終わらなかったものは資料を参考に必ずフォローアップしておくこと。

F.10　モデリングの目から見た検定 1; 二群の平均値差

F.10.1　授業内容

科目の中でのこのコマの位置づけ

データ生成モデリングの観点を踏まえた上で，検定的アプローチとベイズ的アプローチの違いを学ぶ。

二群の平均値の差の検定，いわゆる対応のない t 検定の場合は，同一の正規分布から得られたデータに対して平均値の差があると判断して良いかどうかを判断するという枠組みであった。これらの前提と判断基準を確認し，それがデータ生成モデルの観点ではどのように表されるかを検証する。ここで結果が分布として推定されること，差があるかないかというのは二群の推定された平均値の差であることから，生成量を使って平均値の差を出力することを考える。ここで効果量に改めて目を向けるとその理解が進む。

コマ主題細目

t 検定の仮定 二群の平均値の差を検定するときは t 検定が利用されるが，データが正規分布から得られているという仮定，分散が同じであるという仮定などを踏まえて設計図を書き，これを Stan で表現することを考える。あらためて t 検定のやり方や結果と比べてみることで，モデリングがデータ生成メカニズムに注目していること，パラメータの推定を行なっていることなどが確認できるだろう。また等分散性の仮定を外す方法についてもすぐに応用ができる。

帰無仮説検定と二群の差の検定について，→ 山田･村井 (2004) や一年次の資料をもとに確認しておく。

差の分布 検定は推測に加えて判断を行なっていた，ということを改めて確認するとともに，帰無仮説検定では母平均の差をターゲットにしていたことを確認する。MCMC は母集団からの代表値であるので，推定された結果を使って差を表現することができる。これは R 側で得られたサンプルで行っても良いし，Stan の生成量を使っても良い。ここで `generated quantities` ブロックの考え方を導入し，平均値の差の分布を確認すること，帰無仮説検定が差の分布の一点についての仮説であったことを確認する。また一方が他方よりも大きくなる確率はどれぐらいかとか，一方と他方が c 以上に違っている確率はどれぐらいか，と言ったことが生成量を使って計算することができるようともいえる。

帰無仮説検定を省みる ここまでくると，帰無仮説検定のロジックや考え方について別の視点から見ることができるようになるであろう。まずは帰無仮説と対立仮説という対立のさせ方の不平等さである。帰無仮説は一点についての仮説であり，対立仮説はそれ以外であればなんでも良い，という非対称な関係になっていた。それを省みると，差があるかないかといった二値判断に陥ることがいかに危険であるかがわかるだろう。また量的な判断ができないことから，効果量を合わせて報告することが望ましいとされている。効果量とは，標準化された差の大きさのことであり，生成量を使って簡単に算出することができる。また方向性を持った検定について，片側･両側検定などで考えられてきたが，生成料を使えば自然にそれが検証できることがわかる。ただしこれらの検証の仕方は，今回のデータと仮定されたモデルという前提の上で成立する程度であって，過度な一般化にはならないように注意する必要がある。

キーワード

- t 検定
- 生成量
- 効果量
- 片側検定，両側検定

F.10.2 授業情報

■コマの展開方法 講義/遠隔/演習

予習・復習課題

■予習 Stan の基本的なブロック構成，設計図からコードに落とすやり方を確認しておこう。

■復習 データのサイズが変わるだけでなく，平均値の差，効果量が変化したときの t 検定の結果とベイズ推定の結果がどのように変わるのか，さまざまなケースを想定して「遊んで」みるとよい。加えて，そのほかの

仮説検定がどのようなデータ生成メカニズムで表現できるかを考えることは，次回以降の準備にもつながる。

F.11 モデリングの目から見た検定 2; パラメータの世界とデータの世界

F.11.1 授業内容

科目の中でのこのコマの位置づけ

データ生成メカニズムの観点から帰無仮説検定を省みた場合，その仮定やメカニズムを明記する必要があることから，拙速な解釈に陥る危険性を避けることができる。また事後予測分布を作ることで，柔軟な仮説を考えられることなども示された。

本時は，同じく事後予測分布を使いながら，パラメータでなくデータのレベルでの比較ができることに言及し，実質的に差があるとはどういうことであるかを考える。パラメータの世界，データの世界を分けて考えられるように注意を促す。まずは事後予測分布をみることで，モデルが現在のデータを正しく再現しているかを見ることで，視覚的にモデルの正しさが検証できることを確認する。その上で，新しく作られた分布の特徴から，閾上率や優越率などを計算することができる。これらはデータに基づく予測であるから，より具体的で実感を得やすい予測として使えるだろう。翻って，仮想データを生成して検証する，パラメータリカバリの手法を学ぶ。この方法では真値やサンプルサイズを自由に設定し検証できることから，例数設計に応用することが可能である。ベイズ推定をしない場合であっても，シミュレーションによる例数設計が有用であることを理解する。

コマ主題細目

事後予測分布　推定値をつかって，新たにデータを生成した場合どのようなことが言えるか。事後予測分布を描くことでモデルの正しさが確認できる。事前分布の特徴を反映した，事前予測分布についても触れる。

事前と事後の予測については →Lee・Wagenmakers (2013 井関訳 2017) の Pp.38–42 が詳しい。

データレベルの仮説　これまで考えてこられた仮説は，パラメータについての仮説であった。一方，事後予測分布が新しいデータを作っているのであれば，そこからデータレベルの仮説を考えることもできる。閾上率や優越率といった，データのレベルでの仮設を検証したり検討したりすることを考える。これらの視点は帰無仮説検定およびモーメント法による算出よりもわかりやすいかもしれないし，統計的に差があるということがどの程度意味のあることなのかを実感するのにも役立つだろう。

優越率，閾上率については，→ 豊田 (2016)，Pp.69–70.

パラメータ・リカバリ　事後予測分布は乱数発生による新しいデータの生成である。であれば乱数発生のアプローチは，理論に従う仮のデータを生成することができる，ということでもある。シミュレーションとして仮にデータを作ってみて，サンプルサイズがどの程度であればどの程度正確な推定ができるのか，と言った理論的な検証をすることができる。これはパラメータ・リカバリという試みでもあり，モデルが複雑になって行ったときに正しく機能するかどうかをチェックする方法でもある。また，サンプルサイズも自由に変えることができるのだから，どの程度のサンプルがあればどのような結果が得られるのか，と言ったシミュレーション，あるいは実験前のサンプルサイズ設計にもつながる。

モデリングの基礎的手順について，→ 松浦 (2016) の Pp.12–81.

キーワード

- 生成量
- パラメータ・リカバリ
- 事後予測分布
- 例数設計

F.11.2 授業情報

■コマの展開方法 講義/遠隔/演習

予習・復習課題

■予習 generated quantities ブロックの書き方について，数値を色々変えて確認しておくとよい。

■復習 さまざまなサンプルサイズ，効果量の仮想データを生成し，帰無仮説検定やベイズ法による推定を繰り返すことで，各手法の長所や短所を考えることができる。遊び心を持って，さまざまな状況生成して，実際に試してみること。

F.12 モデリングの目から見た検定 3；多群の平均値差を求めるモデル

F.12.1 授業内容

科目の中でのこのコマの位置づけ

ここまで generated quantities ブロックの活用で，事後分布，事後予測分布を生成できること，パラメータの世界，データの世界それぞれでの仮説が検証できることを見てきた。またパラメータリカバリの方法を見ることで，データ生成モデルを分析前に活用する方法についても見た。

続いて多群の平均値差を求めるモデルを考える。まずは一要因 3 水準の Between モデルから，3 つの平均値をバラバラに求めること，生成量から差分を計算することを考える。データやパラメータレベルでの仮説的な検証方法を再確認し，帰無仮説検定の枠組みで考えなければならなかったアルファ水準のインフレ問題が生じないことを，確率の考え方の違いに即して理解する。続いて差をモデルに組み込む方法を考える。ここでパラメータの数に制約をかける方法として transformed parameters ブロックの使い方を導入する。またパラメータの数の制約は，自由度の概念と深く関係していることへの洞察を得る。またパラメータリカバリのコードがリバースエンジニアリングのコードと同じもので，鏡合わせの関係にあることを確認する。続いて交互作用が含まれるモデルを考える。ここで制約からどれだけのパラメータが必要か，どのように組み上げるかを学ぶことができる。

コマ主題細目

要因計画モデル 一要因 3 水準 Between モデルを考える。対応のない二群の時のように，これは素直に 3 群のモデルとして表現できるし，群間の差を生成量として計算できることを再確認する。また検定と違って差の大きさを直接検証すること，確率的判断を含まないことから，アルファ水準のインフレ問題

に悩む必要がないことがわかる。この考え方は，確率の捉え方の違いにもつながることに留意する。

パラメータの変形と制約 3 群のうち，ある群を基準にした差分を直接パラメータとして推定するモデルを考えると，2 つの差分を計算することができる。またある群を基準におかなくとも，全体平均を基準に置くことができるが，その場合は差分のパラメータに制約をかける必要がある。これらの制約を含んだモデルを，`transformed parameters` ブロックを使って作ることを確認する。ここでパラメータの自由度について理解する。

モデルの洗練 技術的な問題であるが，多群モデルの場合は一般的に描画するためにも，整然データの形式に整えておくことが望ましい。書いたモデルの一般化という観点から，変数にできるところは変数にするなど，コードの洗練を試みる。ここで群を識別する変数を導入することは，今後の個人内反復測定モデルにも応用できる点であるので，しっかり理解する。

パラメータリカバリ これらのモデルのパラメータリカバリから，仮想データはデータ生成モデルを逆転させるだけで出来上がことがわかり，要因計画を裏側から眺めるような，新しい観点からの理解が進むと考えられる。X

キーワード

- 要因計画
- 整然データ
- 生成量
- パラメータリカバリ

F.12.2 授業情報

■コマの展開方法 講義/遠隔/演習

予習・復習課題

■予習 パラメータリカバリの必要性など，データ生成モデルのアプローチにおける標準的な手順を再確認する。また対応のない二群の平均値差の検定，要因計画の検定についてこれまでの復習をしておくことで，今回の内容の理解が深まるだろう。

■復習 要因数が増えた場合どのようになるか，またその都度 Stan モデルを書き換えなくても良くなるような一般的な書き方について，自分なりに試行錯誤することが望ましい。

F.13 確率的プログラミングの応用 1; 項目反応理論

F.13.1 授業内容

科目の中でのこのコマの位置づけ

ここまでで，ベイジアンモデリング・アプローチによって従来の統計モデルがどのように変わるかを見てきた。以後はアラカルト的に，確率的プログラミングの応用による柔軟なモデリング例のトピックスを取り上げる。最初に扱うのは，項目反応理論のモデリングである。尺度作成の文脈で，理論的概要は心理学データ解析応用 1 のシラバスやテキストを参考にしてもらいたいが，改めて確認するとともに確率的モデリングとして実装する。

確率モデルとして考えると，0/1 の反応に対するロジスティック回帰の応用であり，実装自体は既有知識の応用で可能であろう。コーディングのポイントとして，long 型データ (tidy data) にしておくことで欠損値が含まれる場合も対応できるようになることが挙げられる。

コマ主題細目

ロジスティックモデルの復習　項目反応理論を R で実行する場合は，`ltm` パッケージなど専用の関数群がすでに存在する。しかしここでは 0/1 の二値反応に対する確率分布である，ベルヌーイ分布のパラメータにモデルを組み込む形として，フルスクラッチでコードを書き直す必要がある。あらためてロジスティックモデルの概略を説明し，テストデータとの関係について理解する。

ロジスティック回帰モデルでの実装　ロジスティック回帰分析を拡張した 1PL,2PL,3PL モデルそれぞれを，`transformed parameters` ブロックで記述することを演習で学ぶ。

整然データでの分析　データを整然データの形にして分析することで，欠損値が含まれないデータセットを作って分析に応用することができる。ここでは個人と項目それぞれを識別する変数が必要になるが，これまで学んできた技術で十分対応可能であると考えられる。

キーワード

- 項目反応理論
- 1PL ロジスティックモデル
- 2PL ロジスティックモデル
- 3PL ロジスティックモデル
- 整然データ

F.13.2　授業情報

■コマの展開方法　講義/遠隔可/演習

予習・復習課題

■予習　これまでの知識や技術を組み合わせて問題に対応することになる。項目反応理論とロジスティックモデル，GLM におけるロジスティック回帰分析，データハンドリングにおける整然データの考え方など，これまでの資料に戻って復習しておくと良い。

■復習　自分で描いたモデルが R のパッケージが出す答えとどの程度一致するのかを確認しておこう。また欠損値がある被験者の被験者母数は，その確信区間が広くなると考えられる。なぜそうなるかを改めて考え，実際のデータ適用例で確認しておこう。

F.14 確率的プログラミングの応用 2; 変化点と折線回帰

F.14.1 授業内容

科目の中でのこのコマの位置づけ

コマ主題細目

変化点検出は，時系列的なデータの中に異なる 2 つの平均値を持つ群があることをモデリングする手法である。とくにある時点から異なる群に属するという系列的な意味があることと，変化点があるとすればどのあたりになるかという「変化点の位置的不明確さ」を確率分布で表現し，データから検出するという観点は，確率モデルの表現の自由さとデータとの接合を許す確率的プログラミング言語の面白さを味わうには最良の材料である。

まずは混合分布モデルのように，2 つの群を分類するモデルを再確認し，その上で時系列的なデータという既有知識から「変化点」という考え方の導入，モデリングへと繋げる。またデータによっては，一定の点を期に線形モデルの傾きが変わるような表現が可能なものがある。この変化点と回帰分析を融合させた，折線回帰モデルを考えることで，固定的なモデルを超えた柔軟なモデリングが可能であることを理解する。

ただしここで使うデータは時系列的なものであるから，一般的な回帰分析の前提であるサンプルの独立性がない。その意味で不適切なモデルであることに注意し，次回の時系列分析へと繋げる。

混合分布モデル データは可視化することが重要であり，見れば明らかに異なる状態の混合であることがわかる場合がある。具体例とともに可視化を行い，またこれまで学んだ混合分布モデルで表現できることを再確認する。ここで用いるデータは，小杉の体重記録データを用いる。

変化点検出 データの横軸が時系列的な意味を持つのであれば，時空を超えて 2 つの群が混合しているというのは不自然な前提である。そこで横軸に時系列的な意味を置くと，ある時点から状態が変化したものとして考えることができる。ここでその時点が「いつ」であるのかは不明であるが，わからないことを確率で表現するのが確率モデルのおもしろい点である。変化点を確率的パラメータとし，その前後で群が異なるというモデルは，変化点検出のモデリングと言われる。このモデリングはポリグラフ検査など，実践的な場面での利用価値も高い。

→Lee・Wagenmakers (2013 井関訳 2017) の Pp.59–61，松浦 (2016) の Pp.238-245

折線回帰 平均点の位置が変わるだけでなく，変化の傾向が明らかな場合は線形モデルを当てはめることができる。変化点の前後で傾きが変わるような線形モデルは，折線回帰とも呼ばれる。折線回帰モデルの実装については，変化点と回帰モデルを組み合わせたあとで，折れる点を繋げる数学的補正を加えたモデルへと修正する。最後に，説明変数が時点であることから回帰分析の前提として標本の独立性が担保されていない問題を指摘する。

キーワード

- 混合分布モデル
- 変化点
- 折線回帰
- 時系列分析

F.14.2 授業情報

■コマの展開方法 講義/遠隔可/演習

予習・復習課題

■予習 混合分布モデルの応用になるので，混合分布モデルの基本的な書き方や解析方法について，第 F.11 講を復習しておくことが望ましい。

■復習 折線モデルが応用できそうなデータを見つけて，自分なりに実践してみると理解が深まるだろう。とくに折れる点が複数あるモデルや，折れる点の数を検出するモデルへと拡張するなど，モデル展開の可能性をかんがえることもできる。

F.15 確率的プログラミングの応用 3; 状態空間モデル

F.15.1 授業内容

科目の中でのこのコマの位置づけ

前時に時系列的なデータを導入したが，時系列的な性質を無視したモデリングになっていた。時系列的な分析方法は，心理学においてもウェアラブル端末の利用や SNS など公的なデータを分析することなどにも利用できるため，非常に有用なものになりうる。しかしデータの特徴として非独立性の問題，周期性やトレンドの存在などがあり，周波数解析をおこなったり多次元の行列分解などが必要である。中でも状態空間モデルは比較的シンプルであり，とくにベイジアンアプローチで実装が容易になったと言えるだろう。

ここでは状態空間モデルの基本的な考え方を導入し，モデリングについて解説する。ここで状態と観測の分離を行い，とくに観測が行われていない点があっても分析できること，観測が行われていない点をパラメータとして保管することに言及する。観測が行われていない点が保管できるのであれば，未来の時点についても予測が可能になるということである。ホワイトノイズモデルでそれを行うと，確信区間が広がっていくことが観測される。そこでトレンドを入れたモデルにすることで，さらに予測の形を変えられることを学ぶ。

そのほかにも季節項など，時系列特有の情報を組み込めることや，二次元に展開することで空間データの分析にも応用できることに言及する。

コマ主題細目

時系列データの特徴 時系列的なデータがどのように得られ，どのようなシーンで利用可能であるかを概観する。ここで時系列データはサンプルの独立性が満たされていないという問題があるため単純な線形回帰は不適切であること，また周期性やトレンド，介入効果が出てくるまでの期間など独自に考えなければならないことがいろいろ含まれている。これまで研究されてきた領域や研究方法について概観する。

状態空間モデル さまざまな分析方法がこれまで考えてこられているが，状態空間モデルはその中でも比較的簡単な数理的構造を持ち，またベイジアンモデリングを利用することでかつての分析モデルが必要としたスムージングなどを，特段意識することなく分析できる。状態と観測というモデルの基本構造を提示し，これらがどのように実装可能かをみる。

→ 松浦 (2016) の Pp.229-235, 馬場 (2019) の第 5 部

欠損値の補間　観測時点には欠損が含まれることもあり，これをパラメータとして推定・補間することができる。またこれが可能であるということは，未来の時点を欠損値として考えれば予測ができることにもなる。プログラミングの工夫により，欠損を補間するようなコードの書き方を学ぶ。また単純なホワイトノイズモデルであればあまり予測として意味がないが，トレンドを考えることで時系列的な影響について考えることができる。

状態空間モデルの展開　状態空間モデルは，説明変数を加えた回帰モデルに応用したり，周期性をモデリングすることなども可能である。さらに時系列は一次元的であるが，二次元にも広げると空間分析にも利用が可能であることに言及する。これからの心理学は，時系列や空間など状況変数をより積極的に取り組んだモデルも利用するようになるだろう。

キーワード

- 状態空間モデル
- トレンド
- 補間

F.15.2　授業情報

■コマの展開方法　講義/遠隔可/演習

予習・復習課題

■予習　時系列データを，時間を独立変数とした回帰分析にすることでどういった問題があるのかについて，回帰分析や確率モデルの前提などを考えて振り返っておくことが良い予習になるだろう。

■復習　身の回りの身近ところからでもデータを取ることができるのが，時系列データのおもしろいところでもあるので，応用可能なデータを探して分析してみると良い。可能であれば今日からでも，時系列的なデータを取り始めると，長期的に見て非常に興味深い分析ができるようになるだろう。

引用文献

馬場 真哉 (2019). R と Stan ではじめる ベイズ統計モデリングによるデータ分析入門 講談社

Guilford, J.P. (1954). Psychometric Methods. McGraw-Hill Book Company.
(ギルフォード, J.P. 秋重 善治 (訳) (1959). 精神測定法 培風館)

南風原 朝和・芝 祐順 (1987). 相関係数および平均値差の解釈のための確率的な指標 教育心理学研究 , 35 (3), 259–265. https://doi.org/10.5926/jjep1953.35.3_259

池田 功毅・平石 界 (2016). 心理学における再現可能性危機：問題の構造と解決策 心理学評論 , 59 (1), 3–14. https://doi.org/10.24602/sjpr.59.1_3

Isaacson, Walter. (1995). Steve Jobs. JSTOR.
(アイザクソン, W. 井口 耕二 (訳) (2011). スティーブ・ジョブズ 講談社)

小杉 考司 (2019). R でらくらく心理統計 : RStudio 徹底活用 講談社

Kruschke, John K. (2014). Doing Bayesian Data Analysis. Elsevier.
(クルシュケ, J.K. 前田 和寛・小杉 考司 (監訳) 前田 和寛・小杉 考司・井関 龍太・井上 和哉・鬼田 崇作・紀ノ定 保礼・国里 愛彦・坂本 次郎・杣取 恵太・高田 菜美・竹林 由武・徳岡 大・難波 修史・西田 若葉・平川 真・福屋 いずみ・武藤 杏里・山根 嵩史・横山 仁史 (訳) (2017). ベイズ統計モデリング: R, JAGS, Stan によるチュートリアル 原著第 2 版 共立出版)

Lee, M. D. ・ Wagenmakers, Eric-Jan. (2013). Bayesian Cognitive Modeling: A Practical Course. Cambridge University Press.
(リー, M.D & ワゲンメーカーズ, E-J. 井関 龍太 (訳) (2017). ベイズ統計で実践モデリング: 認知モデルのトレーニング)

松浦 健太郎 (2016). Stan と R でベイズ統計モデリング 共立出版

西村 武 (1977). 主観評価の理論と実際 テレビジョン , 31 (5), 369–377. https://doi.org/10.3169/itej1954.31.5_369

Norretranders, Tor. (1999). The user illusiton. Penguin.
(ノーレットランダーシュ, T. 柴田 裕之 (訳) (2002). ユーザーイリュージョン――意識という幻想―― 紀伊國屋書店)

清水 裕士 (2016). フリーの統計分析ソフト HAD : 機能の紹介と統計学習・教育, 研究実践における利用方法の提案 メディア・情報・コミュニケーション研究, No. 1, 59–73.

清水 裕士 (2021). 心理学統計法 放送大学教育振興会

豊田 秀樹 (2016). はじめての 統計データ分析――ベイズ的〈ポスト p 値時代〉の統計学―― 朝倉書店

山田 剛史・村井 潤一郎 (2004). よくわかる心理統計 ミネルヴァ書房

シ (2016). 計算機言語のまとめノート 暗黒通信団

小杉 考司・紀ノ定 保礼・清水 裕士 (2023). 数値シミュレーションで読み解く統計のしくみ～R でためしてわかる心理統計 技術評論社

索引

ま

や

ら

漢字