実証会計・ファイナンスの勉強ノート

Soichi Matsuura

2026-01-01

Table of contents

第 1 章

はじめに

このノートは、大阪大学経済学研究科のファイナンス学者の笠原晃恭先生と会計学者の村宮克彦先生が書かれた「**実証会計・ファイナンス**」新世社に関する学習内容をまとめたものです。本書は、日経ストックリーグに参加する学生が、R を駆使して分析パートを作成できるように作られており、会計学とファイナンスの基礎知識を付けた後で、R の基本を学び、財務分析や投資理論を R で実装するための方法を学習できる、非常に有用な教科書です。ぜひ本書を手に取って、企業のデータ分析を自分の手で実践してみてください。

1.1 準備

各自の PC(Windows か Mac を想定) の好きな場所に `Kasahara_Muramiya` というフォルダを作成し，そこを作業ディレクトリにしていることを想定しています。この文章の意味が分からない人は，R の入門書を確認してください。作業ディレクトリの中に `data` というフォルダを作成し，そこにテキストで使うデータファイルを保存してある，という前提で進めます。

第 2 章

ファイナンス入門

```
pacman::p_load(
  tidyverse,
  ggthemes,
  ggpubr,
  plotly,
  patchwork,
  Rsolnp
)

mystyle <- list(
  # 1. ggthemes 自体の引数でフォントを指定します
  theme_economist_white(
    # gray_bg = FALSE,
    base_family = "HiraKakuProN-W3"
    ),
  scale_colour_economist(),

  # theme_calc(base_family = "HiraKakuProN-W3"),
  # scale_colour_calc(),

  # 3. その他の微調整（フォントサイズなど）
  theme(
    text = element_text(size = 12), # フォントファミリーは上で指定済みなの
    ↪  で省略可
    # 必要であれば個別の要素を調整
    # plot.title = element_text(hjust = 0.5),
    axis.title = element_text(size = 12)
```

```
  )
)
```

2.1　割引率

ファイナンス (finance) の大事な概念に**割引率** (discount rate) があります。これは、ある財の今の価値と将来の価値は異なっており、将来の価値を現在の価値 (これを**現在価値** (present value) という) に変換するための係数です。

たとえば，今の 100 万円が来年 110 万円になる世界において，

$$100 = f(110)$$

と表せる関数 f が存在するとします。このとき，110 万円を現在価値に変換するための係数が割引率です。具体的に，

$$f(110) = \frac{110}{1+r}$$

と表せるとき，r が割引率です。

$$100 = \frac{110}{1+r}$$
$$r = \frac{110}{100} - 1 = 0.1$$

この場合，割引率 r は $0.1(10\%)$ となります。割引率 10% の世界において，今の 100 万円は 1 年後の 110 万円と同じ価値をもつ，ということを意味しています。

2.1.1　確実なキャッシュ・フローに対する割引率

- **現在価値**：将来に発生するキャッシュフローの現時点における価値
- **時間価値**：将来と現在の価値の違い
- **無リスク金利** (risk-free rate)：安全資産へと投資したときのリターンで，投資時点でリターンの額が確定します。無リスク金利は**確率変数ではなく定数** (parameter) です。

上の式を年を T という記号で表して一般的に表現します。T 年後に**確実に**獲得できるキャッシュフローを CF_T で表し、無リスク割引率を R_F で表します[*1]。このとき現在価値 PV は以下のように計算されます。

$$PV = \frac{CF_T}{(1+R_F)^T}$$

[*1] 無リスク割引率は無リスク金利と同義で，全くリスクのない確実に手に入れられる利息のようなものです。

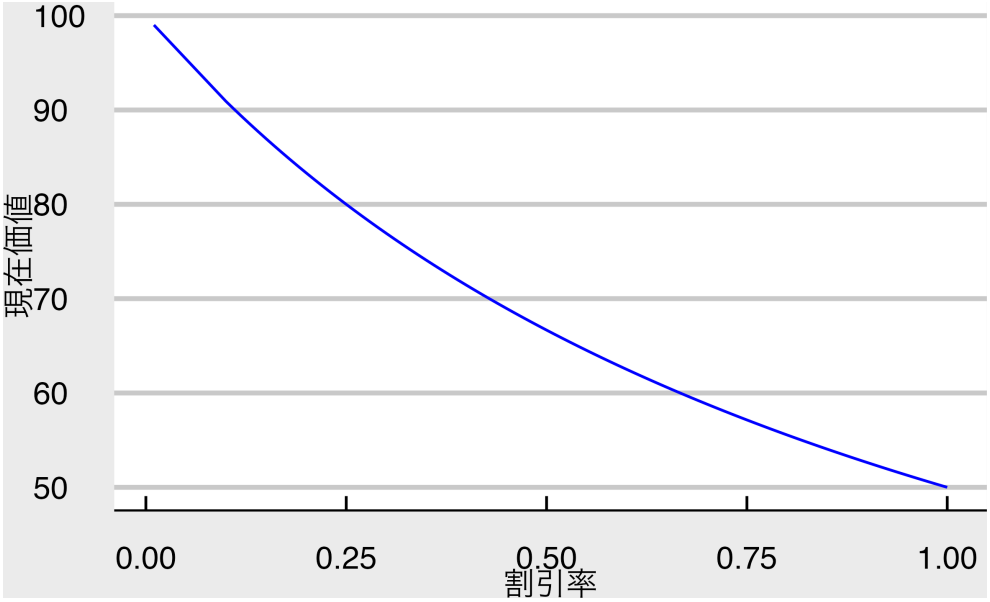
将来の確実なキャッシュ・フロー CF_T の現在価値は、割引率 R_F と将来受け取る時点である T に依存している。$CF_T = 100$ の場合、R_F と T の変化に応じて現在価値がどのように変化するのか確認する。

$T = 1$ として、横軸を割引率、縦軸を現在価値としたグラフが以下のものである。割引率が大きくなるにつれて現在価値が小さくなることが分かる。

現時点で CF_0 を 1 年間貯金する。利息 r は 0.1 とする。1 年後に受け取れるキャッシュフロー CF_1 は、貯金した元本 CF_0 と利息 $CF_0 \times 0.1$ となる。つまり、$CF_0 + CF_0 \times 0.1 = (1 + 0.1)CF_0$ である。
逆に、来年 CF_1 受け取るためには、今いくら貯金するかを考える。必要な貯金額を X とすると、$X \times (1 + 0.1) = CF_1$ となる。つまり $X = CF_1/(1.1)$ となる。これが現在価値である。

Listing 2.1 現在価値と割引率

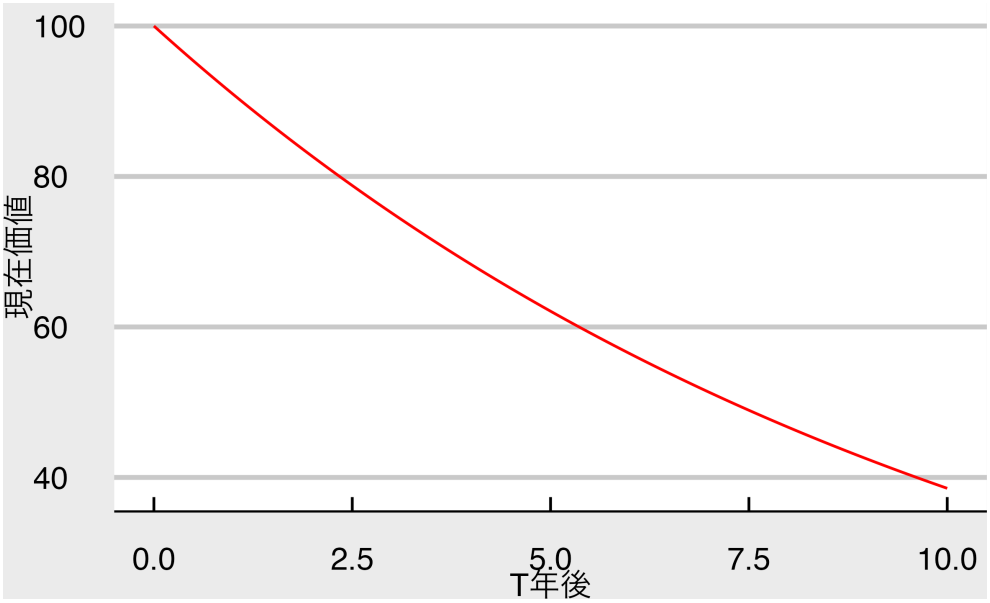
```
T = 1 # 1 年後に固定
R_F <- c(0.01, seq(0.1, 1, by = 0.01)) # 無リスク利子率
PV <- 100 / (1 + R_F)^T # 現在価値の計算
df <- data.frame(R_F, PV) # データフレーム作成
# 作図
df |>
  ggplot() + aes(x = R_F, y = PV) +
  geom_line(color = "blue") +
  xlab("割引率") + ylab("現在価値") + mystyle
```



次に、$R_F = 0.1$ に固定して、横軸を年 T、縦軸を現在価値 PV としてグラフが以下のものです。図を見ると，キャッシュ・フローを受け取る時点が遠くなるほど、現在価値が小さくなることがわかります。1 年後に受け取れる 100 万円と，10 年後に受け取れる 100 万円では，後者の方が価値が低い，というのは直観的ですよね。

Listing 2.2 現在価値と期間

```
R_F <- 0.1 # 無リスク利子率
T <- c(seq(0, 10, by = 0.1)) # 将来受け取る時点
PV <- 100 / (1 + R_F)^T # 現在価値の計算
df <- data.frame(T, PV) # データフレーム作成
# 作図
df |>
  ggplot() +
  aes(x = T, y = PV) +
  geom_line(color = "red") +
  xlab("T 年後") + ylab("現在価値") + mystyle
```



2.1.2　不確実なキャッシュ・フローに対する割引率

モデルに投資家の**リスクプレミアム** (risk premium; RP) を反映させることが割引率の 2 つ目の役割です。通常、将来キャッシュ・フローがいくらになるのか分からないのが普通であり、CF は様々な要因で変化する**確率変数** (random variable) であるため、現時点における**期待値** (expected value) で評価されます。

たとえば，1 年後に確実に 150 万円もらえる投資 A と，確率 50% で 200 万円，確率 50% で 100 万円もらえる投資 B があるとします。期待値で評価すると，投資 A の期待キャッシュフローは 150 万円，投資 B の期待キャッシュフローは $0.5 \times 200 + 0.5 \times 100 = 150$ 万円となり，同じ期待キャッシュフローをもつことになります。しかし，直観的に投資 B は 100 万円しか生み出さないリスクがある分，2 つの投資の価値を現在の価値で比較する

と，投資 A の方が価値が高いと思いませんか？

このリスクのある資産 B の割引率を $\tilde{R}$ と表すと，2 つの投資の現在価値は，次のようになります。

$$\frac{150}{1+R_F} \quad > \quad \frac{200\times 0.5+100\times 0.5}{1+\tilde{R}}$$

となるなら，

$$R_F < \tilde{R}$$

となります。この R_F と $\tilde{R}$ の差が**リスクプレミアム** (risk premium: RP) です。リスクプレミアムは、リスクのある資産に投資することに対する追加的な見返りを意味します。

$$RP \equiv \tilde{R} - R_F$$
$$R_F + RP = \tilde{R}$$

リスクプレミアムは割引率の調整で定量化されます。

$$PV_0 = \frac{\mathbb{E}_{\mathbb{0}}[CF_1]}{1+R_F+\underbrace{(\tilde{R}-R_F)}_{\text{RP}}} = \frac{\mathbb{E}_{\mathbb{0}}[CF_1]}{\underbrace{1+\tilde{R}}_{\text{リスク調整済み割引率}}}$$

割引率は時間価値やリスクプレミアムに関する定量的な情報を含むので、タイミングやリスクの異なるキャッシュフローを現在価値という同一の尺度で評価できます。

ここで、期待値をとる演算子 (operator) を $\mathbb{E}$ で表し、期待値をとる時点を添え字で表す。ここでは現時点 $t=0$ における期待値を $\mathbb{E}_0$ と表現している。たとえば，現時点を $t=0$ として，1 期先に起こりうる結果 X が 100 か 200 であることが分かっていて，それぞれの発生確率が 50% であったとする。この将来に起こりうる結果を現時点での情報を基に期待値をとる，ということは，

$$\mathbb{E}[X] = 0.5\times 100 + 0.5\times 200 = 150$$

となる。このように起こりうる結果と発生確率を掛けて足したものを**期待値**という。

2.1.3 NPV 法

先ほど学習した割引現在価値を判断の基準に，投資意思決定を行う方法を **NPV 法** (Net Present Value Method) といいます。いま，投資するかどうかを決めるタイミングを時点 0($t=0$) とします。キャッシュフローは CF_t で表し，マイナスなら支出，プラスなら収入を意味します。将来キャッシュフローは不確実であるため，現時点での期待値 $\mathbb{E}_{\mathbb{0}}[CF_t]$ で評価します。ちなみに CF_0 は時点 0 での投資なので，マイナスとなります。

投資時点から T 年間にわたって毎年 $\mathbb{E}[CF_t], \quad t=1,\ldots,T$ の期待キャッシュフローが生み出されるなら、それらを割引率 $1+\tilde{R}$ で現在価値に直して足し合わせた値 (つまり NPV) が、現時点で評価したプロジェクトの成果となります。

NPV はそのプロジェクトから発生するすべてのキャッシュフローの現在価値として解釈できます。コーポレートファイナンスでは，

- NPV がゼロ以上のプロジェクトは投資を実行
- NPV が負のプロジェクトは投資を見送る

ことを推奨しています。NPV 法を一般的に表現すると次のようになります (p.48)。

$$\begin{aligned} NPV_0 &= \frac{CF_0}{(1+\tilde{R})^0} + \frac{\mathbb{E}[CF_1]}{(1+\tilde{R})^1} + \cdots + \frac{\mathbb{E}[CF_T]}{(1+\tilde{R})^T} \\ &= CF_0 + \sum_{t=1}^{T} \frac{\mathbb{E}[CF_t]}{(1+\tilde{R})^t} \end{aligned}$$

$x^0 = 1$ であることに注意してください。

2.1.4　配当割引モデル

NPV 法の考え方を株式投資に適用することもできます。株式の保有期間を T 年とし、時点 t における株価を P_t で表します。株式から生み出される将来キャッシュ・フローは、

- 一株当たり配当と，
- 売却時点での売却損益

であり、t 時点の一株当たり配当を D_t，売却時点での一株当たり売却損益を $P_T - P_{T-1}$ と表します。将来配当や将来売却損益は確率変数であるため期待値で考え、それを割引率 $\tilde{R}$ で割り引くことで、一株当たりの株式価値を求めます。

$$\begin{aligned} P_0^* &= \frac{\mathbb{E}_0[D_1]}{1+\tilde{R}} + \frac{\mathbb{E}_0[D_2]}{(1+\tilde{R})^2} + \cdots + \frac{\mathbb{E}_0[D_\infty]}{(1+\tilde{R})^T} + \frac{\mathbb{E}_0[P_T] - P_0}{(1+\tilde{R})^T} \\ &= \sum_{t=1}^{T} \frac{\mathbb{E}_0[D_t]}{(1+\tilde{R})^t} + \frac{\mathbb{E}_0[P_T] - P_0}{(1+\tilde{R})^T} \end{aligned}$$

十分長い期間 $T \to \infty$ を考えると，売却損益の現在価値はゼロに近づくため，株式の理論価値 P_0^* は以下のように表せます。

$$P_0^* = \sum_{t=1}^{\infty} \frac{\mathbb{E}_0[D_t]}{(1+\tilde{R})^t}$$

これが**配当割引モデル** (Dividend Discount Model: DDM) です。将来配当の予測値データがあれば，現時点における理論上の株価を計算することができます。

2.1.5　ゴードン成長モデル

配当割引モデルは理論上は正しいのですが、将来の配当を無限に予測することは現実的ではありません。そこで、将来の配当が一定の割合で成長していくという仮定を置くことで、将来配当の予測を簡略化する方法があります。より具体的に，期待一株当たり配当が**一定の割合で成長**していくと仮定した割引配当モデルを**ゴードン成長モデル**という。

直近の実現した一株当たり配当を D_0 と置き、将来にわたってこの配当の期待値が一定割合 $G\%$ で成長していくと仮定する。つまり 1 時点先の配当額 D_1 が $(1+G)D_0$ となる、と仮定する。すると、t 時点先の配当額 $\mathbb{E}[D_t]$ は、成長率 G を用いた複利計算により、

$$D_t = (1+G)^t D_0$$

で表すことができます。一定割合で配当額が成長する株式の理論価値 P_0 を配当割引モデルで計算すると，

$$\begin{aligned} P_0 &= \frac{(1+G)D_0}{(1+\tilde{R})} + \frac{(1+G)^2 D_0}{(1+\tilde{R})^3} + \frac{(1+G)^3 D_0}{(1+\tilde{R})^3} + \cdots \\ &= \sum_{t=1}^{\infty} \frac{(1+G)^t D_0}{(1+\tilde{R})^t} \\ &= \frac{(1+G)D_0}{\tilde{R}-G} \end{aligned}$$

2 本目の式から 3 本目の式への計算で、**等比級数の和の公式**を利用している。

Tip

初項 a，公比 r の等比数列

$$a, ar, ar^2, ar^3, \ldots, ar^{n-1}, ar^n, \ldots$$

がある。この等比数列の和を S_n で表す。

$$S_n = a + ar + ar^2 + \cdots + ar^{n-1} + \cdots$$

両辺に r を乗じると，

$$rS_n = ar + ar^2 + \cdots + ar^{n-1} + ar^n + \cdots$$

となる。そして，$S_n - rS_n$ を計算すると，

$$\begin{aligned} S_n - rS_n &= a \\ (1-r)S_n &= a \\ S_n &= \frac{a}{1-r} \end{aligned}$$

上のゴードン成長モデルの初項は $(1+G)D_0/(1+\tilde{R})$，公比は $(1+G)/(1+\tilde{R})$ なので，

$$\begin{aligned} S_n &= \frac{\frac{(1+G)D_0}{(1+\tilde{R})}}{1-\frac{1+G}{1+\tilde{R}}} \\ &= \frac{\frac{(1+G)}{(1+\tilde{R})}D_0}{\frac{(1+\tilde{R})-(1+G)}{1+\tilde{R}}} \\ &= \frac{1+G}{\tilde{R}-G}D_0 \end{aligned}$$

ただし，$\tilde{R} \neq G$ の場合のみである。

2.1.6　割引率と期待リターンの関係

割引率 R は投資家が将来キャッシュフローを購入するにあたって最低限要求する期待リターン (**要求収益率**) とも解釈できます。これくらいリスクのある投資をするのだから，リスクのない資産への投資よりも高いリターンを要求する，という考えを反映しています。

2.2　平均分散アプローチ入門

個々の投資家にとって最適となる証券の組み合わせの比率を決めることを**最適ポートフォリオ選択**といいます。ポートフォリオの価値をリターンの期待値と分散で評価する考え方を**平均・分散アプローチ**といいます。

2.2.1　ポートフォリオのリスクとリターン

1. 投資対象が銘柄 A と銘柄 B の 2 銘柄のみが投資対象であるケース
2. 各銘柄への投資割合を w_A と w_B
3. $w_A + w_B = 1$

記号の定義は以下の通りです。

- 元本 X
- 投資銘柄 A のリターン $1 + R_A$
- 投資銘柄 B のリターン $1 + R_B$

> **Note**
>
> リターンの定義 $(P_t - P_{t-1})/P_{t-1} = P_t/P_{t-1} - 1$ は**ネット・リターン** (net return) と呼ばれるものである。これにたいして，元本も含めたリターン $1 + R_t = P_t/P_{t-1}$ は**グロス・リターン** (gross return) という。

元本 X のうち w_A 分だけ銘柄 A に投資すると、1 年後に期待値で $\mathbb{E}[X \times w_A \times (1+R_A)]$ になります。手元現金全額を銘柄 A と B に振り分けると、

$$\begin{aligned}(1 + R_A)w_A X + (1 + R_B)w_B X &= w_A X + w_A R_A X + w_B X + w_B R_B X \\ &= \left[\underbrace{(w_A + w_B)}_{\text{定義より}=1} + (w_A R_A + w_B R_B)\right] X \\ &= (1 + w_A R_A + w_B R_B)X\end{aligned}$$

となります。この **1 年後の価値と初期投資額の比**としてこのポートフォリオのリターン

$1 + R_P$ を計算します。

$$
\begin{aligned}
1 + R_P &= \frac{\overbrace{(1 + w_A R_A + w_B R_B)X}^{\text{将来時点の評価額}}}{\underbrace{X}_{\text{初期投資}}} \\
&= 1 + w_A R_A + w_B R_B \\
&= 1 + w_A R_A + w_B R_B \\
R_P &= w_A R_A + w_B R_B
\end{aligned}
$$

ポートフォリオ構築時には、各銘柄の実現リターンはわからないので (つまり確率変数)、かわりに期待値や分散を評価します。銘柄 A と銘柄 B のネット・リターンをそれぞれ R_A と R_B で表すと，期待値，分散，標準偏差は次のようになります。

- 期待値 $\mathbb{E}[R_A] = \mu_A$，$\mathbb{E}[R_B] = \mu_B$
- 分散 $\mathbb{V}[R_A] = \sigma_A^2$，$\mathbb{V}[R_B] = \sigma_B^2$
- 共分散 $\mathbb{C}\mathrm{o}\mathbb{V}[R_A, R_B] = \sigma_{AB}$
- 相関係数 $\rho = \frac{\sigma_{AB}}{\sigma_A \sigma_B}$

$$
\begin{aligned}
\rho &= \frac{\sigma_{AB}}{\sigma_A \times \sigma_B} \\
\sigma_{AB} &= \rho \sigma_A \sigma_B
\end{aligned}
$$

と表せます。

上の式より、ポートフォリオのリターンは $R_P = w_A R_A + w_B R_B$ なので、ポートフォリオのリターンの期待値 $\mathbb{E}[R_P] = \mu_P$ も各銘柄の期待リターンの加重平均となります。ここで投資割合 w_A と w_B は定数であり，R_A, R_B は確率変数です。

$$
\begin{aligned}
\mathbb{E}[R_p] = \mu_P &= \mathbb{E}[w_A R_A + w_B R_B] \\
&= w_A \mathbb{E}[R_A] + w_B \mathbb{E}[R_B] \\
&= w_A \mu_A + w_B \mu_B
\end{aligned}
$$

つぎに，ポートフォリオのリターン R_P の分散 $\mathbb{V}[R_P] = \sigma_P^2$ は各銘柄の分散及び相関係数を用いて計算できます。

$$
\begin{aligned}
\mathbb{V}[R_P] = \sigma_P^2 &= \mathbb{V}[w_A R_A + w_B R_B] \\
&= w_A^2 \mathbb{V}[R_A] + w_B^2 \mathbb{V}[R_B] + 2 w_A w_B \mathbb{C}\mathrm{o}\mathbb{V}[R_A, R_B] \\
&= w_A^2 \sigma_A^2 + w_B^2 \sigma_B^2 + 2 w_A w_B \sigma_{AB} \\
&= w_A^2 \sigma_A^2 + w_B^2 \sigma_B^2 + \underbrace{2 w_A w_B \rho \sigma_A \sigma_B}_{\text{ここの}\rho\text{の符号が重要}}
\end{aligned}
$$

確率変数の分散

確率変数 X と Y の線形結合 $aX+bY$ の分散は次のように計算できます。

$$\begin{aligned}\mathbb{V}(aX+bY) &= \mathbb{E}\left[\{(aX+bY)-\mathbb{E}[aX+bY]\}^2\right] \\ &= \mathbb{E}\left[\{(aX+bY)-(a\mu_X+b\mu_Y)\}^2\right] \\ &= \mathbb{E}\left[\{a(X-\mu_X)+b(Y-\mu_Y)\}^2\right] \\ &= \mathbb{E}\left[a^2(X-\mu_X)^2+2ab(X-\mu_X)(Y-\mu_Y)+b^2(Y-\mu_Y)^2\right] \\ &= a^2\mathbb{E}\left[(X-\mu_X)^2\right]+2ab\mathbb{E}\left[(X-\mu_X)(Y-\mu_Y)\right]+b^2\mathbb{E}\left[(Y-\mu_Y)^2\right] \\ &= a^2\mathbb{V}(X)+b^2\mathbb{V}(Y)+2ab\mathrm{Cov}(X,Y)\end{aligned}$$

ポートフォリオ P の分散 $\mathbb{V}(R_P)$ は，各銘柄の分散に投資割合の二乗を乗じたものに，各銘柄のリターンの相関関係部分を加えたものとなっていることが分かります。つまり，この 2 銘柄のリターンの相関係数 ρ に応じて，ポートフォリオ P の分散が大きくなるかどうかが決まる，ということです。

2.2.2　分散投資のメリット

保有比率 (w_A, w_B) を変化させたときのポートフォリオの μ_P と σ_P がどのように変化するかを確認しましょう。分散は非負の値をとることに注意してください。

$$\begin{aligned}\mathbb{V}[R_P] = \sigma_P^2 &= w_A^2\sigma_A^2+w_B^2\sigma_B^2+2\rho w_A w_B\sigma_A\sigma_B \\ &= (w_A\sigma_A+w_B\sigma_B)^2-2w_Aw_B\sigma_A\sigma_B+2\rho w_Aw_B\sigma_A\sigma_B \\ &= \underbrace{(w_A\sigma_A+w_B\sigma_B)^2}_{>0}-\underbrace{2(1-\rho)w_Aw_B\sigma_A\sigma_B}_{\text{ここ重要}}\end{aligned}$$

$0 \le w_A \le 1$ かつ $0 \le w_B \le 1$，$w_A+w_B=1$ のとき、

$$2(1-\rho)w_Aw_B\sigma_A\sigma_B \ge 0$$

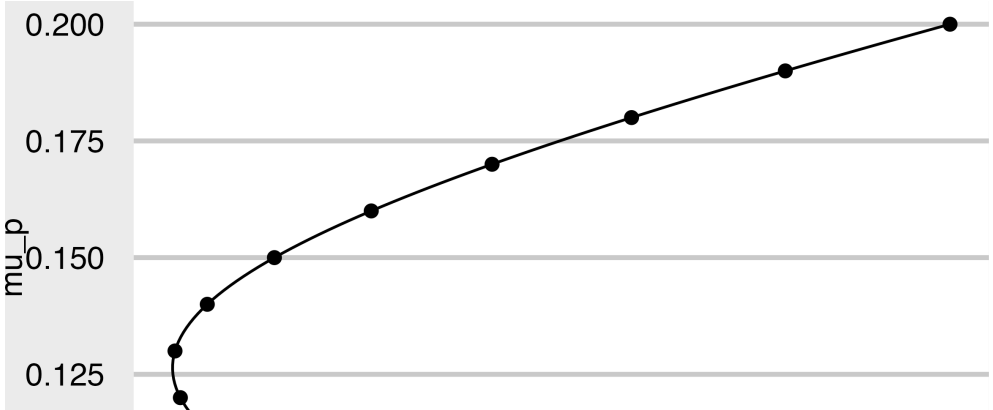
となります。$\rho=1$ のときのみ等号で成立します。したがって、$-1 \le \rho < 1$ のとき、$(1-\rho)w_Aw_B\sigma_A\sigma_B > 0$ となり，次の不等式が成立します。

$$\begin{aligned}&\sigma_P^2 = (w_A\sigma_A+w_B\sigma_B)^2-\underbrace{2(1-\rho)w_Aw_B\sigma_A\sigma_B}_{>0} \\ \iff &\sigma_P^2 \le (w_A\sigma_A+w_B\sigma_B)^2 \\ \iff &\sigma_P \le \underbrace{w_A\sigma_A+w_B\sigma_B}_{\text{リスクの加重平均}}\end{aligned}$$

となり，ポートフォリオのリスクを表す標準偏差 σ_P が銘柄 A と B の標準偏差の加重平均 $w_A\sigma_A+w_B\sigma_B$ 以下になることがわかります。これを**分散投資効果**という。

Listing 2.3 分散投資効果 p.65

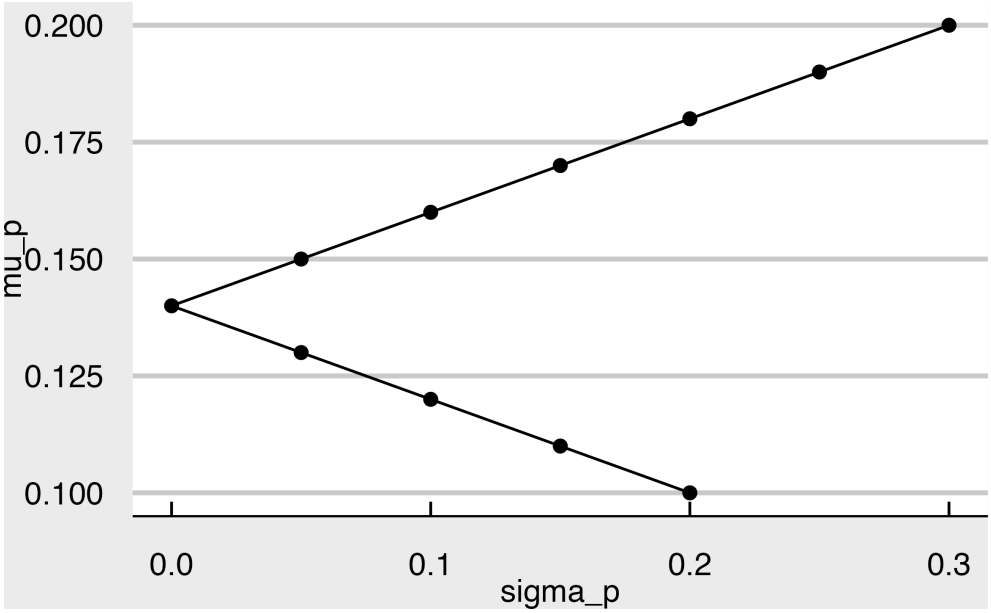
```
# 設定
rho = 0.2
mu_A <- 0.1
sigma_A <- 0.2
mu_B <- 0.2
sigma_B <- 0.3
# 保有費率
wa <- seq(0,1,by = 0.01)
wb <- 1 - wa

df <- tibble(
  mu_p = wa * mu_A + wb * mu_B,
  sigma_p = sqrt(wa^2 * sigma_A^2 + wb^2 * sigma_B^2 + 2 * rho * wa *
   ↪  wb * sigma_A * sigma_B),
  label= c(
  "0,1", rep("", 9),
  "0.1,0.9",rep("", 9),
  "0.2,0.8",rep("", 9),
  "0.3,0.7",rep("", 9),
  "0.4,0.6",rep("", 9),
  "0.5,0.5",rep("", 9),
  "0.6,0.4",rep("", 9),
  "0.7,0.3",rep("", 9),
  "0.8,0.2",rep("", 9),
  "0.9,0.1",rep("", 9),
  "1,0")
  )

ggplot(df) + aes(y = sigma_p, x = mu_p) + geom_line() +
 ↪  geom_point(data = filter(df, label != ""), aes(label = label),
 ↪  size = 2) + coord_flip() + mystyle
```



完全な負の相関 ($\rho = -1$) であるの場合、$\sigma_P^2 = 0$ のポートフォリオを構築できる。確認のため、2 銘柄の価格が完全な負の相関 $\rho = -1$ をもつとき、ポートフォリオ P のリスク σ_P は

$$\begin{aligned}
\mathbb{V}[R_P] = \sigma_P^2 &= (w_A\sigma_A + w_B\sigma_B)^2 - 2(1-(-1))w_Aw_B\sigma_A\sigma_B \\
&= (w_A\sigma_A + w_B\sigma_B)^2 - 4w_Aw_B\sigma_A\sigma_B \\
&= w_A^2\sigma_A^2 + w_B^2\sigma_B^2 + 2w_Aw_B\sigma_A\sigma_B - 4w_Aw_B\sigma_A\sigma_B \\
&= w_A^2\sigma_A^2 - 2w_Aw_B\sigma_A\sigma_B + w_B^2\sigma_B^2 \\
&= (w_A\sigma_A - w_B\sigma_B)^2 \\
\sigma_P &= |w_A\sigma_A - w_B\sigma_B|
\end{aligned}$$

以下のように，$w_A\sigma_A = w_B\sigma_B$ となるように w_A と w_B を選べば、$\sigma_P = 0$ となるポートフォリオを作れる。



このケースでは、$\sigma_A = 0.2$、$\sigma_B = 0.3$ となっているため、$0.2w_A = 0.3w_B$ となる保有割合を考える。

$$\begin{aligned}
0.2w_A &= 0.3w_B \\
0.2w_A &= 0.3(1-w_A) \\
0.2w_A &= 0.3 - 0.3w_A \\
0.5w_A &= 0.3 \\
w_A &= 0.6
\end{aligned}$$

となるため、銘柄 A に 60 %、銘柄 B に 40% を投資することで、リスクゼロで期待リターン $0.6 \times 0.1 + 0.4 \times 0.2 = 0.14$ を獲得することができる。

2.2.3　空売りの効果

空売り (short sale) とは、1. 保有していない証券を誰か (普通は証券会社) から借りてきて売却し、2. 一定期間後に買い戻して元の持ち主に返却する取引をいい、**値下がりから**

Listing 2.4 完全な負の相関の分散投資効果 p.65

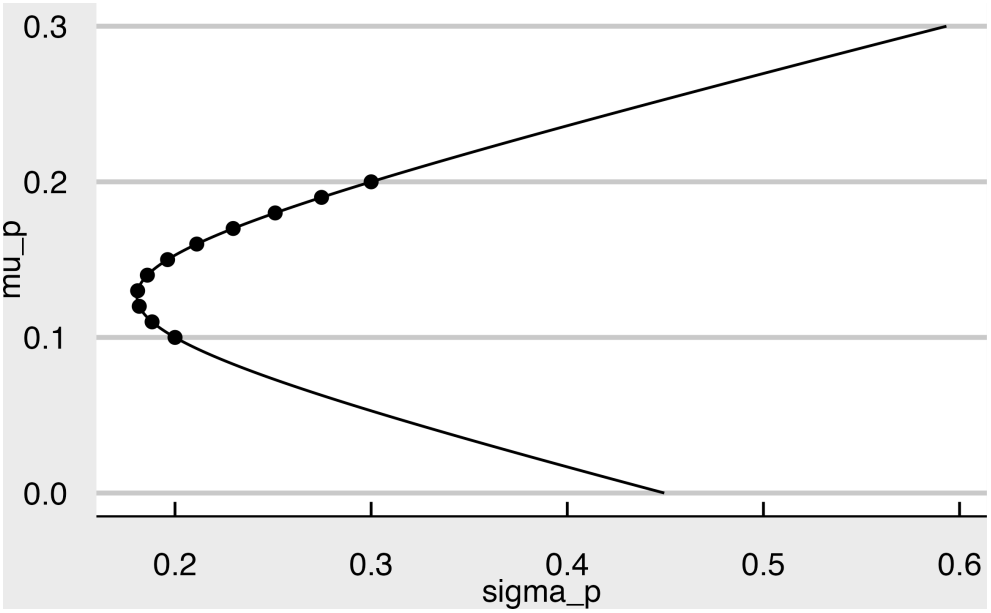
```
mu_A <- 0.1
sigma_A <- 0.2
mu_B <- 0.2
sigma_B <- 0.3

wa <- seq(0,1,by = 0.01)
wb <- 1 - wa



df <- tibble(
  mu_p = wa * mu_A + wb*mu_B,
  sigma_p = abs(wa*sigma_A - wb*sigma_B),
  label= c(
  "0,1", rep("", 9),
  "0.1,0.9",rep("", 9),
  "0.2,0.8",rep("", 9),
  "0.3,0.7",rep("", 9),
  "0.4,0.6",rep("", 9),
  "0.5,0.5",rep("", 9),
  "0.6,0.4",rep("", 9),
  "0.7,0.3",rep("", 9),
  "0.8,0.2",rep("", 9),
  "0.9,0.1",rep("", 9),
  "1,0")
)
ggplot(df) + aes(y = sigma_p, x = mu_p) + geom_line() +
 ↪  geom_point(data = filter(df, label != ""), aes(label = label),
 ↪  size = 2) + coord_flip() + mystyle
```

利益を得る。空売りを行う投資家をショートセラー (short seller) という。

いままでは、$0 \leq w_A, w_B \geq 1$ $という制約を置いていたが、この制約をはずして、$w_A + w_B = 1$ のみを課す。つまり $w_A < 0$ や $w_B < 0$ が空売りを表す。

```
rho = 0.2
mu_A <- 0.1
sigma_A <- 0.2
mu_B <- 0.2
```

```
sigma_B <- 0.3
# 保有費率
wa <- seq(-1,2,by = 0.01)
wb <- 1 - wa

df <- tibble(
  mu_p = wa * mu_A + wb*mu_B,
  sigma_p = sqrt(wa^2 * sigma_A^2 + wb^2 * sigma_B^2 + 2 * rho * wa *
   ↪  wb * sigma_A * sigma_B),
  label= c(
  rep("",100),
  "0,1", rep("", 9),
  "0.1,0.9",rep("", 9),
  "0.2,0.8",rep("", 9),
  "0.3,0.7",rep("", 9),
  "0.4,0.6",rep("", 9),
  "0.5,0.5",rep("", 9),
  "0.6,0.4",rep("", 9),
  "0.7,0.3",rep("", 9),
  "0.8,0.2",rep("", 9),
  "0.9,0.1",rep("", 9),
  "1,0" , rep("",100)
  )
)
ggplot(df) + aes(y = sigma_p, x = mu_p) + geom_line() +
 ↪  geom_point(data = filter(df, label != ""), aes(label = label),
 ↪  size = 2) + coord_flip() + mystyle
```



2.2.4 安全資産の導入

安全資産の購入についても考える。

安全資産 F のリターンを R_F、ポートフォリオ P の保有比率を w_F とおく（安全資産のリターンは確実に実現する成果なので，確率変数を表すチルダをつけてない)。以降は，確率変数にはなるべくチルダをつける。銘柄 A と銘柄 B と安全資産の投資割合をそれぞれ w_A, w_B, w_F で表す。

まず銘柄 A と安全資産の 2 資産からなるポートフォリオを考える。つまり各資産への投資割合を $w_A + w_F = 1, \quad w_B = 0$ とする場合を考える。この安全資産と銘柄 A からなるポートフォリオ P の (ネット) リターン $\tilde{R}_P$ は，

$$\tilde{R}_P = w_F R_F + w_A \tilde{R}_A$$

となり，このポートフォリオの期待値は，

$$\begin{aligned}
\mu_P = \mathbb{E}[R_P] &= \mathbb{E}[w_F R_F + w_A \tilde{R_A}] \\
&= w_F R_F + w_A \mathbb{E}[\tilde{R_A}] \\
&= w_F R_F + w_A \mu_A \\
&= (1 - w_A) R_F + w_A \mu_A \\
&= R_F - w_A R_F + w_A \mu_A \\
&= R_F + w_A (\underbrace{\mu_A - R_F}_{\text{リスクプレミアム}})
\end{aligned}$$

となる。もちろん安全資産のリターンは確率変数でないので，期待値をとってもそのままである。$\mu_A - R_F$ はリスク資産である銘柄 A の**リスクプレミアム**を表している。通常，リスクのある資産の期待リターンは安全資産のリターンより大きいため，リスクプレミ

アムは正の値となる。したがって，リスク資産への投資割合 w_A を 1 単位増加させれば，$\mathbb{E}[R_P]$ はリスクプレミアム分増加する

つぎに，ポートフォリオのリスクを表す標準偏差 σ_P とリターンの関係は次式で表せる。まず安全資産のリスクはゼロであるため，リスク資産の銘柄Aを保有する分だけリスクが生じる。

つぎに，ポートフォリオのリスクを表す標準偏差 σ_P とリターンの関係は次式で表せる。まず安全資産のリスクはゼロであるため，リスク資産の銘柄Aを保有する分だけリスクが生じる。

$$
\begin{aligned}
\sigma_P^2 = \mathbb{V}[R_P] &= \mathbb{V}[w_F R_F + w_A R_A] \\
&= \mathbb{V}[w_A R_A] \\
&= w_A^2 \mathbb{V}[R_A] \\
&= w_A^2 \sigma_A^2 \\
\sigma_P &= |w_A| \sigma_A
\end{aligned}
$$

空売りを想定する場合 $w_A < 0$ となるため，標準偏差を求める際に絶対値をとっている。空売りはない状況（つまり，$w_A > 0$）を想定すると，

$$
\begin{aligned}
\sigma_P &= w_A \sigma_A \\
w_A &= \frac{\sigma_P}{\sigma_A}
\end{aligned}
$$

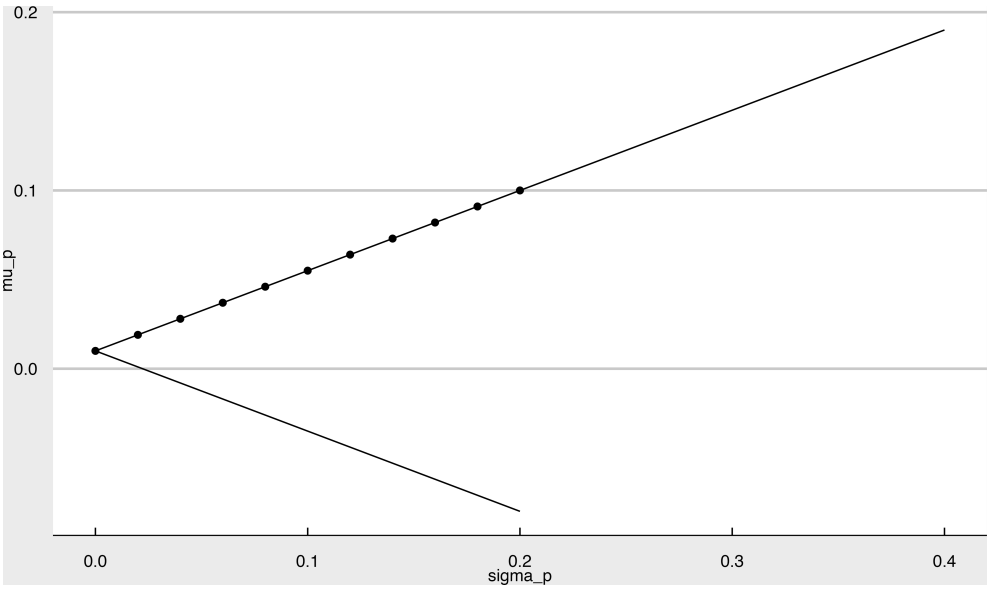
のように，リスク資産である銘柄 A への投資割合 w_A が，ポートフォリオ P とリスク資産 A のリスクの割合で決定されることがわかる。これを，ポートフォリオの期待リターン μ_P に代入すると，

$$
\begin{aligned}
\mu_P &= R_F + w_A(\mu_A - R_F) \\
&= R_F + \frac{\sigma_P}{\sigma_A}(\mu_A - R_F) \\
&= R_F + \frac{\mu_A - R_F}{\sigma_A}\sigma_P
\end{aligned}
$$

となり，期待リターン μ_P は，切片が R_F，傾きが $(\mu_A - R_F)/\sigma_A$ とする σ_P の線形関数となる。

```
R_F = 0.01
mu_A <- 0.1
sigma_A <- 0.2
mu_B <- 0.2
sigma_B <- 0.3
# 保有費率
wa <- seq(-1,2,by = 0.01)
wb <- 1 - wa
```

```
df <- tibble(
  mu_p = R_F + wa*(mu_A - R_F),
  sigma_p = abs(wa)*sigma_A,
  label= c(
  rep("",100),
  "0,1", rep("", 9),
  "0.1,0.9",rep("", 9),
  "0.2,0.8",rep("", 9),
  "0.3,0.7",rep("", 9),
  "0.4,0.6",rep("", 9),
  "0.5,0.5",rep("", 9),
  "0.6,0.4",rep("", 9),
  "0.7,0.3",rep("", 9),
  "0.8,0.2",rep("", 9),
  "0.9,0.1",rep("", 9),
  "1,0" , rep("",100)
  )
)
ggplot(df) + aes(y = sigma_p, x = mu_p) + geom_line() +
 ↪  geom_point(data = filter(df, label != ""), aes(label = label),
 ↪  size = 2) + coord_flip() + mystyle
```



2.2.5　3 資産のポートフォリオ

リスク資産 A と B，安全資産 F の 3 資産に投資するポートフォリオを考える。ここで，$w_A + w_B > 0$ を仮定し，少なくとも少しはリスク資産を保有するケースを考える。当然だけれど，$w_A = w_B = 0$ のケースでは，安全資産のみを保有するケースとなり，リスクも無く，リターンも確定している。

3 資産 A,B,F への投資割合をそれぞれ w_A，w_B,w_F とすると，3 資産からなるポートフォリオの期待リターンは次のように計算できる。

$$\begin{aligned}\mathbb{E}[R_P] &= w_F R_F + w_A \mathbb{E}[\tilde{R}_A] + w_B \mathbb{E}[\tilde{R}_B] \\ &= w_F R_F + (w_A + w_B)\left(\frac{w_A}{w_A + w_B}\mathbb{E}[\tilde{R}_A] + \frac{w_B}{w_A + w_B}\mathbb{E}[\tilde{R}_B]\right)\end{aligned}$$

安全資産への投資割合 w_F とリスク資産への投資割合 $w_C = w_A + w_B$ とまとめて，式を変形させる。安全資産への投資以外の資金で構築したリスク資産 A と B からなるポートフォリオを P_C を考えると，P_C の期待リターン R_C は次のように計算できる。

$$R_C = \frac{w_A}{w_A + w_B}\mathbb{E}[\tilde{R}_A] + \frac{w_B}{w_A + w_B}\mathbb{E}[\tilde{R}_B]$$

安全資産 F とポートフォリオ C を保有した場合の期待リターンは次式となる。

$$\begin{aligned}\mu_P &= \mathbb{E}[w_F R_F + w_A R_A + w_B R_B] \\ &= w_F R_F + w_A \mathbb{E}[R_A] + w_B \mathbb{E}[R_B] \\ &= w_F R_F + (w_A + w_B) \underbrace{\left(\frac{w_A}{w_A + w_B}\mathbb{E}[R_A] + \frac{w_B}{w_A + w_B}\mathbb{E}[R_B]\right)}_{=w_C \text{とおく}} \\ &= w_F R_F + w_C \mu_C\end{aligned}$$

安全資産と 2 つのリスク資産からなるポートフォリオの期待リターンは，安全資産の期待リターンとリスク資産の期待リターンの和となる。

安全資産と 2 つのリスク資産に投資可能な場合，(μ_P, σ_P) の取りうる値を図示できる。テキストの数値例を用いて R で図示してみる。

- リスク資産 A の期待リターン 0.1，標準偏差 0.2
- リスク資産 B の期待リターン 0.2，標準偏差 0.3
- 安全資産の期待リターンを 0.01
- リスク資産 A と B の間の相関係数は 0.2
- 安全資産，銘柄 A，銘柄 B への投資割合を 0.2，0.3，0.5 とするポートフォリオを考える。このポートフォリオの期待リターン μ_P と標準偏差 σ_P は次のようになる。

2.3　最適ポートフォリオ問題

「どのようなポートフォリオが投資家にとって望ましいか」

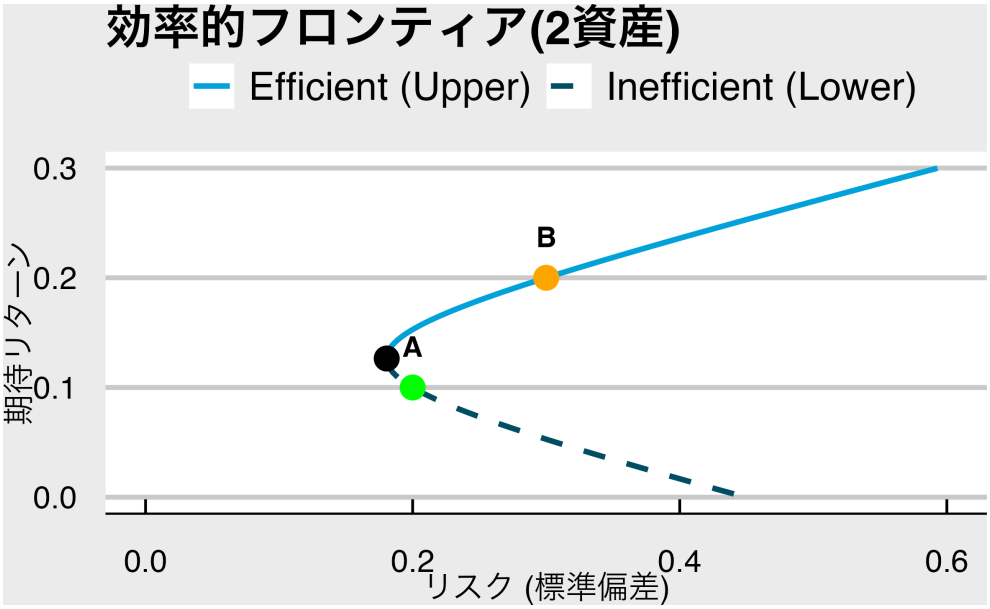
一般に、投資家はリスクが小さい一方でリターンが大きいポートフォリオを好む。ここでは，リターンをポートフォリオの期待リターン μ_P，リスクをポートフォリオの標準偏差 σ_P で表し，この 2 つの変数から更正される平面 (μ_P, σ_P) 上で最適ポートフォリオ問題を分析する。

2.3.1 効率的フロンティア

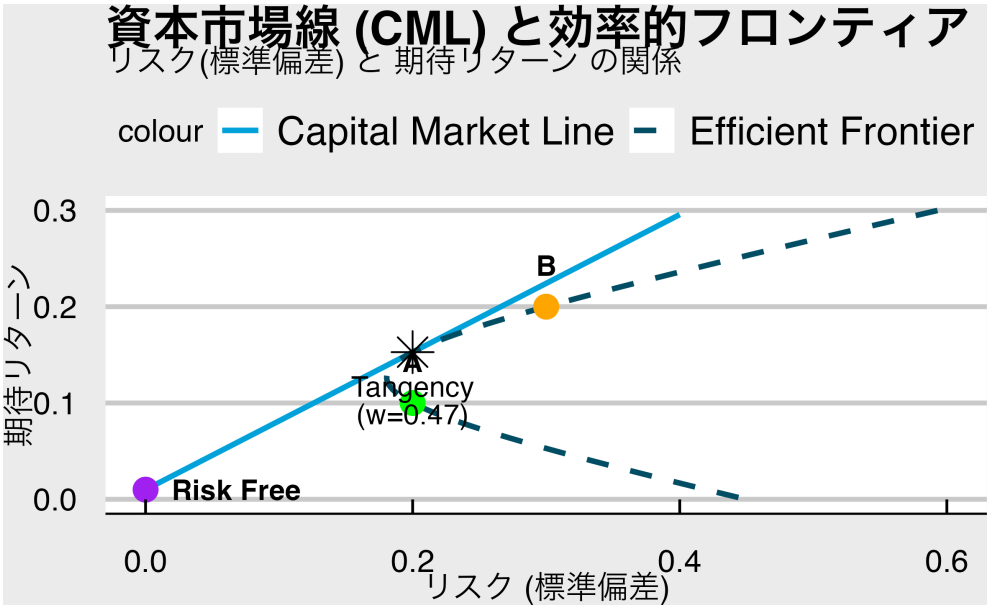
リスク資産 A と資産 B にのみ投資可能であり，それぞれに異なる割合で投資したポートフォリオ D と Eを比較する (以下の図)

- 標準偏差はともに 0.25 $\sigma_D = \sigma_E = 0.25$
- D の方が E よりも期待リターンが大きい，$\mu_D > \mu_E$

つまり，同じリスク（標準偏差）ならリターン（期待リターン）が高いポートフォリオに投資したほうがよい。よってリスク・リターンのトレードオフの意味で D の方が E よりも望ましい。以下のグラフでいうと、同じリスク (緑のライン上) なら、リターンの高いポートフォリオが望ましい。そのため赤い実線が効率的フロンティアとなり、青い点線は選択されないポートフォリオになる。



リスク資産 A と B に加え、安全資産 F にも投資可能な場合、効率的フロンティアは**直線**になる。この場合、効率的フロンティアは**資本市場線** (Capital Market Line; CML) とも呼ばれる。傾きは、この金融市場におけるリスクとリターンのトレードオフを表す。



2.3.2　投資家のリスク回避度と最適ポートフォリオ

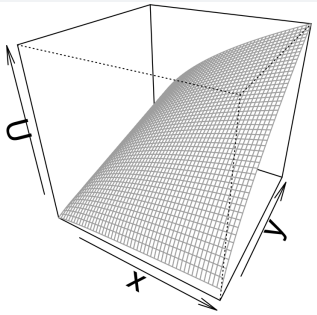
効率的フロンティアのうち、どの点が投資家の最適ポートフォリオになるのか，について考える。そのためには投資家のリスク・リターンのトレードオフに関する**選好 (preference)** の特徴，つまり**リスクの回避度**の情報が必要となる。(μ_P, ρ_p) 平面上でそれを描く方法の一つが**無差別曲線** (indifference curve) である。**無差別曲線**とは，投資家の効用 (utility) が**一定**となるリスクとリターンの組み合わせを描いた曲線をいう。つまり同じ効用水準を達成できるリスクとリターンの組み合わせを表現した曲線である。

2.3.2.1　効用関数の例

以下では，財 x と y を消費したときの効用 U を図示している。この消費者の効用関数は $U(x, y) = x^{\frac{2}{5}} \times y^{\frac{3}{5}}$ としている。

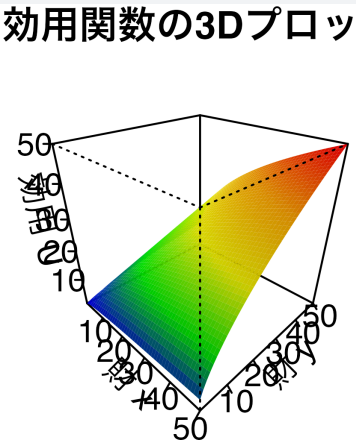
```
library(Rsolnp)
x <- 1:50
y <- 1:50
u <- function(x, y) {x^(2/5) * y^(3/5)} #効用関数を定義
U <- outer(x, y, u) #outer() は x_1,x_2 に対応した f(x_1,x_2) の値を行列で
 ↪ 返す
persp(x, y, U,
      theta = 30, # 横回転の角度
      phi = 30, # 縦回転の角度
      ticktype = "simple", # 線の種類
      lwd = 0.5, # 線の太さ
```

```
      col = F,
      border = 8)
```




```
# 必要なライブラリ
library(plotly)

# データ作成
x <- 1:50
y <- 1:50
u <- function(x, y) {x^(2/5) * y^(3/5)}
U <- outer(x, y, u)

# 色の準備（高さに応じて色を変える処理）
nrz <- nrow(U)
ncz <- ncol(U)
# グラデーションパレットを作成（青→緑→黄）
jet.colors <- colorRampPalette(c("blue", "green", "yellow", "orange",
 ↪  "red"))
nbcol <- 100
color <- jet.colors(nbcol)
# 高さ（z）を色のインデックスに変換
zfacet <- U[-1, -1] + U[-1, -ncz] + U[-nrz, -1] + U[-nrz, -ncz]
facetcol <- cut(zfacet, nbcol)

# 描画
persp(x, y, U,
      theta = 45, phi = 30,
      col = color[facetcol], # 計算した色を指定
      shade = 0.2,           # 陰影
      border = NA,           # 枠線なしで滑らかに
      ticktype = "detailed",
```

```
      xlab = "財 x", ylab = "財 y", zlab = "効用 U",
      main = "効用関数の 3D プロット")
```



この立体図を等高線を使って表現したものが以下の図である。青いラインは予算制約であり、予算の範囲内で購入可能な財の組み合わせを意味している。つまり、この予算制約と無差別曲線が接する点が、予算内で達成可能な最も高い効用水準を表している。

無リスク利子率が 10% で，リスクプレミアムが 5 %，β が 1.2 の場合の期待リターンは，$R_F + \beta \times (R_M - R_F) = 0.1 + 1.2 \times 0.05 = 0.16$ となる。このときの無差別曲線は，$U = 0.16$ となるような点を結んだ曲線となる。

```
contour(x, y, U, method = "edge", labcex = 1,lwd = 2)
abline(a = 100/6, b = -4/6, lwd = 2, col = "blue") #予算制約線
points(x = 10, y = 10, lwd = 3, col = "darkblue", pch = 16) #最適消費点
```

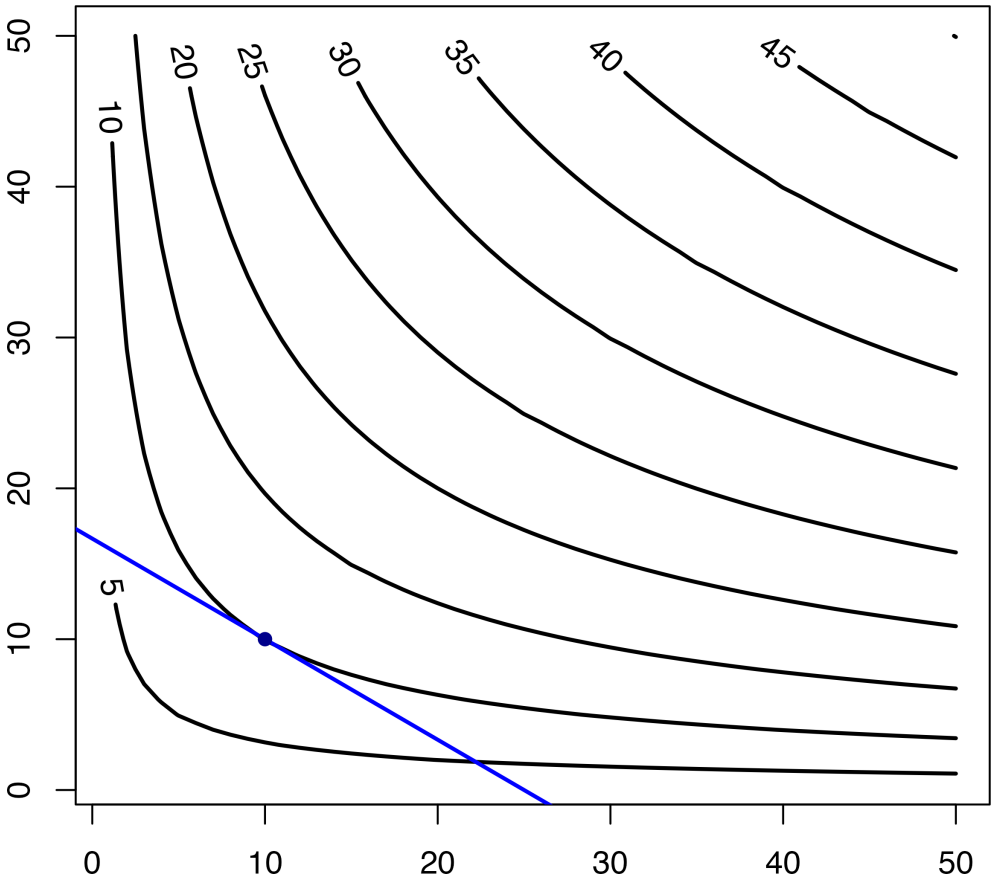


Figure2.1: 無差別曲線と消費可能集合

リスクとリターンの無差別曲線

複数ポートフォリオを比べるとき，この図の左上のものほど高リターン低リスクに対応するので、より高い効用水準が実現する。また無差別曲線の**局所的な傾き**は、その投資家が追加的なリスクを引き受けるうえで要求するリスクプレミアムを表す。リスク回避的な投資家ほど，リスクを 1 単位負担する際に，より大きなリスクプレミアムを要求するので、傾きは大きくなる。つまり，1 単位リスクを負担する代わりに欲しいリターンの額が大きくなるほど，傾きが大きくなる。

無差別曲線と効率的フロンティアが接する点が、この投資家にとっての最適ポートフォリオとなる。投資可能なポートフォリオの範囲で、最も左上の無差別曲線を実現するのが接点となる。どの点が最適ポートフォリオとして選ばれるかは個々の投資家の無差別曲線の形状 (リスク回避度) に依存する。最適ポートフォリオにおいて、無差別曲線と効率的フロンティアの局所的な傾きは一致 (接線だから当然) するため、その投資家が要求するリスクプレミアムがちょうど実現されている。

上図の場合，この投資家の最適ポートフォリオは $(w_F, w_{tan}) \approx (0.29, 0.71)$ の比率で構成される。

接点ポートフォリオは銘柄 A に 47 %、銘柄 B に 53 %投資するポートフォリオだった

ので、最適保有比率は、$(w_F, w_A, w_B) \approx (0.29, 0.33, 0.38)$ と書き換えられる。

2.3.3　トービンの分離定理

安全資産が投資可能な場合の最適ポートフォリオ問題を考える。

1. 接点ポートフォリオを求め、リスク資産同士の相対的な保有比率を求める。
2. 投資家ごとのリスク回避度に応じて安全資産と接点ポートフォリオの最適保有比率の決定

1 は各投資家で共通している。いったん接点ポートフォリオを求めてしまえば、他の投資家はその情報を用いて 2 を考えればよい。

最適ポートフォリオ問題を 2 段階に分離できるという命題は、**トービンの分離定理** (又は二基金文理定理) と呼ばれている。

2.4　CAPM

金融市場全体の均衡を考えるために、**資本資産価格モデル** (Capital Asset Pricing Model; CAPM) を学習します。

2.4.1　仮定の確認

- **選好**：ポートフォリオを**期待値と標準偏差の基準で評価**
- **取引コスト**：取引コストは存在せず、空売りも自由
- **流動性**：どれだけ売買しても証券の価格は不変
- **情報集合**：みんな同じ情報を共有

上記の仮定を満たす市場のことを、**完全資本市場** (完全市場: perfect market) と呼びます。空想上の市場ですが、理論構築の大事な出発点です。

2.4.2　CAPM の第一命題

先の仮定の下で、安全資産が投資可能なとき、全ての投資家の最適ポートフォリオ問題に対してトービンの分離定理を応用することができます。全ての投資家は「安全資産」と「接点ポートフォリオ」のみに投資し、危険資産に限定すれば全員が同じ構成比率のポートフォリオ（接点ポートフォリオ）を保有します。金融市場全体の均衡を議論するうえで、市場にその資産が供給されている以上、誰かがその最適ポートフォリオの一部として保有しているはずです。すなわち、需要と供給が一致している点がポイントです。

以上の議論をよりフォーマルに述べるために、市場ポートフォリオを導入します。

! 市場ポートフォリオ

市場ポートフォリオ (market portfolio) とは、市場に存在する全ての危険資産を時価総額比率で保有したポートフォリオをいう。厳密には、リスク資産には株式や債券に代表される金融資産の他、不動産や貴金属などの実物資産も含まれるが、実用上は TOPIX や S&P500 といった株価指数と同一視されることが多い。

! CAPM の第一命題

市場ポートフォリオは接点ポートフォリオと一致し、効率的フロンティア（資本市場線）上に位置する。

投資家は市場ポートフォリオに投資するとき、σ_M のリスクを背負う見返りとして、R_F に加えて $\mu_M - R_F$ だけ追加的な報酬を期待します。この追加的な報酬を**市場リスクプレミアム** (market risk premium) といいます。したがってこの命題の下では、資本市場線を市場リスクプレミアム ($\mu_M - R_F$) を利用して、以下のように表せます。

$$\mu_P = R_F + \frac{\mu_M - R_F}{\sigma_M}\sigma_P$$

前の章で用いたパラメータをそのまま考えます。接点ポートフォリオの保有比率は概ね 47% を銘柄 A に、53% を銘柄 B に投資する結果となりました。CAPM の第一命題によると、均衡した市場における銘柄 A と B の時価総額比率は約 0.47 対 0.53 になっていなければなりません。この命題に従えば、投資家は各銘柄の期待リターンや分散から接点ポートフォリオを計算する必要はなく、単に時価総額加重で市場ポートフォリオを保有すればよいことになります。

- **パッシブ運用：**幅広い銘柄に分散投資し、市場平均と同じようなパフォーマンスを目指す運用手法。
- **アクティブ運用：**市場平均を上回るパフォーマンスを目指し、投資銘柄を絞ったり、投資比率を工夫したりする運用方法。

任意のポートフォリオの収益性を測る指標として、**シャープ・レシオ**が提唱されています。シャープ・レシオは追加的なリスク・テイクによってどれだけリスクプレミアムを改善できるのかを表す指標です。CAPM の第一命題によると、市場ポートフォリオはシャープ・レシオを最大化するという意味で最も効率的なポートフォリオであり、資本市場線の傾き $\frac{\mu_M - R_F}{\sigma_M}$ は市場ポートフォリオのシャープ・レシオと一致します。

$$\text{Sharpe Ratio} = \frac{\mu_P - R_F}{\sigma_P}$$

2.4.3　CAPM の第二命題

第二命題は個々の資産のリスクとリターンのトレードオフを数式で表現したものです。ある証券に投資するときのリスクと、その証券に投資するときの期待リターンとの関係を知ることができるようになります。

各投資家が証券 i を追加的に保有する際、重要となるのは市場ポートフォリオとの相関です。分散が大きい資産であっても、市場ポートフォリオと負に相関していれば、その資産を追加的に保有することでポートフォリオ全体のリスクは低減されます。CAPM の第二命題は、この相関を以下の**マーケット・ベータ**として定量化します。（※ R_i は証券 i のリターン、R_M は市場ポートフォリオのリターン）

この β_i は市場ポートフォリオのリスクを 1 としてベンチマーク化し、その証券のリスクがベンチマークの 1 を上回るか下回るかを測るものです。β_i が大きいほど証券 i は投資家にとってリスク（システマティック・リスク）が大きいことを意味します。CAPM の世界では、証券 i のリスクはその証券のリターンの標準偏差ではなく、この β_i によって測られます。

$$\beta_i = \frac{\mathbb{C}\mathrm{ov}[R_i, R_M]}{\mathbb{V}\mathrm{ar}[R_M]}$$

金融市場全体が均衡しているには、リスクの高い証券はその分だけ期待リターンも高くなければなりません。もし β_i が低いにもかかわらず期待リターンが高い証券があるなら、投資家は市場ポートフォリオから離れてその証券をさらに買い増しするインセンティブを持ちます。その結果、市場価格が上がり、期待リターンが下がるため、最終的に β_i に応じた期待リターンが均衡で実現されます。

これまでは市場リスクプレミアムを $\mu_M - R_F$ と表記していましたが、以後ではより一般的な $\mathbb{E}[R_M] - R_F$ と表記します。

> **!** CAPM の第二命題
>
> 各証券のリスクプレミアムは、その証券のマーケット・ベータに比例する。この式は、証券 i のリスクプレミアム $\mathbb{E}[R_i] - R_F$ を、β_i と市場リスクプレミアム $\mathbb{E}[R_M] - R_F$ に分解している。

$$\mathbb{E}[R_i] - R_F = \beta_i(\mathbb{E}[R_M] - R_F)$$
$$\text{ただし、} \beta_i = \frac{\mathbb{C}\mathrm{ov}[R_i, R_M]}{\mathbb{V}\mathrm{ar}[R_M]}$$

通常、市場リスクプレミアムは正の値をとるので、CAPM の第二命題によると、個々の証券のリスクプレミアムは β_i に関して線形に増加します。β_i はあくまで市場ポートフォリオとの相関でリスクを定量化しているのがポイントです。いくら個々の証券の分散（リ

スク）が大きくても、それが市場ポートフォリオと相関しない固有リスクであれば、リスクプレミアムには反映されません。期待値をとる前の R_i を分解して確認します。

$$R_i = R_F + \beta_i(R_M - R_F) + \varepsilon_i$$

ここで ε_i は期待値ゼロで R_M と相関しない誤差項です。

$$\mathbb{E}[\varepsilon_i] = 0, \qquad \mathbb{C}\mathrm{ov}[\varepsilon_i, R_M] = 0$$

分散を計算すると以下のようになります。

$$\begin{aligned}
\mathbb{V}\mathrm{ar}[R_i] &= \mathbb{V}\mathrm{ar}[\beta_i R_M + \varepsilon_i] \\
&= \beta_i^2 \mathbb{V}\mathrm{ar}[R_M] + \mathbb{V}\mathrm{ar}[\varepsilon_i] + \underbrace{2\mathbb{C}\mathrm{ov}[\beta_i R_M, \varepsilon_i]}_{=0} \\
&= \underbrace{\beta_i^2 \mathbb{V}\mathrm{ar}[R_M]}_{\text{市場ポートフォリオとの相関による寄与分}} + \underbrace{\mathbb{V}\mathrm{ar}[\varepsilon_i]}_{\text{誤差項による寄与分}}
\end{aligned}$$

R_i の分散は、市場ポートフォリオとの相関による寄与分（システマティック・リスク）と誤差項による寄与分（固有リスク）に分解できます。誤差項の分散が大きければその分だけ R_i の分散も大きくなりますが、証券 i のリスクプレミアムは $\beta_i(\mathbb{E}[R_M] - R_F)$ のままで変化はありません。つまり、分散投資によって消去可能な固有リスクに対しては、市場は報酬（プレミアム）を与えないのです。

2.4.4 証券市場線

安全資産と複数の危険資産が投資可能な場合、投資家の最適ポートフォリオは資本市場線上の 1 点となります（図 2.14 左図）。一方、CAPM の第二命題が示唆するように、各証券のリスク（ベータ）とリターンとの関係は、右上がりの直線となります（図 2.14 右図）。縦軸に各証券の期待リターン、横軸に各証券のリスクを表すマーケット・ベータをとると、CAPM が完全に成立する世界では全ての資産が一直線上に並びます。この直線を**証券市場線** (Securities Market Line; SML) と呼びます。

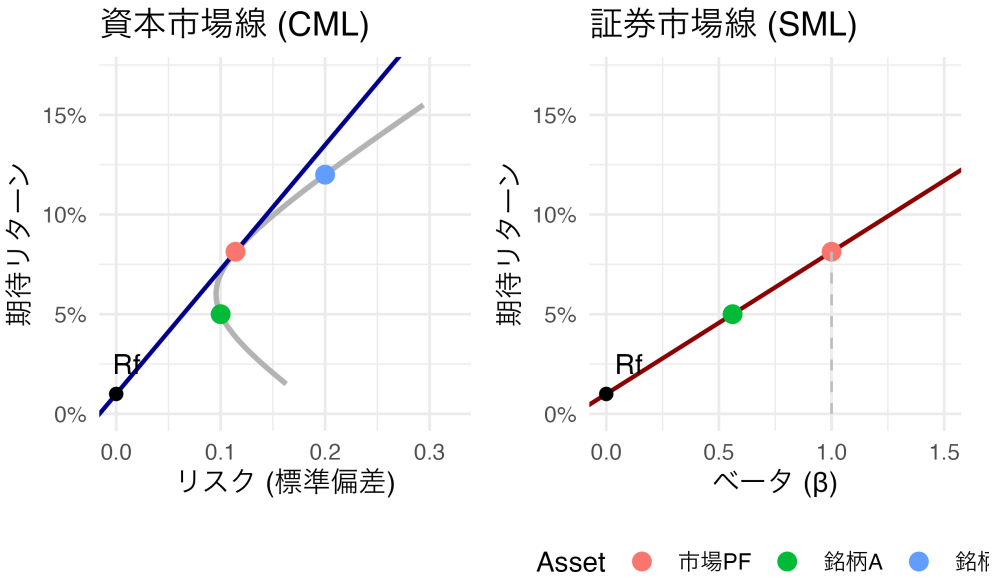
定義通り β を計算すると銘柄 A は約 0.63、銘柄 B は約 1.33 となります。両者の期待リターンおよび β を図示すると証券市場線に乗っており、この仮想的な市場では CAPM が成立していることがわかります。

```
library(tidyverse)
library(patchwork)


# ------------------------------------------------------------
# 1. パラメータ設定（本文の記述と整合するように調整した数値）
# ------------------------------------------------------------
# ※ ユーザーの手元に前の章のデータがある場合はそれを読み込んでください。
# ここでは本文中の「A:47%, B:53%」「Beta A:0.63, Beta B:1.33」
```

```
# と概ね一致するパラメータを設定します。

mu_A <- 0.05       # 銘柄 A の期待リターン
sig_A <- 0.10      # 銘柄 A の標準偏差
mu_B <- 0.12       # 銘柄 B の期待リターン
sig_B <- 0.20      # 銘柄 B の標準偏差
rho <- 0.2         # 相関係数
Rf <- 0.01         # 安全利子率

# 共分散
cov_AB <- rho * sig_A * sig_B

# ------------------------------------------------------
# 2. 接点ポートフォリオ（市場ポートフォリオ）の計算
# ------------------------------------------------------
# 過剰リターンベクトル
R_excess <- c(mu_A - Rf, mu_B - Rf)
# 分散共分散行列
Sigma <- matrix(c(sig_A^2, cov_AB, cov_AB, sig_B^2), nrow = 2)

# 接点ポートフォリオのウェイト計算（解析解）
Sigma_inv <- solve(Sigma)
w_tangency_unscaled <- Sigma_inv %*% R_excess
w_M <- w_tangency_unscaled / sum(w_tangency_unscaled) # ウェイトの和を 1
 ↪  にする

# 市場ポートフォリオの期待リターンとリスク
mu_M <- as.numeric(t(w_M) %*% c(mu_A, mu_B))
sig_M <- as.numeric(sqrt(t(w_M) %*% Sigma %*% w_M))

# Beta の計算: Cov(Ri, RM) / Var(RM)
# Cov(Ra, Rm) = w_A*Var(A) + w_B*Cov(A,B)
cov_AM <- w_M[1] * sig_A^2 + w_M[2] * cov_AB
cov_BM <- w_M[1] * cov_AB + w_M[2] * sig_B^2

beta_A <- cov_AM / sig_M^2
beta_B <- cov_BM / sig_M^2
```

```
# データフレーム化（プロット用）
assets <- tibble(
  Asset = c("銘柄 A", "銘柄 B", "市場 PF"),
  Mu = c(mu_A, mu_B, mu_M),
  Sigma = c(sig_A, sig_B, sig_M),
  Beta = c(beta_A, beta_B, 1) # 市場 PF のベータは定義上 1
)

# ------------------------------------------------------
# 3. 効率的フロンティアのデータ生成（左図用）
# ------------------------------------------------------
w_seq <- seq(-0.5, 1.5, length.out = 300)
frontier_data <- tibble(
  w_A = w_seq,
  w_B = 1 - w_seq
) %>%
  mutate(
    Mu = w_A * mu_A + w_B * mu_B,
    Sigma = sqrt(w_A^2 * sig_A^2 + w_B^2 * sig_B^2 + 2 * w_A * w_B *
     ↪  cov_AB)
  )

# ------------------------------------------------------
# 4. 作図 (ggplot2)
# ------------------------------------------------------

# --- 左図: 資本市場線 (CML) ---
p_cml <- ggplot() +
  # 効率的フロンティア（双曲線）
  geom_path(data = frontier_data, aes(x = Sigma, y = Mu),
            color = "gray70", size = 1) +
  # 資本市場線 (CML): 切片 Rf, 接点を通る直線
  geom_abline(intercept = Rf, slope = (mu_M - Rf) / sig_M,
              color = "darkblue", linetype = "solid", size = 0.8) +
  # 各資産のポイント
  geom_point(data = assets, aes(x = Sigma, y = Mu, color = Asset),
   ↪  size = 3) +
  # Rf の点
```

```
  geom_point(aes(x = 0, y = Rf), color = "black", size = 2) +
  annotate("text", x = 0.01, y = Rf, label = "Rf", vjust = -1) +
  # ラベル類
  labs(title = "資本市場線（CML)",
       x = "リスク（標準偏差)", y = "期待リターン") +
  theme_minimal() +
  scale_x_continuous(limits = c(0, max(frontier_data$Sigma)*1.1)) +
  scale_y_continuous(limits = c(0, max(frontier_data$Mu)*1.1), labels
   ↪  = scales::percent) +
  theme(legend.position = "none") # 凡例は右図と共通または省略

# --- 右図: 証券市場線（SML） ---
p_sml <- ggplot() +
  # 証券市場線（SML): 切片 Rf, 傾き (Rm-Rf)
  geom_abline(intercept = Rf, slope = (mu_M - Rf),
              color = "darkred", linetype = "solid", size = 0.8) +
  # 各資産のポイント
  geom_point(data = assets, aes(x = Beta, y = Mu, color = Asset), size
   ↪  = 3) +
  # 補助線（市場 PF のベータ=1 を示す線）
  geom_segment(aes(x = 1, xend = 1, y = 0, yend = mu_M),
               linetype = "dashed", color = "gray") +
  # Rf の点
  geom_point(aes(x = 0, y = Rf), color = "black", size = 2) +
  annotate("text", x = 0.1, y = Rf, label = "Rf", vjust = -1) +
  # ラベル類
  labs(title = "証券市場線（SML)",
       x = "ベータ（β)", y = "期待リターン") +
  theme_minimal() +
  scale_x_continuous(limits = c(0, 1.5)) +
  scale_y_continuous(limits = c(0, max(frontier_data$Mu)*1.1), labels
   ↪  = scales::percent) +
  theme(legend.position = "bottom")

# --- 図の結合 ---
p_combined <- p_cml + p_sml
p_combined
```



2.5　N 資産が投資可能な場合への拡張

今までは，リスク資産が銘柄 A と B の二つしかない場合を分析してきたが，現実は多くのリスク資産が存在し、海外株式や債券、REIT(不動産投資信託) といったその他の投資可能な金融資産を含めればその数は飛躍的に増加する。本節での平均分散アプローチや CAPM は危険資産の数が任意の N 個であっても成立する。ただしその場合は**行列での表記**が必須となる (第 7 章やサポートサイト 4.5 節参照)。

一般に，平均分散の意味で効率的なポートフォリオ (**平均分散ポートフォリオ**) を計算するには、目標期待リターンを所与として、それを実現するポートフォリオの中でリスクを最小化するものを求める。得られた期待リターンとリスクのペアを一点として、目標期待リターンを動かすとリスク・リターン平面上に双曲線が描ける。この双曲線を**平均分散フロンティア**と呼び、効率的フロンティアはその上半分の領域である (確認済み)。

一般に投資可能な資産の数が増えると、平均分散フロンティアは左上に移動し、投資家はより望ましいポートフォリオが実現できるようになる。リスク資産 A と B に加えて C が投資可能な状況を考えると投資可能な資産が増えたからと言って必ずしもその資産に投資する必要はない。$w_C = 0$ とすれば、投資家は危険資産 A と B のみに投資可能だった場合と同じ投資機会集合を実現できる。新しい危険資産が既存資産の組み合わせによって完全に再現できるような極端な例を除けば分散投資のメリットが生じるため、投資家はより望ましいリスク・リターンのトレードオフを実現できる。

統計学における分散の定義は，N 個の確率変数 $R_1, R_2, \cdots, R_N$ の共分散行列 Σ の対角成分の和である。

$$\sigma^2 = \sum_{i=1}^{N}\sum_{j=1}^{N}\sigma_{ij} = \sum_{i=1}^{N}\sigma_{ii}$$

ここで，σ_{ij} は R_i と R_j の共分散であり，σ_{ii} は R_i の分散である。共分散行列 Σ は対称

行列であり，対角成分は分散を表す。また，R_i と R_j の共分散は $\sigma_{ij} = \sigma_{ji}$ である。共分散行列の対角成分以外の成分は共分散を表す。

$$\Sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1N} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{N1} & \sigma_{N2} & \cdots & \sigma_{NN} \end{bmatrix}$$

上のは，N 個の確率変数の共分散行列の定義である。Copilot で作成しました。

Listing 2.5 効率的フロンティア

```
# --- 1. パラメータ設定 ---
mu_a    <- 0.1  # 株式 A の期待収益率
sigma_a <- 0.2  # 株式 A の標準偏差
mu_b    <- 0.2  # 株式 B の期待収益率
sigma_b <- 0.3  # 株式 B の標準偏差
rho     <- 0.2  # 相関係数

# --- 2. 最小分散ポートフォリオ (MVP) の算出 ---
# 効率的フロンティアと非効率部を分ける頂点を正確に計算します
# 分散共分散
cov_ab <- rho * sigma_a * sigma_b
var_a  <- sigma_a^2
var_b  <- sigma_b^2

# MVP における資産 A のウェイト w_mvp
# 公式: w = (Var(B) - Cov(A,B)) / (Var(A) + Var(B) - 2Cov(A,B))
w_mvp <- (var_b - cov_ab) / (var_a + var_b - 2 * cov_ab)

# MVP のリターンとリスク
mu_mvp    <- w_mvp * mu_a + (1 - w_mvp) * mu_b
sigma_mvp <- sqrt(w_mvp^2 * var_a + (1 - w_mvp)^2 * var_b + 2 * w_mvp
 ↪  * (1 - w_mvp) * cov_ab)

# --- 3. プロット用データの作成 ---
# ウェイトを広めにとる
w <- seq(-1, 2, by = 0.001)

# リターンとリスクを計算
mu_p    <- w * mu_a + (1 - w) * mu_b
sigma_p <- sqrt((w * sigma_a)^2 + ((1 - w) * sigma_b)^2 + 2 * w * (1 -
 ↪  w) * rho * sigma_a * sigma_b)

# データフレーム化し、MVP よりリターンが高いかどうかでフラグ立て
df_plot <- data.frame(w, mu_p, sigma_p) %>%
  mutate(frontier_type = ifelse(mu_p >= mu_mvp, "Efficient (Upper)",
   ↪  "Inefficient (Lower)"))

# --- 4. プロット作成 ---
ggplot() +
  # A. フロンティアの描画（実線と破線で色分け）
  geom_path(data = df_plot, aes(x = sigma_p, y = mu_p, color =
   ↪  frontier_type, linetype = frontier_type), size = 1) +
```

Listing 2.6 効率的フロンティアと資本市場線

```
# --- 1. パラメータ設定（講義資料より）---
# を参照
mu_a    <- 0.1  # 株式 A の期待収益率
sigma_a <- 0.2  # 株式 A の標準偏差
mu_b    <- 0.2  # 株式 B の期待収益率
sigma_b <- 0.3  # 株式 B の標準偏差
rho     <- 0.2  # 相関係数
R_F     <- 0.01 # 無リスク利子率

# --- 2. 効率的フロンティアのデータ作成 ---
# ウェイト（w）を変化させる（-100% から 200% まで）
w <- seq(-1, 2, by = 0.001)

# ポートフォリオの期待リターン（mu_p）とリスク（sigma_p）を計算
# 公式: mu_p = w*mu_A + (1-w)*mu_B
mu_p <- w * mu_a + (1 - w) * mu_b

# 公式: sigma_p = sqrt(w^2*sigma_A^2 + (1-w)^2*sigma_B^2 +
 ↪  2w(1-w)*rho*sigma_A*sigma_B)
sigma_p <- sqrt((w * sigma_a)^2 + ((1 - w) * sigma_b)^2 + 2 * w * (1 -
 ↪  w) * rho * sigma_a * sigma_b)

# データフレームにまとめる
df_frontier <- data.frame(w, mu_p, sigma_p)

# --- 3. 接点ポートフォリオ（Tangency Portfolio）の導出 ---
# シャープレシオ（リスク単位あたりの超過リターン）を計算
df_frontier <- df_frontier %>%
  mutate(sharpe_ratio = (mu_p - R_F) / sigma_p)

# シャープレシオが最大になる点（=接点ポートフォリオ）を取得
tangency_port <- df_frontier %>%
  filter(sharpe_ratio == max(sharpe_ratio))

# 最大シャープレシオ（これが資本市場線の傾きになる）
max_sharpe <- tangency_port$sharpe_ratio

# --- 4. 資本市場線（CML）のデータ作成 ---
# リスク 0 から 0.6 までの範囲で直線のデータを生成
# CML の式: mu = R_F + (Sharpe Ratio) * sigma
cml_sigma <- seq(0, 0.6, by = 0.01)
cml_mu    <- R_F + max_sharpe * cml_sigma
```

第 3 章

R 言語入門

Listing 3.1 いろいろな設定

```
pacman::p_load(tidyverse, ggthemes)
mystyle <- list(
  theme_economist_white(
    # gray_bg = FALSE,
    base_family = "HiraKakuProN-W3"
    ),
  scale_colour_economist(),
  theme(
    text = element_text(size = 12), # フォントファミリーは上で指定済みなの
    ↪ で省略可
    axis.title = element_text(size = 12)
  )
)

knitr::opts_chunk$set(
  class.source = "numberLines lineAnchors"
#   class.output = "numberLines lineAnchors chunkout"
  )
```

プログラミング言語にはいろんな種類があるけれど、今回学習する **R 言語**は、インタプリタ型とよばれるもので、コンパイルという作業の必要が無く、書いたらすぐ実行できる仕様となっています。たとえば、教科書にあるように

```
1 100 / (1 + 0.1)
```

```
[1] 90.90909
```

を実行すれば、結果がすぐ表示されます。Rstudio とか Visual Studio Code とか

Antigravity を使って、上のような R ソースコードを一気に書いてまとめて実行するための**スクリプト・ファイル**を作成します。

ソースコードを書くにあたり注意する点が 4 つあります。

1. **大文字と小文字は区別される**ので、`x` と `X` は別の変数として扱われます。
2. **半角スペース**は、**区切り文字**として扱われるので、`x <- 100` と `x<-100` は同じ意味になります。
3. **改行**も、**区切り文字**として扱われます。長いソースコードは改行して読みやすくしましょう。
4. **コメント**は、`#`から行末までの文字列がコメントとして扱われ、実行されません。プログラムの内容を説明するためにたくさん書いて残しておきましょう。

3.1　R の基本的な機能

3.1.1　スカラー変数の定義

この学習を通じて**変数** (variable) とは、**数値や文字といったデータを格納するための箱**を表し、中に何が入っているのかにより、スカラー変数、ベクトル、行列、データフレームなどに分類されます。まずは、スカラー変数の定義を学びます。

スカラー (scalar) とは、大きさだけで決まる量のことで、つまり、**1 つの数値**を指します。R 言語ではスカラー変数を定義するには、`<-`を使います。たとえば、`x <- 100` と書けば、x というスカラー変数に 100 という数値を格納できます。このとき、`<-`は代入演算子と呼ばれ、右辺の値を左辺の変数に代入するという意味です。また、`x` という変数を**左辺値** (left-hand side)、`100` という数値を**右辺値** (right-hand side) と呼びます。

```
x <- 100 # 代入演算子<- の前後に半角スペースを入れるのがお作法
```

この中身を表示されるには、`print()` 関数を使います。

```
print(x) # x の中身を表示
```

```
[1] 100
```

あるいは

```
x
```

```
[1] 100
```

でも表示されます。

R では#の後ろの文章はコメントとして扱われ、実行されません。コメントはプログラムの内容を説明するためにたくさん書いて残しておきましょう。

3.1.2　ベクトル変数の定義

ベクトル (vector) とは、大きさと向きで決まる量のことで、つまり、**複数の数値**を指します。R 言語ではベクトル変数を定義するには、`c()` を使います。たとえば、`x <- c(1, 2, 3)` と書けば、x というベクトル変数に 1, 2, 3 という数値を格納できます。このとき、

c() はベクトルを作る関数と呼ばれ、1，2，3 という数値を引数として与えています。

```
x <- c(1, 5, 9) # x に 1 と 5 と 9 を要素とするベクトルを代入
print(x)
```

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

等差数列を作る関数に seq() 関数があります。seq() は 3 つの引数をとり、

- from：始点
- to：終点
- by：差分

を指定します。たとえば、2000 年から 2020 年を表す年度の変数を year として定義するには、

```
year <- seq(from = 2000, to = 2020, by = 1)
print(year)
```

```
 [1] 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
[16] 2015 2016 2017 2018 2019 2020
```

と書けば、2000 から 2020 までの公差 1 の等差数列を作ります。seq() 変数の引数には、from と to と by の 3 つの引数を指定することができますが、from と to のみを指定することもできます。このとき、by の値は 1 となります。次のように書いても、上と同じ結果を得ることができます。

```
seq(2000,2020)
```

```
 [1] 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
[16] 2015 2016 2017 2018 2019 2020
```

ベクトルの要素数を知るには、length() 関数を使います。

```
length(year) # year の要素数を表示
```

```
[1] 21
```

ベクトル変数 year の中には 21 個の要素があることがわかります。

3.1.2.1 ベクトルの要素の取り出し

複数の要素をもつベクトルから、一部の要素を取り出すには、[] を使います。たとえば、x の 2 番目の要素を取り出すには、x[2] と書きます。このとき、[] は添字演算子と呼ばれ、2 という添字を引数として与えています。添字は 1 から始まります。

上の year から 2000 を取り出すには、year[1]、2020 を取り出すには year[20] と書きます。次のような書き方で、好きな要素を指定して取り出すことができます。

```
year[1] # 1 番目のデータを取り出す
```

```
[1] 2000
```

```
year[20] # 20 番目のデータを取り出す
```

```
[1] 2019
```

```
year[2:5] # 2 番目から 5 番目のデータを取り出す
```

```
[1] 2001 2002 2003 2004
```

```
year[c(5,10)] # 1 番目と 20 番目のデータを取り出す
```

```
[1] 2004 2009
```

```
year[6:length(year)] # 6 番目から最後のデータを取り出す
```

```
 [1] 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019
[16] 2020
```

3.1.2.2　現在価値の計算

今の時点を $t = 0$ として、T 年後に確実に得られるキャッシュ・フロー CF_T の現在価値 PV_0 は、

$$PV_0 = \frac{CF_T}{(1+r)^T}$$

と書けます。たとえば 1 年後に確実に受け取れる 100 万円の現在価値 PV_0 を計算してみます。いま、無リスク利子率 r は 10% とします。

```
100 / (1 + 0.1)^1
```

```
[1] 90.90909
```

次に、この無リスク利子率 r が変化した場合の現在価値の計算を考えます。まず、無リスク利子率のベクトルを定義します。

```
# 下の２つは同じ結果
R <- seq(from = 0.1, to = 0.2, by = 0.01)  # 省略せずに書いた場合
R <- seq(0.1, 0.2, 0.01) # 略した場合
```

次に、無リスク利子率が変化した場合の現在価値を計算します。

```
PV <- 100 / (1 + R)^1
print(PV)
```

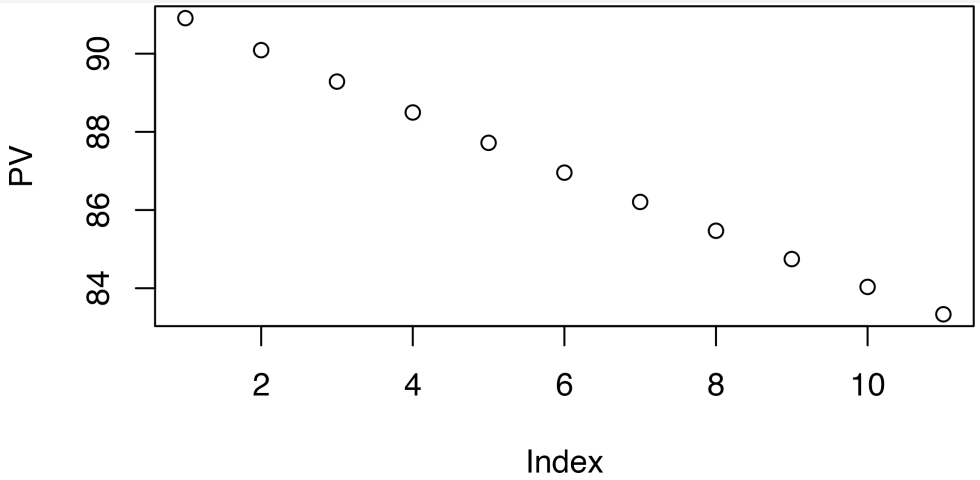
```
 [1] 90.90909 90.09009 89.28571 88.49558 87.71930 86.95652 86.20690 85.47009
 [9] 84.74576 84.03361 83.33333
```

無リスク利子率が 0.1 から 0.2 まで 0.01 ずつ変化した場合の現在価値が計算されました。この結果をグラフにしてみます。

3.1.3 基本パッケージ plot による作図

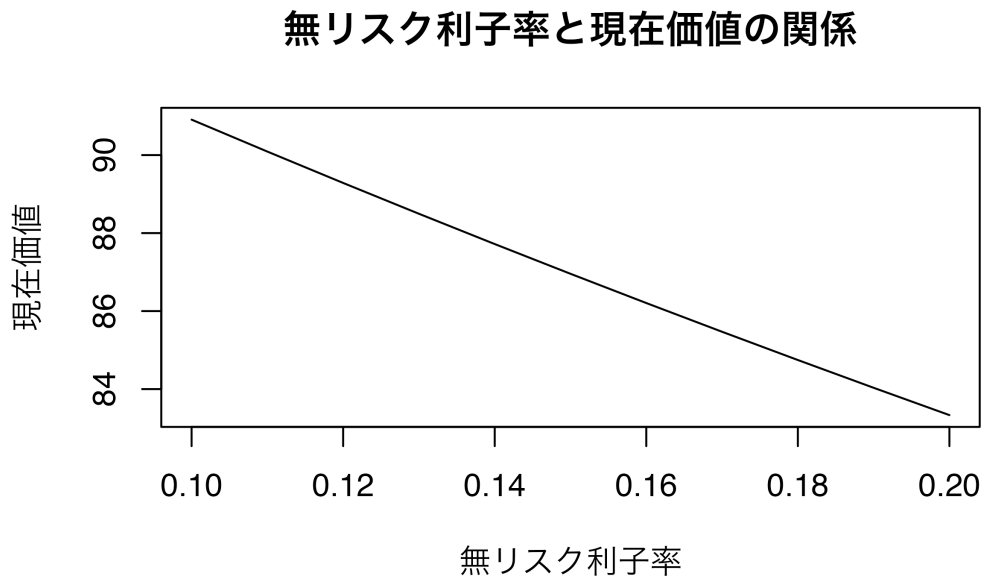
とりあえずサクッと作図してデータをチェックしたいとき、もとから R 言語に組み込まれている基本関数 plot() が便利です。先ほど作成したベクトル変数 PV をグラフにしてみます。

```
plot(PV)
```



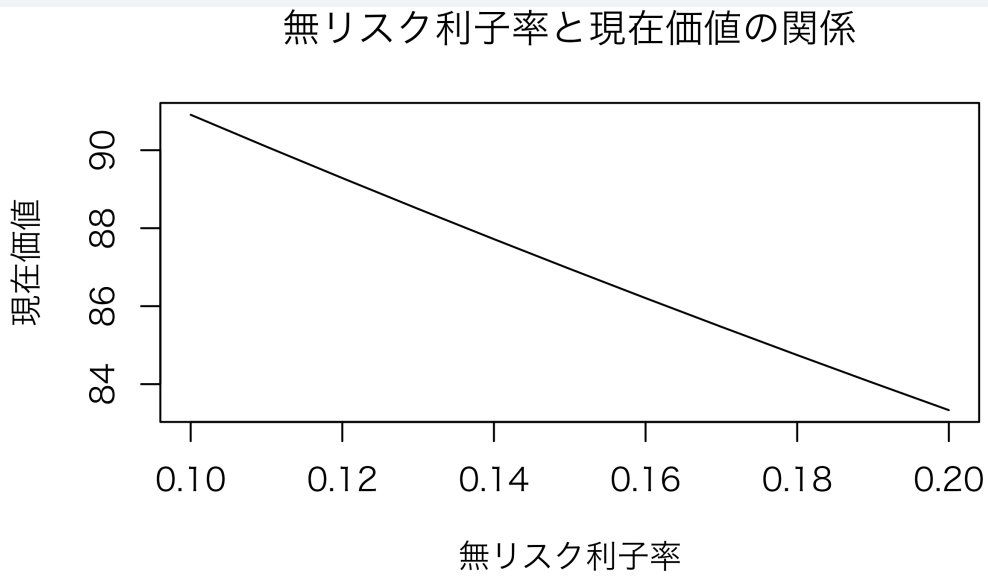
いま、PV は 11 個の要素をもつベクトル変数なので、データを左から順番に並べた散布図 (scatter diagram) が作成されています。これだと何のグラフか分かりづらいので、いろいろとオプションを指定してみます。

```
plot(
    x = R, # x 軸のデータ
    y = PV, # y 軸のデータ
    xlab = "無リスク利子率",
    ylab = "現在価値",
    main = "無リスク利子率と現在価値の関係",
    type = "l" # 線グラフ
)
```



Mac だと文字化けしてしまいました。そこで文字コードを指定します。Windows だとこの作業は不要です。

```
par(family = "HiraKakuProN-W3") # Mac の場合のみ
plot(
    x = R, # x 軸のデータ
    y = PV, # y 軸のデータ
    xlab = "無リスク利子率", # x 軸のラベル
    ylab = "現在価値", # y 軸のラベル
    main = "無リスク利子率と現在価値の関係", # グラフのタイトル
    type = "l" # 折れ線グラフ
)
```



3.2 for 文の使い方

プログラミングの基本要素である

- 繰り返し
- 分岐
- 関数

の最初の要素である「繰り返し」を行うための文法が for 文です。for 文は、ある処理を繰り返し行うための文法です。たとえば、1 から 10 までの整数を順番に表示するには、次のように書きます。

```
for (i in 1:10) { # i は 1 から 10 まで
    print(i) # i を表示
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

この文の構造は、基本的には

```
for (好きな変数 in 繰り返す範囲) {
    繰り返したい処理
}
```

となっています。

たとえば、教科書のように、

- 初期投資 100 万円
- 1 年後に 50 万円のキャッシュ・フロー
- 2 年後に 50 万円のキャッシュ・フロー
- 3 年後に 50 万円のキャッシュ・フロー

という投資プロジェクトの現在価値を計算する場合、愚直に書くと次のようになります。

```
NPV1 <- -100 +
    50 / (1 + 0.1)^1 +
```

```
    50 / (1 + 0.1)^2 +
    50 / (1 + 0.1)^3
print(NPV1)
```

```
[1] 24.3426
```

この上のコードの 2 行目から 4 行目はほぼ同じ内容なので、数字が変化しているところに注目し、for 文を使って書き換えてみます。ここでは^1 のところが 1 ずつ大きくなってます。この部分を i という変数に置き換えてみます。ついでに、後で変化させることがあるかもしれない部分をすべて変数として定義しておきます。

```
R <- 0.1 # 無リスク利子率
NPV <- -100 # 初期投資
CF <- 50 # キャッシュ・フロー

for (i in 1:3) { # i は 1 から 3 まで
    NPV <- NPV + CF / (1 + R)^i # 現在価値の計算
}
print(NPV)
```

```
[1] 24.3426
```

愚直に計算した場合の同じ結果となりました。これを 10 年間の現在価値を計算する場合だとすると、

```
R <- 0.1 # 無リスク利子率
NPV <- -100 # 初期投資
NPV1 <- NPV +
    50 / (1 + R)^1 +
    50 / (1 + R)^2 +
    50 / (1 + R)^3 +
    50 / (1 + R)^4 +
    50 / (1 + R)^5 +
    50 / (1 + R)^6 +
    50 / (1 + R)^7 +
    50 / (1 + R)^8 +
    50 / (1 + R)^9 +
    50 / (1 + R)^10
print(NPV1)
```

```
[1] 207.2284
```

と面倒くさいことこの上ないですが、for 文を使えば、

```r
R <- 0.1 # 無リスク利子率
NPV <- -100 # 初期投資
for (i in 1:10) { # i は 1 から 10 まで
    NPV <- NPV + 50 / (1 + R)^i
}
print(NPV)
```

```
[1] 207.2284
```

と短く書くことができます。使いこなせるように練習しておきましょう。次のように、print() 関数の位置を変えた場合、どうなるか考えてみてください。

```r
R <- 0.1 # 無リスク利子率
NPV <- -100 # 初期投資
for (i in 1:3) { # i は 1 から 10 まで
    print(NPV)
    NPV <- NPV + 50 / (1 + R)^i
}
```

```
[1] -100
[1] -54.54545
[1] -13.22314
```

```r
print(NPV)
```

```
[1] 24.3426
```

この場合、最初に NPV の中を表示し、次に 1 期目の現在価値を計算し、またその結果を表示し、2 期目の現在価値を計算し・・・という順番で繰り返しが行われるので、計算の途中経過が表示されることになります。

3.2.1 if 文

次に、プログラミングの基本要素である

- 繰り返し
- **分岐**
- 関数

のうち分岐を行うための文法が if 文です。if 文は、ある条件を満たす場合にのみ処理を行うための文法です。たとえば、ある変数 x が 0 より大きい場合にのみ、その変数を表示するには、次のように書きます。

```r
x <- -1
if (x > 0) { # x が 0 より大きい場合
    print(x) # x を表示
```

```
}
```

この文の構造は、基本的には

```
if (条件) {
    条件を満たす場合に実行する処理
}
```

のようになっています。この if 文を使って、NPV が 0 より大きい場合にのみ、「プロジェクトを実行！」と表示されるようにしてみます。

```
R <- 0.1 # 無リスク利子率
NPV <- -100 # 初期投資
for (i in 1:10) { # i は 1 から 10 まで
    NPV <- NPV + 50 / (1 + R)^i
}
if (NPV > 0) { # NPV が 0 より大きい場合
    print("プロジェクトを実行！") # 文字列を表示
}
```

```
[1] "プロジェクトを実行！"
```

ここでは NPV の値が 207.2284 となりプラスになっているので、「プロジェクトを実行！」と表示されます。

3.3　NPV と割引率の関係の可視化

無リスク利子率が 0.1 から 0.2 まで 0.01 ずつ変化した場合の現在価値 NPV の値を計算してみます。

```
R <- seq(0.1, 0.2, 0.01) # 無リスク利子率
N <- length(R) # 無リスク利子率の要素数 11 個
NPV <- rep(NA, N) # ベクトル変数に N 個の NA を代入

for (i in 1:N) { # i は 1 から N まで
    NPV[i] <- -100 # 初期投資
    for (j in 1:3) { # j は 1 から 3 まで
        NPV[i] <- NPV[i] + 50 / (1 + R[i])^j # 現在価値
    }
}
print(NPV) # 11 個の現在価値を表示
```

```
[1] 24.342600 22.185736 20.091563 18.057630 16.081601 14.161256 12.294477
```

```
[8] 10.479248  8.713646  6.995838  5.324074
```

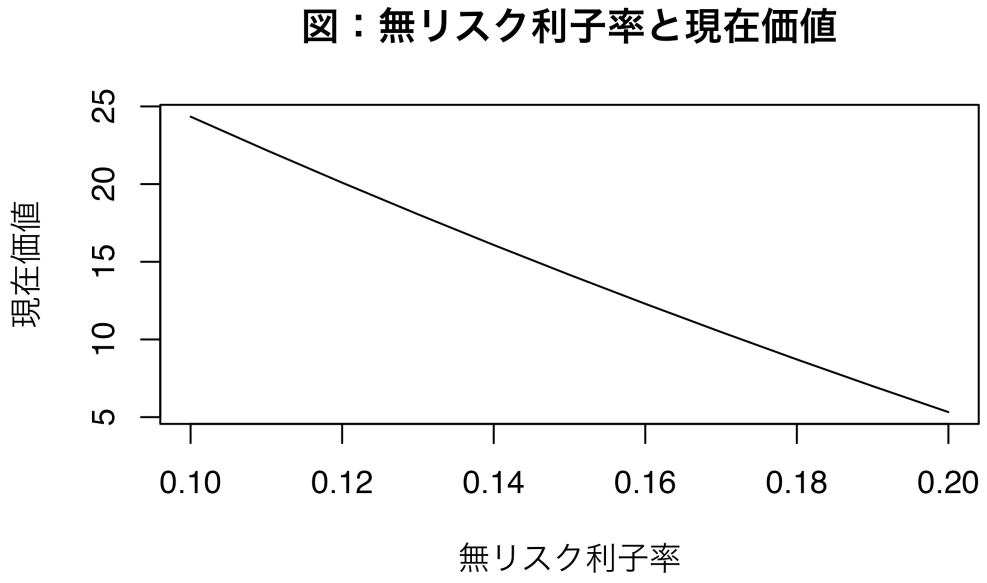
少し複雑な構造しているので、順番に説明します。

- 1 行目は、無リスク利子率のベクトル変数 R を定義しています。ここでは、0.1 から 0.2 まで 0.01 刻みのデータを作成しています。
- 2 行目は、ベクトル変数 R の要素数を N として定義しています。ここでは、N は 11 となります。
- 3 行目は、ベクトル変数 NPV に N 個の NA を代入しています。NA は Not Available の略で、欠損値を表します。NA を代入することで、空っぽの箱が 11 個入ったベクトル変数 NPV を用意します。
- 4 行目から 9 行目は、for 文を使って、NPV の中身を計算しています。for が 2 回出てきているので、二重に繰り返しの処理を行っています。これを**ネスト**と呼びます。1 つのめ for は i が 1 から N(ここでは 11) まで変化し、2 つめの for は j が 1 から 3 まで変化します。1 つめの for 文の i が 1 のとき、次の for 文の j が 1 から 3 までの処理を繰り返し、次に 1 つめの for 文の i が 2 のとき、次の for 文の j が 1 から 3 までの処理を繰り返し・・・という順番で処理が行われます。
- 10 行目は、NPV の中身を表示しています。

この結果をグラフにしてみます。

```
plot(
    x = R, # x 軸のデータ
    y = NPV, # y 軸のデータ
    xlab = "無リスク利子率", # x 軸のラベル
    ylab = "現在価値", # y 軸のラベル
    main = "図：無リスク利子率と現在価値", # グラフのタイトル
    type = "l" # 線グラフ
)
```

TAB キーを使って、インデントを行い、ソースコードのまとまりをわかりやすくしています。インデントは、プログラムの構造をわかりやすくするために行います。インデントを行うときは、半角スペース 2 つか 4 つを使います。どちらを使っても構いませんが、どちらかに統一することが大切です。



3.3.0.1　ベクトル化

上のコードは、for 文を使って、NPV の中身を計算しています。しかし、R 言語では、for 文を使わずに、ベクトルを使って、同じことを行うことができます。このように、for 文を使わずに、ベクトルを使って処理を行うことを**ベクトル化**と呼びます。ベクトル化を行うと、処理が高速化されることがあります。

```
R <- 0.1 # 無リスク利子率 10%
CF <- c(-100, 50, 50, 50) # キャッシュ・フローのベクトル
year <- 0:3 # 年度のベクトル
PV_CF <- CF / (1 + R)^year # 各期の現在価値を計算
NPV <- sum(PV_CF) # 現在価値の合計
print(NPV)
```

```
[1] 24.3426
```

3.4　独自関数の定義の仕方

プログラミングの基本要素である

- 繰り返し
- 分岐
- **関数**

の作り方について説明します。R では自分で関数を定義することができます。関数を定義することで、同じ処理を何度も書く必要がなくなり、プログラムの見通しがよくなります。例えば、足し算をする関数 my_add() を定義してみます。

```r
my_add <- function(x, y){
    x + y
}
```

この関数の構造は、

```r
好きな関数名 <- function(引数 1, 引数 2){
    処理内容
}
```

となっています。つまり、この独自関数 `my_add()` は、`x` と `y` という 2 つの引数 (ひきすう) を足し合わせる関数です。数学的に書くなら、

$$f(x, y) = x + y$$

となります。これは f という関数は 2 つの引数を足す関数であるという意味になっています。作成した独自関数 `my_add()` を使ってみます。

```r
my_add(1, 2)
```

```
[1] 3
```

3 が出力されました。

このように、独自関数を作成する場合には、

1. どのような引数を与えるのか？
2. それに対してどのような処理を行うのか？
3. 最終的にどの値を返す (出力させる) のか？

を考えておく必要があります。

では今までの流れで、現在価値を計算する関数を作成してみます。変化させたい値は、キャッシュフロー `CF` と無リスク利子率 `R` なので、その 2 つを引数とする独自関数を作成します。少し注意する必要がある点として、以下の計算例では `CF` の 1 番目の要素は初期投資額となることに注意しましょう。

```r
calc_PV <- function(CF, R) {
    PV <- CF[1] # 初期投資額なのでマイナスの値
    for (i in 2:length(CF)) { # i は 2 から CF の要素数まで
        PV <- PV + CF[i] / (1 + R)^(i - 1) # 現在価値
    }
    return(PV) # 現在価値を返す
}
```

この関数 `calc_PV()` を使って、現在価値を計算してみます。

```r
calc_PV(c(-100, 50, 50, 50), 0.1)
```

```
[1] 24.3426
```

ちゃんと計算されました。この関数を使って、無リスク利子率が 0.1 から 0.2 まで 0.01 ずつ変化した場合の現在価値を計算してみます。

```r
CF <- c(-100, 50, 50, 50) # キャッシュ・フローのベクトル
R <- seq(0.1, 0.2, 0.01) # 無リスク利子率のベクトル
calc_PV(CF,R)
```

```
 [1] 24.342600 22.185736 20.091563 18.057630 16.081601 14.161256 12.294477
 [8] 10.479248  8.713646  6.995838  5.324074
```

計算されました。関数の引数にデフォルトで値を設定することで、入力を楽にすることができます。例えば、無リスク利子率のデフォルト値を 0.1 に設定してみます。

```r
calc_PV <- function(CF, R = 0.1) {
    PV <- CF[1] # 初期投資額なのでマイナスの値
    for (i in 2:length(CF)) { # i は 2 から CF の要素数まで
        PV <- PV + CF[i] / (1 + R)^(i - 1) # 現在価値
    }
    return(PV) # 現在価値を返す
}
```

すると、無リスク利子率を指定しなくても、デフォルト値が使われるようになります。

```r
CF <- c(-100, 50, 50, 50) # キャッシュ・フローのベクトル
calc_PV(CF)
```

```
[1] 24.3426
```

ただ計算を間違えるもとにもなるので、なるべく省略せずに、しっかり書くことが大事です。

3.4.0.1　もっと凝った独自関数

繰り返し、分岐、関数というプログラミングの基本要素を勉強したので、もう少し複雑なプログラムを作成してみます。

まずは、引数に正の数字以外のもの、あるいは文字列を入力した場合にエラーを表示する関数を作成します。

```r
calc_PV_new <- function(CF, R) {
    if (R <= 0) {
        stop("無リスク利子率は正の値を入力してください。") # エラー処理
    }
```

```
    if ( !is.numeric(CF) ) {
        stop("キャッシュ・フローは数値を入力してください。")
    }
    if ( !is.numeric(R) ) {
        stop("無リスク利子率は数値を入力してください。")
    }

    PV <- CF[1]
    for (i in 2:length(CF)) {
        PV <- PV + CF[i] / (1 + R)^(i - 1)
    }
    return(PV)
}
```

できました。ついでに、NPV の計算結果とともに、NPV が 0 より大きい場合にのみ、「プロジェクトを実行！」と表示する機能も実装してみます。

```
calc_PV_new <- function(CF, R = 0.1) {
    if (R <= 0) {
        stop("無リスク利子率は正の値を入力してください。") # エラー処理
    }
    if ( !is.numeric(CF) ) {
        stop("キャッシュ・フローは数値を入力してください。")
    }
    if ( !is.numeric(R) ) {
        stop("無リスク利子率は数値を入力してください。")
    }

    PV <- CF[1]
    for (i in 2:length(CF)) {
        PV <- PV + CF[i] / (1 + R)^(i - 1)
    }

    if (PV >= 0) { # NPV が 0 より大きい場合
    paste0("NPV が", round(PV, digits = 2), "なので、プロジェクトを実
    ↪ 行！") # 文字列を表示
    } else {
    paste0("NPV が", round(PV, digits = 2), "なのでプロジェクト中止！") #
    ↪ 文字列を表示
```

```
    }
}
```

いろいろ駆使してより短く簡単に書くなら、

```
calc_PV <- function(CF, R = 0.1) {
  if (!is.numeric(CF) || !is.numeric(R) || R <= 0) {
    stop("キャッシュ・フローと無リスク利子率は数値を入力し、無リスク利子率は正
     ↪ の値を入力してください。")
  }
  PV <- sum(sapply(1:length(CF), function(i) CF[i] / (1 + R)^(i - 1)))

  if (PV >= 0) {
    paste0("NPV が", round(PV, digits = 2), "なので、プロジェクトを
     ↪ 実行！")
  } else {
    paste0("NPV が", round(PV, digits = 2), "なのでプロジェクト中止！")
  }
}
```

```
CF <- c(-100, 50, 50, 50)
calc_PV(CF)
```

```
[1] "NPV が 24.34 なので、プロジェクトを実行！"
```

うまくいきました。ちょっとキャッシュフローのベクトルを変化させて、初期投資を-200 にすると、

```
CF <- c(-200, 50, 50, 50)
calc_PV(CF)
```

```
[1] "NPV が-75.66 なのでプロジェクト中止！"
```

ちゃんと中止のメッセージが出ました。

このように、分岐、繰り返し、関数を駆使して、様々なプログラムを作成することができます。プログラミングの基本要素を使いこなせるように、練習を重ねてください。まずは教科書に書いてあるソースコードを自分の PC 上で実行してみてください。その際は、コピペせずに自分で入力するようにしてください。

3.4.0.2　付録：ベクトル化で早くなるのか？

どれほど高速化されるのかを確認するため、100 万年分の現在価値を計算してみます。最初に松浦の R 環境を確認してみます。Mac mini で、CPU は M1、メモリは 8GB です。

```
R.version
```

```
               _
platform       aarch64-apple-darwin20
arch           aarch64
os             darwin20
system         aarch64, darwin20
status
major          4
minor          5.0
year           2025
month          04
day            11
svn rev        88135
language       R
version.string R version 4.5.0 (2025-04-11)
nickname       How About a Twenty-Six
```

まずは、for 文を使って計算してみます。for で現在価値を計算する関数を作成します。この関数では、繰り返し毎期の現在価値を計算し、それを足し合わせていく、という処理を繰り返し行っています。

```
calc_PV_for <- function(CF, R = 0.1) {
    PV <- CF[1] # 初期投資額なのでマイナスの値
    for (i in 2:length(CF)) { # i は 2 から CF の要素数まで
        PV <- PV + CF[i] / (1 + R)^(i - 1) # 現在価値
    }
    return(PV) # 現在価値を返す
}
```

プログラムの実行時間を計算するため、system.time() 関数を使います。

```
CF <- c(-100, rep(50, 10^6))
(res_for <- system.time(calc_PV_for(CF)))
```

```
   user  system elapsed
  0.042   0.001   0.043
```

0.043 秒かかりました。次に、ベクトル化した計算結果を見てみましょう。

```
calc_PV_vec <- function(CF, R = 0.1){
    year <- 0:(length(CF) - 1) # 年度のベクトル
    PV_CF <- CF / (1 + R)^year # 各期の現在価値を計算
```

```
    NPV <- sum(PV_CF) # 現在価値の合計
    return(NPV)
}
```

ベクトルから直接現在価値を計算する関数を用いた計算速度を測ってみます。

```
CF <- c(-100, rep(50, 10^6))
(res_vec <- system.time(calc_PV_vec(CF)))
```

```
   user  system elapsed
  0.010   0.001   0.010
```

0.01 秒と、for 文を使った場合に比べて、約 4.3 倍高速化されました。これがベクトル化による実行速度の効率化です。とはいえ、演算に時間がかかるような大規模データや複雑なシミュレーションをするようになるまで、ベクトル化の恩恵はそれほど大きくないので、まずは読みやすく、確実に動くプログラムを書くことを心がけましょう。

3.5　演習

各自でやってみてください。

3.6　データフレーム入門

3.6.1　CSV ファイルの読み込み

R で CSV ファイルを読み込むには、readr パッケージの read_csv() 関数を使うのがよいでしょう。たとえば、data.csv というファイルを読み込むには、次のように書きます。

```
df <- readr::read_csv("data/ch03_daily_stock_return.csv")
```

この df というオブジェクトの型を見てみると,

```
class(df)
```

```
[1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

spec_tbl_df, tbl_df, tbl, data.frame という 4 つの型が表示されます。data.frame という属性が含まれており，df が**データフレーム**であることを示しています。

あとは,

- nrow() 関数で行数を確認
- str() 関数で構造を確認
- mean() 関数で平均を計算
- sd() 関数で標準偏差を計算

spec_tbl_df は，readr パッケージが読み込んだデータフレームに付与する属性で，データの仕様情報を保持しています。tbl_df は，tibble パッケージが提供するデータフレームの拡張型で，表示や操作がしやすくなっています。tbl は，dplyr パッケージが提供するデータフレームの拡張型で，データ操作が効率的に行えるようになっています。基本的には，data.frame として扱うことができます。

- cor() 関数で相関係数を計算

などを使って，データの中身を確認してみましょう。

- which.min() 関数や which.max() 関数で最小値・最大値のインデックスを取得

例えば，最も日次リターンが高い日付を調べるには，次のようにします。

```
best_day_ID <- which.max(df$firm1)
best_day_ID
```

```
[1] 13
```

13 行目が最も日次リターンが高い日付なので，df からその行を取り出してみます。

```
df$date[best_day_ID]
```

```
[1] "2020-04-17"
```

3.7　ファクター型と日付型

3.7.1　ファクター型入門

ファクター型あるいは因子型 (factor) は，カテゴリカル変数を表現するためのデータ型で，分かりづらいものの，非常に便利かつ重要なデータの型なので，しっかり理解しておきましょう。

数値型や文字列型を因子型に変換する方法には，

1. 基本関数 factor()
2. forcats パッケージの as_factor() 関数
3. forcats パッケージの fct_relevel() 関数

先ほど読み込んだデータの industry 変数は，文字列型ですが，これを因子型に変換してみます。

```
firm_ID <- c(1,2,3)
name <- c("Firm A", "Firm B", "Firm C")
industry <- c("Machinery", "Chemicals", "Machinery")

firm_data <- data.frame(
  firm_ID = firm_ID,
  name = name,
  industry = industry
)
```

この industry 変数はカテゴリー変数ですが，文字列型になっているので，因子型に変換してみます。

```r
firm_data <- firm_data |>
    mutate(
        industry = forcats::fct_inorder(industry)
        )
class(firm_data$industry)
```

```
[1] "factor"
```

forcats パッケージには非常に便利な関数がたくさんあるので，ぜひドキュメントを参照してみてください。代表的なものに

- as_factor()：他の型から因子型に変換
- fct_inorder()：出現順にレベルを設定
- fct_order()：頻度順にレベルを設定
- fct_relevel()：レベルの順序を変更
- fct_recode()：レベルの名前を変更
- fct_collapse()：レベルをまとめる
- fct_lump()：頻度の低いレベルをまとめる

があります。

3.7.2　日付型入門

日付データは lubridate パッケージを使うと便利です。たとえば，文字列型の日付データを日付型に変換するには，ymd() 関数を使います。

```r
df$date
```

```
 [1] "2020-04-01" "2020-04-02" "2020-04-03" "2020-04-06" "2020-04-07"
 [6] "2020-04-08" "2020-04-09" "2020-04-10" "2020-04-13" "2020-04-14"
[11] "2020-04-15" "2020-04-16" "2020-04-17" "2020-04-20" "2020-04-21"
[16] "2020-04-22" "2020-04-23" "2020-04-24" "2020-04-27" "2020-04-28"
[21] "2020-04-30"
```

```r
class(df$date)
```

```
[1] "Date"
```

read_csv() 関数で読み込んだ date 変数は，すでに Date 型になっています。もし，文字列型の日付データを Date 型に変換したい場合は，次のようにします。

```r
df <- df |>
    mutate(
        date = ymd(date) # 年月日
    )
```

3.8 外部パッケージ

pacman パッケージの p_load() 関数を使うと，CRAN に登録されているパッケージのうち，必要なパッケージを一括でインストール・読み込みできます。たとえば，tidyverse パッケージと skimr パッケージをインストール・読み込みするには，次のようにします。

```
pacman::p_load(tidyverse, skimr)
```

まれに，開発中のぱッケージを GitHub からインストールしたい場合があります。その場合は，pacman パッケージの p_load_gh() 関数を使います。たとえば，username/repo という GitHub リポジトリからパッケージをインストール・読み込みするには，次のようにします。

```
pacman::p_load_gh("username/repo")
```

これで，GitHub からパッケージがインストールされ，読み込まれます。

第 4 章

財務データの取得と可視化

```
pacman:: p_load(
    tidyverse,
    ggthemes
)
mystyle <- list(
  theme_economist_white(
    # gray_bg = FALSE,
    base_family = "HiraKakuProN-W3"
    ),
  scale_colour_economist(),
  theme(
    text = element_text(size = 12), # フォントファミリーは上で指定済みなの
    ↪ で省略可
    axis.title = element_text(size = 12)
  )
)
```

4.1 ディスクロージャー制度の概要とデータの入手先

4.1.1 法定開示と適時開示

	年次開示	四半期開示	重要事実
法定開示	有価証券報告書	四半期報告書	臨時報告書
適時開示	決算短信	四半期決算短信	適時開示

4.1.2　財務データの入手先

- **EDINET**：金融庁が運営する電子開示システムで，全上場企業の法定開示資料をデータベースとして提供
- **TDnet**：東京証券取引所が運営する電子開示システムで，上場企業の決算短信をデータベースとして提供

XBRL(eXtensible Business Reporting Language) 形式で財務諸表などの主要情報を公開しています。XBRL からデータを読み込むスキルは本書の枠を超えるため，ここでは練習用データで分析しますが、立命館大学では日経 NEEDS を利用して財務データを収集します。

4.2　R を利用した財務データの分析

4.2.1　tidyverse パッケージの概要

tidyverse とは，R 神 Wickham 氏が基本コンセプトを設定し，**整然データ** (tidy data) に対して一貫した記法でデータを扱えるパッケージ群です。インストールと読み込みは以下の通りです。

```
#install.packages("pacman") # 初回だけ
pacman::p_load(tidyverse) # 読み込み
```

`tidyverse` パッケージを読み込むことで，次の代表的なパッケージが読み出されます。よく使うものは以下のものになります。

- `ggplot2` データの可視化　めっちゃ使う
- `dplyr` データハンドリング　めっちゃ使う
- `tidyr` tidy データにもっていく　使う
- `readr` データを読み込む めっちゃ使う
- `purrr` 関数型プログラミングで使う　慣れてくると使う
- `tibble` data.frame ではなく tibble にする　あまり使わない
- `stringr` 文字列の加工・操作　ちょいちょい使う
- `forcats` ファクター型変数の操作　そんなに使わない

4.2.2　財務データの読み込み

サポートサイトにある練習用のデータセット `ch04_financial_data.csv` をダウンロードして，作業ディレクトリに置きます。

いままでは csv データを読み込むために，基本関数の `read.csv()` を使ってきましたが、より高速かつオプション指定が柔軟な tidyverse 関数群の 1 つである `readr` パッケージの `read_csv()` 関数を使います。read と csv の間がピリオド . からアンダースコア _ に

作業ディレクトリの場所を確認するには `getwd()` を使います。作業ディレクトリを変更するときは，`setwd()` で作業ディレクトリを絶対パスで指定するとよいでしょう。

変わっているので注意してください。readr パッケージの read_csv() 関数は,

1. データの読み込みが高速かつ型の推論が柔軟
2. 基本の data.frame ではなく，その拡張版である tibble で返す
3. 列名を勝手に変換しない。
4. 文字列を勝手にファクター型にしない (read.csv() だと勝手にファクターになる)。

という利点があります。

Listing 4.1 データの読み込み

```
financial_data <- read_csv("data/ch04_financial_data.csv")
nrow(financial_data) # 行数
```

```
[1] 7920
```

```
ncol(financial_data) # 列数
```

```
[1] 11
```

```
head(financial_data, 5) # 最初の 5 行
```

```
# A tibble: 5 x 11
   year firm_ID industry_ID sales    OX   NFE     X     OA    FA    OL     FO
  <dbl>   <dbl>       <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>  <dbl>
1  2015       1           1 5261.  437.  NA    287. 13006. 3543. 4373.  2481.
2  2016       1           1 5949.  564.  50.7  513. 13866. 4642. 4534.  3960.
3  2017       1           1 6505.  691.  29.5  662. 13953. 7744. 5111.  6159.
4  2018       1           1 6846.  751.  86.5  665. 18818. 7285. 5137. 10124.
5  2019       1           1 7572.  959. 298.   660. 18190  9735. 5488. 11362.
```

この financial_data には、11 個の変数に観測値が 7920 個あることがわかります。

- year：年度
- firm_ID：企業 ID
- industry_ID：産業 ID
- sales：売上高
- OX：事業利益 (operating income)
- NFE：純金融費用 (net financial expenses)
- X：当期純利益 (net income)
- OA：事業資産 (operating assets)
- OL：事業負債 (operating liabilities)
- FE：金融資産 (financial assets)
- FO：金融負債 (financial obligations)

このデータフレームの構造を確認します。

```
glimpse(financial_data)
```

```
Rows: 7,920
Columns: 11
$ year        <dbl> 2015, 2016, 2017, 2018, 2019, 2020, 2015, 2016, 2017, 2018~
$ firm_ID     <dbl> 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4~
$ industry_ID <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ sales       <dbl> 5261.40, 5948.96, 6505.06, 6846.38, 7572.24, 7537.63, 3505~
$ OX          <dbl> 437.49, 564.14, 691.18, 751.29, 958.53, 778.37, 45.82, 51.~
$ NFE         <dbl> NA, 50.667498, 29.543157, 86.486500, 298.049774, -
65.45877~
$ X           <dbl> 286.64, 513.48, 661.64, 664.80, 660.48, 843.83, 40.07, 49.~
$ OA          <dbl> 13005.55, 13865.58, 13952.58, 18818.48, 18190.00, 20462.86~
$ FA          <dbl> 3543.43, 4642.16, 7743.99, 7284.72, 9735.13, 10274.25, 225~
$ OL          <dbl> 4372.96, 4534.22, 5111.22, 5137.28, 5487.96, 5371.38, 1840~
$ FO          <dbl> 2480.72, 3959.70, 6159.02, 10123.91, 11362.22, 13772.15, 2~
```

変数はすべて数値型 double になっていますが、firm_ID と industry_ID はカテゴリーを表す変数ですので、数値型ではなくファクター型に変換します。ついでに year は年度という時間を尺度なので、数値型ではなく factor 型に変換します。ここで重要なのは、year はただのファクター型ではなく、順序のあるファクター型とすることです。

ここでは as.factor() を使います。

```
# firm_ID と industry_ID を factor 型に変換
financial_data <- financial_data |>
  mutate(
    year = as.factor(year),
    firm_ID = as.factor(firm_ID),
    industry_ID = as.factor(industry_ID)
  )
# 確認
glimpse(financial_data)
```

```
Rows: 7,920
Columns: 11
$ year        <fct> 2015, 2016, 2017, 2018, 2019, 2020, 2015, 2016, 2017, 2018~
$ firm_ID     <fct> 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4~
$ industry_ID <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ sales       <dbl> 5261.40, 5948.96, 6505.06, 6846.38, 7572.24, 7537.63, 3505~
$ OX          <dbl> 437.49, 564.14, 691.18, 751.29, 958.53, 778.37, 45.82, 51.~
$ NFE         <dbl> NA, 50.667498, 29.543157, 86.486500, 298.049774, -
```

```
65.45877~
$ X           <dbl> 286.64, 513.48, 661.64, 664.80, 660.48, 843.83, 40.07, 49.~
$ OA          <dbl> 13005.55, 13865.58, 13952.58, 18818.48, 18190.00, 20462.86~
$ FA          <dbl> 3543.43, 4642.16, 7743.99, 7284.72, 9735.13, 10274.25, 225~
$ OL          <dbl> 4372.96, 4534.22, 5111.22, 5137.28, 5487.96, 5371.38, 1840~
$ FO          <dbl> 2480.72, 3959.70, 6159.02, 10123.91, 11362.22, 13772.15, 2~
```

4.3 探索的データ分析

4.3.1 データセットの概要確認

データセットを操作するまえに，データの概要を大まかにつかむ必要があり，この作業を探索的データ分析 (exploratory data analysis) といいます。仮説などを持たず，とりあえず特徴や構造を理解するための方法です。データセットの概要を確認するために，`skimr` パッケージの `skim()` 関数を用います。

引数の型に応じて自動的に最適な結果を返す機能を**多態性** (polymorphism) といい，多態性をもつ関数を**総称関数** (generic function) という。

```
skimr::skim(financial_data)
```

Table4.2: Data summary

Name	financial_data
Number of rows	7920
Number of columns	11
Column type frequency:	
factor	3
numeric	8
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
year	0	1	FALSE	6	202: 1363, 201: 1356, 201: 1323, 201: 1319
firm_ID	0	1	FALSE	1515	1: 6, 2: 6, 3: 6, 4: 6

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
industry_ID	0	1	FALSE	10	3: 1760, 10: 1702, 7: 1334, 1: 1143

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
sales	0	1	166007.08	381980.29	205.34	16103.33	40430.71	118313.82	3496433.5	▇▁▁▁▁
OX	0	1	7968.91	25951.56	-353606.72	399.28	1602.88	5260.46	398034.5	▁▁▇▁▁
NFE	1	1	64.02	5941.34	-285383.86	-76.43	-1.19	41.36	331035.2	▁▁▇▁▁
X	0	1	7904.88	26910.18	-357624.83	383.27	1586.10	5204.60	572588.7	▁▇▁▁▁
OA	0	1	152272.74	453879.24	216.51	12559.93	30799.29	93469.20	7987936.5	▇▁▁▁▁

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
FA	0	1	80185.35	422852.06	88.43	6835.07	19095.35	52117.92	29250611	▇
OL	0	1	50260.88	148602.43	35.04	3964.84	10868.33	33110.89	2817974	▇
FO	0	1	70680.61	290625.74	43.64	3757.44	11125.20	35446.03	7026923	▇

さらに，データセットのある変数に含まれる固有な要素を抽出するには，unique() 関数を用います。

```
unique(financial_data$year) # financial_data の year 変数に含まれる固有要素
```

```
[1] 2015 2016 2017 2018 2019 2020
Levels: 2015 2016 2017 2018 2019 2020
```

```
# 2015, 2016, 2017, 2018, 2019, 2020
```

固有要素の数を確認するには，unique() 関数で取り出した要素の数を length() 関数で返します。企業-年の企業数と年度数を確認するには次のようにします。

Listing 4.2 企業数と産業数の確認

```
sort(unique(financial_data$year))
```

```
[1] 2015 2016 2017 2018 2019 2020
Levels: 2015 2016 2017 2018 2019 2020
```

```
financial_data$firm_ID |> n_distinct()
```

```
[1] 1515
```

```
financial_data$industry_ID |> n_distinct()
```

```
[1] 10
```

よってこのデータには 10 の産業、1515 の企業があることが分かります。

4.4　4.3.2 欠損データの処理

ほとんどのデータセットには，欠損値 (NA) が含まれているため，この欠損値の処理は非常に重要になります。欠損値の有無を確認するためには，complete.cases() 関数を用いるのが便利です。欠損値が含まれていると FALSE を返し，欠損値がないと TRUE を返します。

```
head(complete.cases(financial_data)) # 最初の６行の結果を表示
```

```
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

sum() 関数で，TRUE の個数を数え上げることもできます。

```
sum(complete.cases(financial_data)) # TRUE/FALSE を 1/0 に置き換えて合計
```

```
[1] 7919
```

欠損値の出現に何らかの傾向がある場合，欠損値の削除が**生存者バイアス** (survivorship bias) をもたらす可能性があります。たとえば，過去 20 年間にわたって連結財務諸表データに欠損値が含まれていない上場企業ばかりを分析すると，途中で倒産したり上場したりした企業は削除され，20 年間経営し続けている優良企業しかデータに残らない生存者バイアスが発生します。

このようなバイアスを考慮しなくても良いなら，欠損値をもつ個体 (unit) のデータ (行) を削除するのが単純な処理となります。このとき，tidyr パッケージに含まれる drop_na() 関数を用いると簡単に欠損値を含む行を削除できます。基本関数の na.omit() でもよいですが，tidyr::drop_na() の方がオプションが豊富なのでおすすめです。

```
nrow(financial_data) # 欠損行を削除する前の行数
```

```
[1] 7920
```

```
nrow(drop_na(financial_data)) # 欠損行を削除した場合の行数
```

```
[1] 7919
```

```
financial_data <- drop_na(financial_data) # 欠損行を削除した上でデータを上
↪ 書き
# この作業には注意が必要である。オリジナルデータはそのまま残しておいたほうが良い
```

欠損値を含む行を削除するのではなく，欠損値に適切な推定値を代入することでサンプルサイズを減らさない方法も開発されていますが，欠損値の出現を説明する確率モデルを

仮定し，その推定値を求める必要があります。

> **Note**
>
> 詳しくは髙橋・渡辺 (2019) 欠損データ処理：R による単一代入法と多重代入法を参照してください。
> さらに欠損値についての議論では，星野・岡田 (2016)「欠測データの統計科学―医学と社会科学への応用」岩波書店がめちゃめちゃ有用です。

第 5 章

4.4 データの抽出とヒストグラムによる可視化

5.1 4.4.1 条件にあうデータの抽出方法

教科書では複数の方法が紹介されているが，このメモでは tidyverse パッケージを用いた方法だけ取り上げます。具体的には，データベース操作のパッケージである dplyr の中の filter() 関数について説明します。さらに magrittr を用いたパイプ演算子 %>% を用いたデータの受け渡しの記法を活用して，可読性の高いソースコードを書くことも紹介します。ここでは dplyr パッケージの filter() 関数であることを明示的に示すため、dplyr::filter() と書いていますが、dplyr パッケージを読み込んでいる場合は filter() と書いても同じです。

```
financial_data_2015 <- financial_data %>%
    dplyr::filter(year == 2015) # year 変数が 2015 のデータを抽出
```

filter() で条件を満たすデータのみを取り出し，それを financial_data_2015 に代入している。

パイプ演算子 %>% は**左のオブジェクトを右の関数の第 1 引数に代入**する，という処理を行います。つまり，x %>% filter(year == 2015) は，filter(x, year == 2015) と同じ意味になります。パイプ演算子を使うことで，データが次の処理に受け渡されていくプロセスが読みやすくなります。たとえば、

1. 欠損値を除去して，
2. 2015 年のデータを抽出し，
3. ROE を計算して，
4. 産業ごとに平均値を出す

というよく使いそうな処理を行いたい場合，tidyverse なら次のように書きます。

```
financial_data %>%
    drop_na() %>% # 欠損値を除去し，
```

```
    filter(year == 2015) %>% # 2015 年のデータを抽出し，
    mutate(ROE = earnings / equity) %>% # ROE を計算し，
    group_by(industry) %>% # 業種コードごとに
    summarise(mean_ROE = mean(ROE)) # mean() で平均値を計算
```

基本関数の場合は、

```
financial_data <- na.omit(financial_data)
financial_data_2015 <- financial_data[financial_data$year == 2015, ]
financial_data_2015$ROE <- financial_data_2015$earnings /
 ↪  financial_data_2015$equity
mean_ROE_by_industry <- aggregate(financial_data_2015$ROE,
        by = list(financial_data_2015$industry),
        FUN = mean)
```

となりますので、上の方が読みやすいことがわかります。

5.2　4.4.2 ヒストグラムによる売上高の可視化

5.2.1　ヒストグラム

ヒストグラム (histogram) は，データの分布を可視化するためのグラフです。ヒストグラムは，連続データを区間に分けて，区間ごとのデータの個数を棒グラフで表現したものです。したがってヒストグラムの棒の高さは、その区間に含まれるデータの個数を表します。

たとえば、例として生徒 100 人の身長データがあるとします。この身長データを R で生成するには，`rnorm()` 関数を使います。

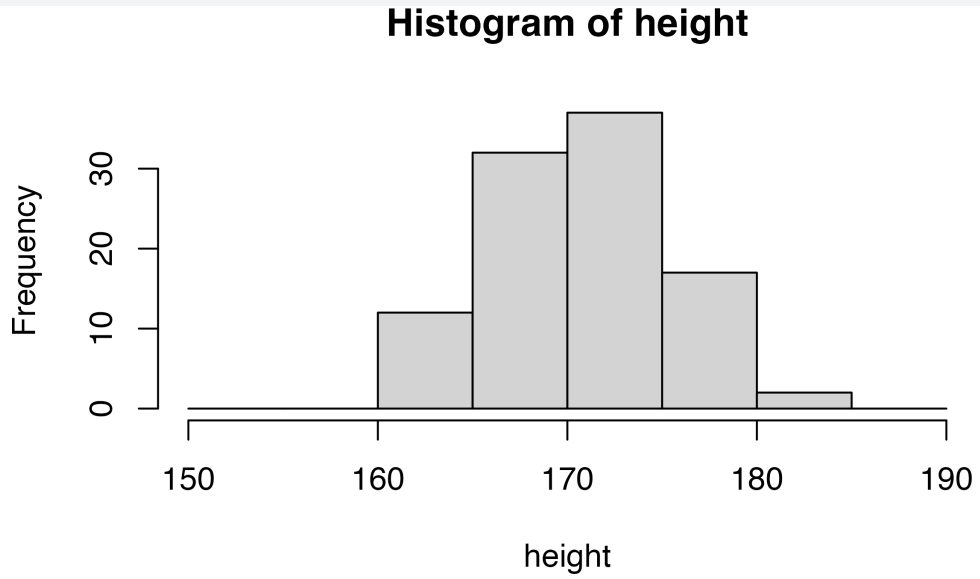
```
# 平均 170cm，標準偏差 5cm の正規分布から 100 個のデータを生成
height <- rnorm(100, mean = 170, sd = 5)
print(height)
```

```
  [1] 172.0614 161.7918 163.5219 176.4785 171.2045 164.9264 176.5255 166.6281
  [9] 162.2730 180.2238 171.9478 174.4014 169.8743 174.7936 171.0020 169.8563
 [17] 168.3494 171.6014 164.4021 167.9890 173.7844 162.4140 164.8660 162.2148
 [25] 168.9842 169.4523 167.8406 169.5611 176.4203 171.2523 165.9491 173.7813
 [33] 171.2189 170.2076 170.9359 163.9504 167.9767 172.2836 167.2427 171.4263
 [41] 165.0325 179.9893 170.1331 175.3006 170.3592 168.3844 180.9103 161.7723
 [49] 176.2633 173.3180 164.1850 177.7006 167.1123 170.6098 168.7652 167.7224
 [57] 179.7148 173.2612 172.7649 168.4119 167.7556 166.7830 166.7060 169.6784
 [65] 175.0206 174.1048 169.1911 166.4146 177.9882 175.3253 177.3805 171.9994
```

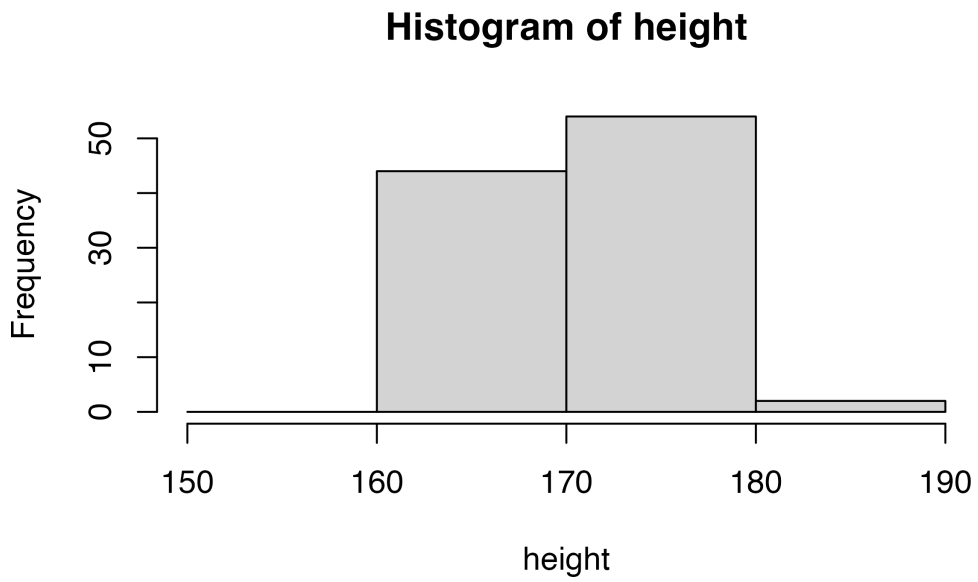
```
 [73] 170.1829 168.0461 166.3599 177.7924 177.1933 177.0200 174.1005 162.0624
 [81] 170.6398 168.4295 170.4855 167.9123 170.4511 170.3181 174.1942 174.6493
 [89] 167.5374 166.7730 169.4637 176.0613 170.9694 170.5105 173.4945 172.0073
 [97] 165.6700 170.4643 171.2456 176.2393
```

100 個のデータを眺めていても、なかなか特徴をつかめませんよね。そこでこの身長という連続データを 5 センチごとの区間に分けます。たとえば、165cm 以上、170cm 未満の区間には何人の生徒がいるのか、170cm 以上、175cm 未満の区間には何人の生徒がいるのか、というように区間ごとのデータの個数を数えます。このとき、区間の幅を 5cm にするか、10cm にするか、20cm にするか、ということは、データの特徴をつかむ上で重要なことです。区間の幅を大きくすると、データの特徴がざっくりとしかつかめません。一方、区間の幅を小さくすると、データの特徴が細かくつかめますが、データの個数が少ない区間が多くなり、データの特徴をつかむのに時間がかかります。やってみましょう。

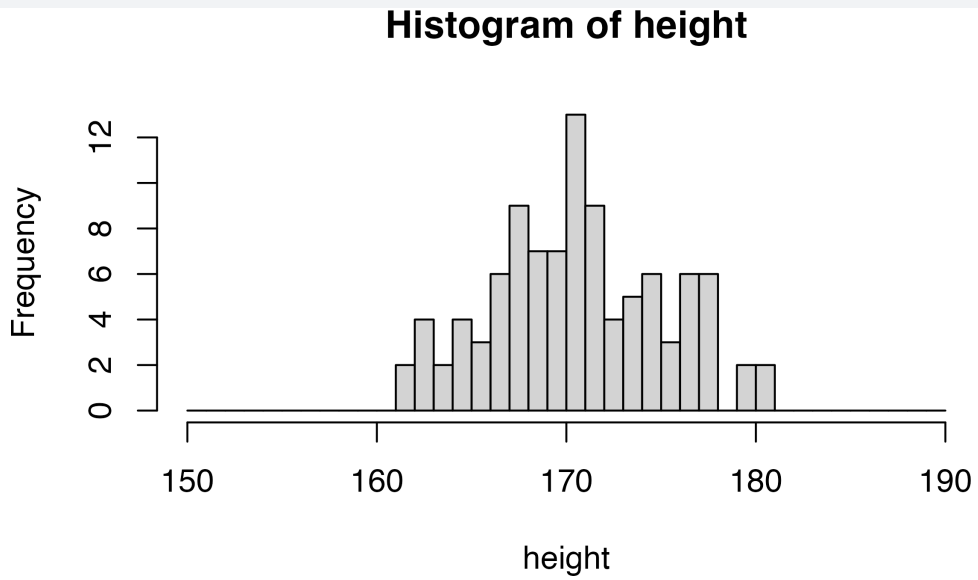
```
hist(height, breaks=seq(150,190,by=5)) # 区間の幅を 5cm にする
```




```
hist(height, breaks=seq(150,190,by=10)) # 区間の幅を 10cm にする
```




```
hist(height, breaks=seq(150,190,by=1)) # 区間の幅を 1cm にする
```



どのヒストグラムがデータの特徴を最も良く表しているのか、を考えて区間幅を設定しましょう。

5.2.2　ggplot でヒストグラム

ヒストグラムを書くためには，基本関数の `hist()` が最も簡単ですが，より高性能な ggplot2 を用いたヒストグラムの書き方を説明します。ここでは、上で作成した 2015 年のデータ `financial_data_2015` を使って、売上高のヒストグラムを書きます。

ggplot2 の書き方は少し特殊ですが、慣れてくると非常に便利です。ggplot2 ではレイヤー (階層) を上から重ねていくようにグラフを作っていきます。まず `ggplot()` 関数

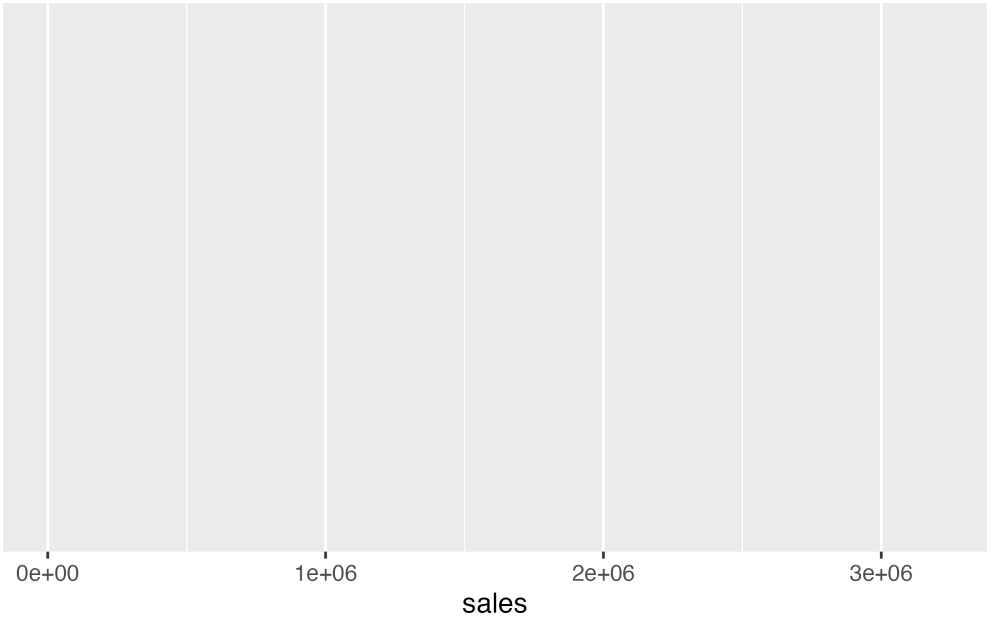
でグラフの土台を作ります。ggplot() に入れるデータの型は data.frame でなければならないので注意しましょう。

```
g <- ggplot(data = financial_data_2015)
print(g)
```


真っ白で何も出力されていませんが、financial_data_2015 というデータフレームを指定して、グラフの土台を作りました。

次に軸の設定をします。ヒストグラムは 1 変数のグラフなので x 軸のみを設定します。aes() 関数で変数を指定します。

```
g <- g + aes(x = sales)
print(g)
```

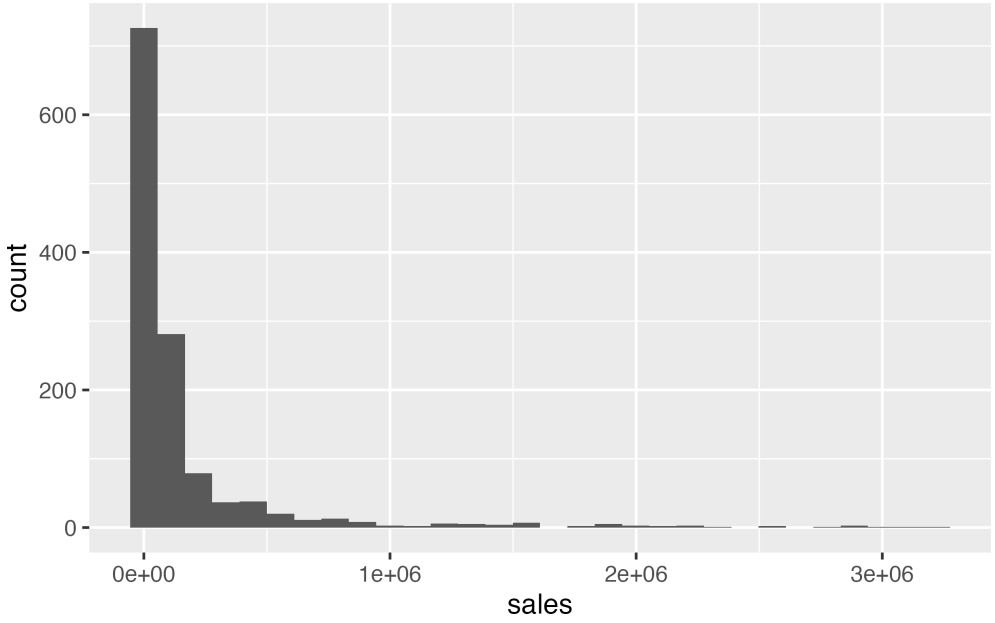


横軸が表示されました。この上に、ヒストグラムを書くために geom_histogram() 関数を追加します。ggplot2 パッケージでは、geom_***の形でグラフを指定します。例えば、

- geom_bar 棒グラフ
- geom_point 散布図
- geom_line 折れ線グラフ
- geom_boxplot 箱ひげ図
- geom_histogram ヒストグラム

あたりがよく使われるグラフです。

```
g <- g + geom_histogram() # グラフはヒストグラム
print(g)
```



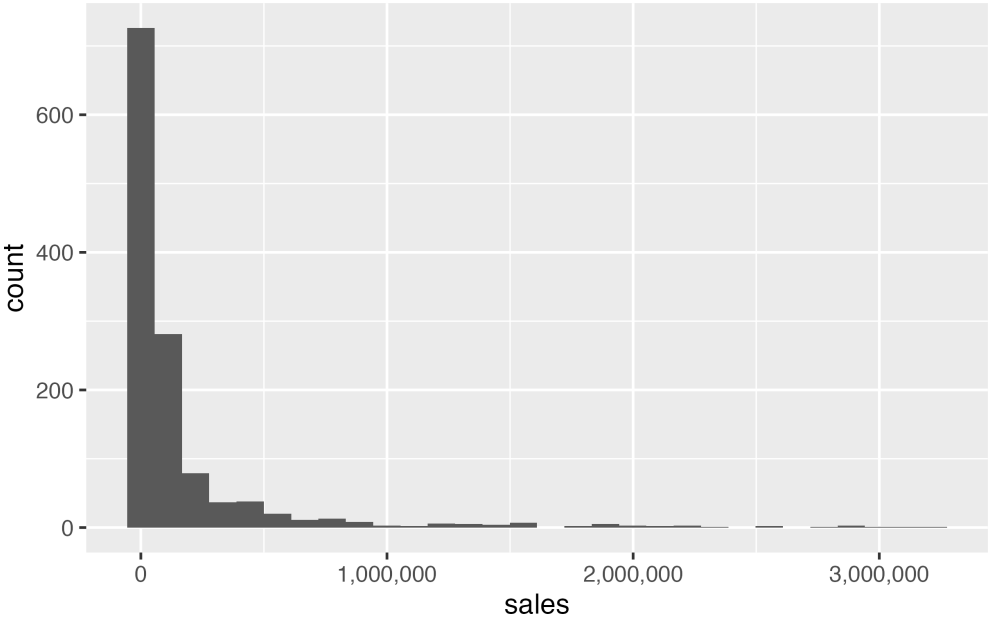
ここでコンソールに，

> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

というメッセージが出ますが、これは「何も指定されなかったので，ヒストグラムのビンの数を 30 にして作図したけど，オプションの stat_bin() で適切な区間幅を binwidth で設定してね」ということです。無視しても大丈夫です。

x 軸が指数表記となっていて見づらいので，`scales()` 関数を使って表記を変更します。

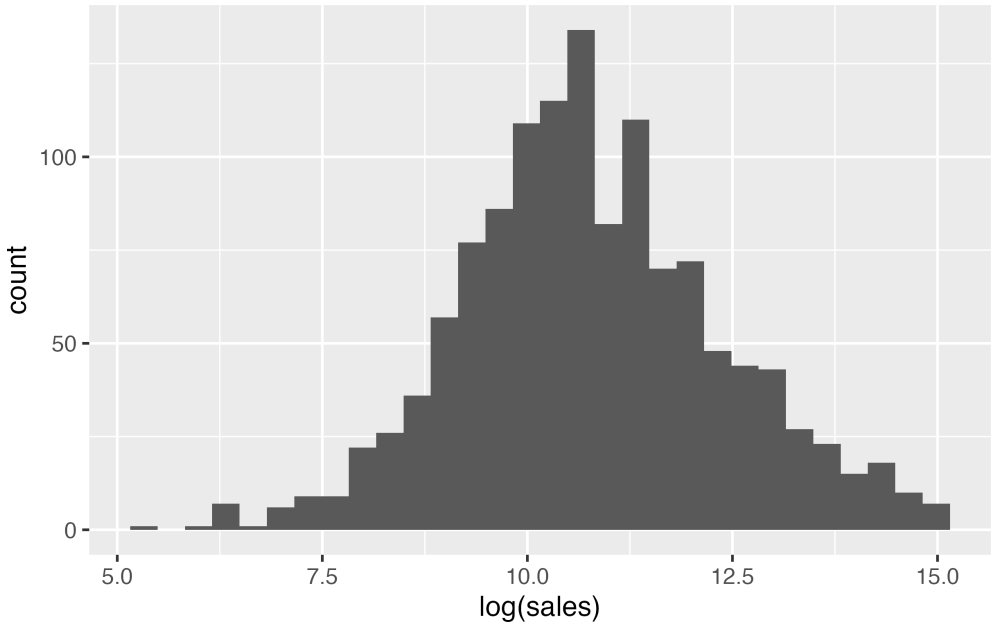
```
g <- g + scale_x_continuous(label = scales::label_comma()) # 3 桁ごとに
 ↪  コンマで区切った数値で表示
print(g)
```



横軸の数値が変化したことが分かります。

また、小数ながら非常に大きな売上高をもつ企業があるため，ヒストグラムの形が左側に集まるように歪んでいます。そこで売上高を自然対数に変換して，分布の歪みを修整したヒストグラムを書いてみます。データを変更するので、最初から全部書きます。

```
g <- ggplot(financial_data_2015) +
  aes(x = log(sales)) + # log() で自然対数変換
  geom_histogram()
print(g)
```



うまくいきました。

第 6 章

4.5 データの集計と折れ線グラフによる可視化

6.1 4.5.3 dplyr を用いた集計

もっとデータを加工して、データの特徴をつかむグラフを作成してみます。データを加工するために、非常に便利なパッケージである tidyverse の `dplyr` を用いたデータ加工を説明します。`dplyr` でよく使う関数に、

- `group_by()`
- `summarise()`
- `mutate()`
- `filter()`

があります。これらの関数を組み合わせることで、データの加工が非常に簡単にできます。たとえば、`financial_data` に含まれる売上高を年度ごとに集計してみましょう。`dplyr` の `group_by()` 関数を使うと、変数を指定してデータをグループ化することができます。たとえば、`year` を指定すると、年度ごとにデータをグループ化します。`group_by()` でグループ化したあとに、`summarise()` 関数を使って平均や分散などの統計量を計算します。

```
N_firms_by_year <- financial_data %>%
    group_by(year) %>% # 年度ごとにグループ化
    summarize( # 以下の統計量を計算
        N_firms = n(), # データ個数 n()
        mean_sales = mean(sales) # 売上高の年度平均 mean()
)
```

これで `N_firms_by_year` というオブジェクトに、`financial_data` を年度ごとにグループ化して、年度ごとの企業数 `N_firms` と平均売上高 `mean_sale` を計算したデータが入っています。2015 年から 2020 年の 6 年間のデータがあるので、6 行のデータが入っているはずです。中身を確認しておきましょう。

```
glimpse(N_firms_by_year)
```

```
Rows: 6
Columns: 3
$ year       <fct> 2015, 2016, 2017, 2018, 2019, 2020
$ N_firms    <int> 1265, 1293, 1319, 1323, 1356, 1363
$ mean_sales <dbl> 173614.9, 173359.5, 170010.9, 157995.4, 160928.2, 161043.7
```

以下の変数について 6 個のデータが入っていることがわかります。

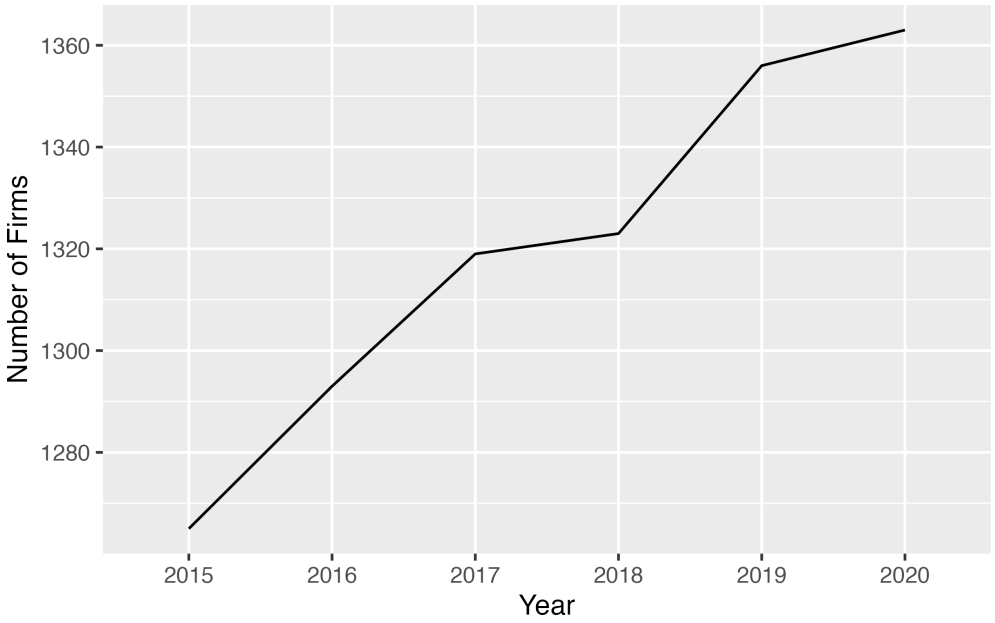
- year：年度 (ord)
- N_firms：企業数 (int)
- mean_sales：平均売上高 (dbl)

> **Note**
>
> 関数型プログラミング (functional programming) は，現代的なプログラミング・パラダイムの 1 種であり，定義された関数を用いて各データに対して行いたい処理を切り分ける。R では apply 系関数として，様々な関数が用意されている。tidyverse 群では，purrr がある。ちょっと難しいですが purrr 超便利

6.2　4.5.4 折れ線グラフによる上場企業数の可視化

データの成形が終わったので，折れ線グラフを作っていきます。ここでは x 軸 (横軸) を年度 year，y 軸 (縦軸) を上場企業数 N_firms とする折れ線グラフを作ってみます。折れ線グラフを作るには geom_line() 関数を使います。

```
g <- ggplot(N_firms_by_year) +
    aes(x = year, y = N_firms, group=1) +
    geom_line()
g <- g + labs(x = "Year", y = "Number of Firms") # 軸ラベル
print(g)
```



ここで突然現れた `group = 1` という `aes()` のオプションですが、これはすべてのデータが同じグループに属していることを指定しています。x 軸にファクター型を指定する場合、`group = 1` を指定しないと、x 軸の値ごとに別のグループとして認識されてしまい、折れ線グラフがうまく描けません。

第 7 章

4.6 変数の作成とヒストグラムによる可視化

tidyverse の dplyr パッケージの mutate() 関数を用いれば，パイプ演算子 %>% を用いて可読性の高いシンプルな書き方で、新しい変数を作成することができます。

7.1 キーボードショートカット

Mac なら command + shift + m でパイプ演算子が入力できます。Windows なら ctrl + shift + m です。

ここでは，ROE(Return on Equity) を計算してみます。ROE の定義は，

$$ROE_t = \frac{X_t}{BE_{t-1}}$$

となります。分子の X_t は t 期の当期純利益，分母の BE_{t-1} は t 期首の株主資本です。練習用データである financial_date.csv には，当期純利益は X という列名で収録されていますが、株主資本の列はありません。よってデータから株主資本は次のように計算します。

$$BE_t = \underbrace{(OA_t - OL_t)}_{NOA_t} - \underbrace{(FO_t - FA_t)}_{NFO_t}$$

この計算を行い，新しい変数 BE をデータフレームに加えるには，dplyr::mutate() を使います。

```
financial_data <- financial_data %>%
    mutate(
        BE = (OA - OL) - (FO - FA) # 新たな BE 変数が加わる
    )
```

分母の株主資本は期首，つまり前期末の数値を用いる必要があります。1 期前の値を参照するには，lab() 関数を用います。ただ，クロスセクションのデータで普通に lag() 関

数を用いると，次のように別の企業のデータを参照してしまいます。

```
financial_data <- financial_data %>%
    mutate(
        lag_BE = lag(BE),
        ROE = X / lag_BE # これはダメ
    )

head(financial_data, 10)[,c("firm_ID", "year", "BE","lag_BE","ROE")]
```

```
# A tibble: 10 x 5
   firm_ID year      BE lag_BE      ROE
   <fct>   <fct>  <dbl>  <dbl>    <dbl>
 1 1       2016  10014.     NA  NA
 2 1       2017  10426. 10014.  0.0661
 3 1       2018  10842. 10426.  0.0638
 4 1       2019  11075. 10842.  0.0609
 5 1       2020  11594. 11075.  0.0762
 6 2       2015   1055. 11594.  0.00346
 7 2       2016   1082.  1055.  0.0468
 8 2       2017   1135.  1082.  0.0702
 9 2       2018   1184.  1135.  0.0763
10 2       2019   1237.  1184.  0.0770
```

結果の `firm_ID` が 2 の企業の 2015 年の ROE を計算するには，1 期前の企業 1 の 2014 年の株主資本を参照する必要があるけれど，2014 年のデータは存在しないため、企業 2 の 2015 年度の ROE は欠損値 `NA` になっている必要があるのに、`lag()` 関数が 1 つ前の企業 1 の 2020 年の株主資本を参照してしまっています。ROE を企業ごとに計算するために，`dplyr::group_by()` を使って，計算を企業群ごとに行うことで、この問題を解決できます。

```
financial_data <- financial_data %>%
    group_by(firm_ID) %>% # firm_ID ごとに以下の処理を繰り返す
    mutate(
        lagged_BE = lag(BE), # lag 関数で前期の値を取り出す
        ROE = X / lagged_BE # これは OK
    ) %>%
    ungroup() # group 化を解除
head(financial_data, 10)[,c("firm_ID", "year",
↪  "BE","lagged_BE","ROE")]
```

```
# A tibble: 10 x 5
```

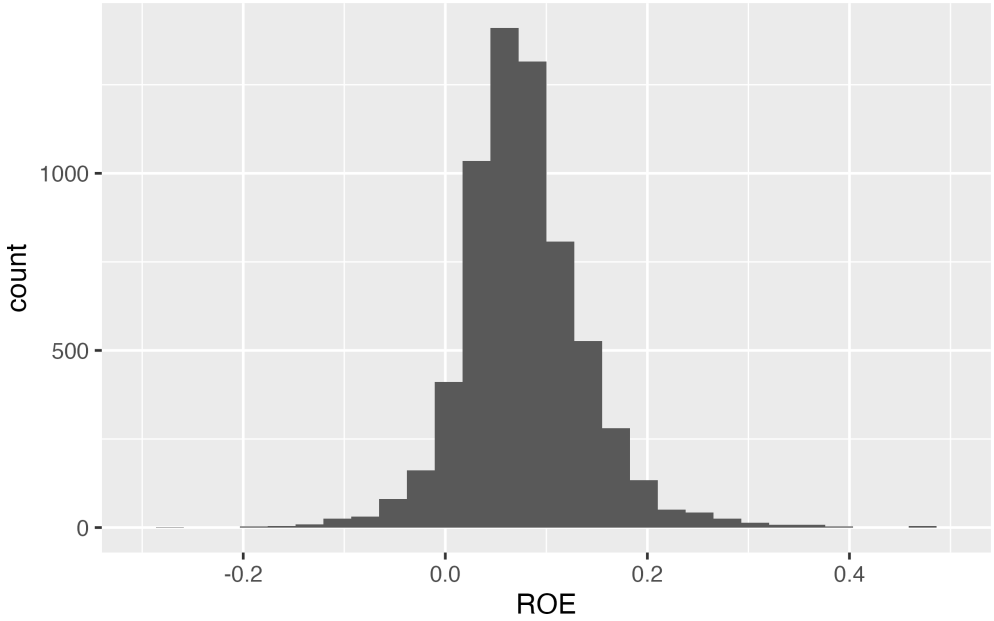
```
   firm_ID year      BE lagged_BE     ROE
   <fct>   <fct>  <dbl>     <dbl>   <dbl>
 1 1       2016  10014.       NA  NA
 2 1       2017  10426.    10014.  0.0661
 3 1       2018  10842.    10426.  0.0638
 4 1       2019  11075.    10842.  0.0609
 5 1       2020  11594.    11075.  0.0762
 6 2       2015   1055.       NA  NA
 7 2       2016   1082.     1055.  0.0468
 8 2       2017   1135.     1082.  0.0702
 9 2       2018   1184.     1135.  0.0763
10 2       2019   1237.     1184.  0.0770
```

2015 年の ROE が欠損値になっており、正しい計算ができています。クロスセクション分析における lag() 関数の問題点を分かりやすくするために、上のように lagged_BE 変数と ROE 変数を別々に作成しましたが、通常は次のように書きます。

```
financial_data <- financial_data %>%
    group_by(firm_ID) %>% # firm_ID ごとに以下の処理を繰り返す
    mutate(
        ROE = X / lag(BE) # これで一気に計算する方が OK
    ) %>%
    ungroup() # group 化を解除
```

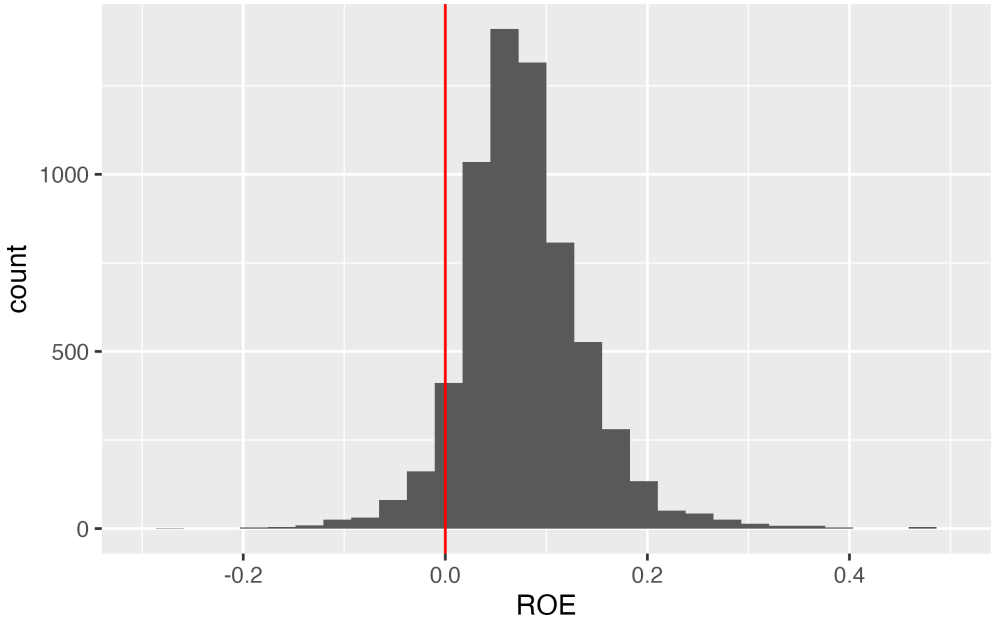
これで ROE の計算ができので、次に ROE のヒストグラムを作ってみます。

```
g <- ggplot(financial_data) + # データの選択
    aes(x = ROE) + geom_histogram()
g <- g + scale_x_continuous(limits = c(-0.3, 0.5)) # x 軸の範囲を調整
print(g)
```



ROE の分布が分かりました。赤字企業が分かりやすいように、ROE がゼロのところに縦線を引いてみます。縦線を引くには geom_vline() を使い、横線を引くには geom_hline() を使います。

```
g <- g + geom_vline(xintercept = 0, color = "red") # x 軸に縦線を引く
print(g)
```



第 8 章

4.7 グループごとの集計とランク付け

8.1 4.7.1 産業ごとの ROE 平均値と棒グラフによる可視化

グループごとに平均値を出すといった処理は，dplyr の group_by() と summarise() を用いることで簡単にできます。

ここでは、産業ごとに ROE の平均値と標準偏差を求めてみます。ROE には欠損値が含まれているため、mean() 関数を使うと NA が返ってきます。NA を無視して平均値を計算するには、mean() 関数のオプション na.rm = TRUE を指定します。

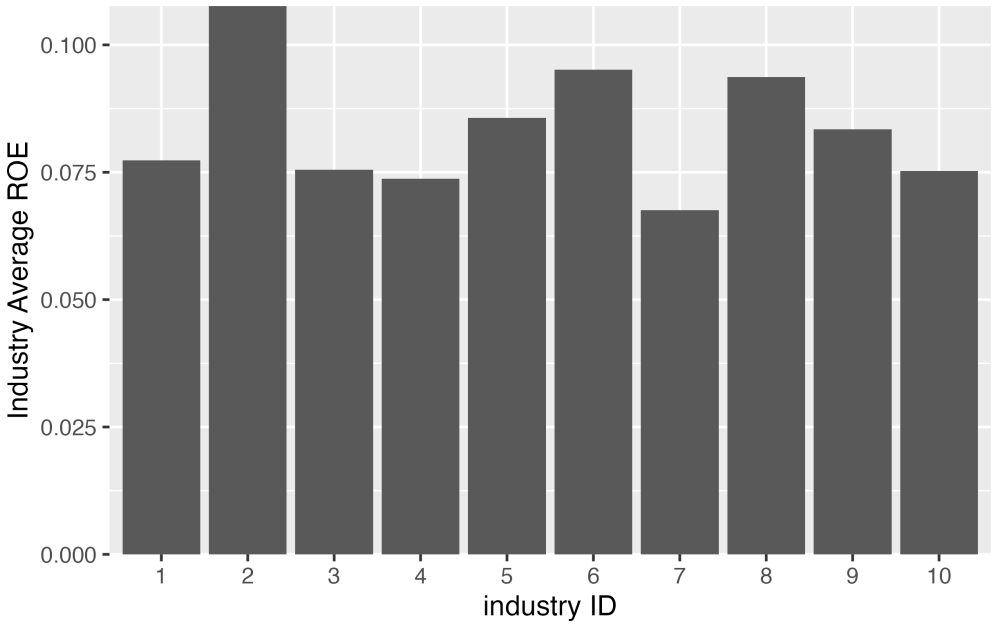
```
df_ind <- financial_data %>%
    group_by(industry_ID) %>% # 集計したいグループを指定
    summarize(
        mean_ROE = mean(ROE, na.rm = TRUE), # 産業平均
        sd_ROE = sd(ROE, na.rm = TRUE)  # 産業標準偏差
    )
glimpse(df_ind)
```

```
Rows: 10
Columns: 3
$ industry_ID <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
$ mean_ROE    <dbl> 0.07731106, 0.10761577, 0.07548896, 0.07371925, 0.08570296~
$ sd_ROE      <dbl> 0.09264764, 0.09530125, 0.05569899, 0.04676826, 0.04859797~
```

10 の産業ごとに統計量を計算したので、10 行のデータが返ってきました。このデータを用いて産業ごとの ROE 平均の棒グラフを作成してみます。

```
# 作図
ggplot(df_ind) + # データフレームを指定
    aes(x = industry_ID, y = mean_ROE) + # 変数を 2 つ指定
```

```
  geom_col() + # 棒グラフ geom_bar() もあるけどこっち
  labs(x = "industry ID", y = "Industry Average ROE") + # ラベル設定
  scale_y_continuous(expand = c(0,0)) # グラフの原点 0,0 に設定
```



教科書では，パイプ処理で直接 ggplot() にデータフレームを渡していますが，個人的に可読性が低くなりオススメできないので，上の例ではデータ操作と作図を分けて書きました。

次に、2020 年度の産業別 ROE ランキングを作ってみます。ROE を大きい順にならべて、一番大きい企業に 1、2 番目の企業に 2、という風にランキングを表す変数を作成するには、`rank(desc())` を使います。`desc()` は降順に並べ替える関数です。

下のソースコードでは、前半のまとまりで、以下の処理を行ったデータを新しいデータフレーム `ROE_rank_data` に代入しています。

1. `financial_data` の中から 2020 年度のデータを抽出し、
2. 必要な変数として `firm_ID`、`industry_ID`、`ROE` の 3 つを選択し、
3. `industry_ID` ごとにグループ化して、
4. `mutate()` 関数で `ROE_rank` 変数を作成し，
5. `ungroup()` 関数でグループ化を解除しています。

後半のまとまりでは、上で作成した `ROE_rank_data` に対して、

1. 産業ごとの ROE ランキング第 1 位の企業を抽出し、
2. ROE が大きい順に並べ替えて、
3. それを `knitr::kable()` 関数で表として出力

という処理をしています。

```
# 2020 年度の産業内の ROE ランキングの変数を作成
ROE_rank_data <- financial_data %>%
    filter(year == 2020) %>% # 2020 年度データを抽出
    select(firm_ID, industry_ID, ROE) %>% # 必要な変数を選択
    group_by(industry_ID) %>% # 産業コードごとに以下の処理を実行
    mutate(
        ROE_rank = rank(desc(ROE)) # ROE_rank 変数を降順で作成
    ) %>%
    ungroup()# グループ化を解除

ROE_rank_data %>%
    filter(ROE_rank == 1) %>% # 各産業のランク 1 のものを抽出
    arrange(desc(ROE)) %>%  # ROE が大きい順
    knitr::kable(booktabs = TRUE, # ここから下は表の装飾
        caption = "2020 年度産業別 ROE ランキング第 1 位企業",
        position = "h!" # 表示場所はここに
        )
```

Table8.1: 2020 年度産業別 ROE ランキング第 1 位企業

firm_ID	industry_ID	ROE	ROE_rank
929	7	0.5641813	1
475	3	0.4975356	1
8	1	0.3882552	1
242	2	0.3749986	1
661	5	0.2673141	1
1042	8	0.2559963	1
1380	10	0.2497929	1
1167	9	0.2346232	1
619	4	0.1491307	1
719	6	0.1422026	1

第 9 章

4.8 上級デュポン・モデルによる ROE の分析とその可視化

上級デュポン・モデルとは，次式で表される ROE の分解式です。

$$ROE_t := \frac{X_t}{BE_{t-1}} = \underbrace{\frac{OX_t}{NOA_{t-1}}}_{RNOA_t} + \underbrace{\frac{NFO_{t-1}}{BE_{t-1}}}_{FLEV_{t-1}} \times \left[\frac{OX_t}{NOA_{t-1}} - \frac{NFE_t}{NFO_{t-1}} \right]$$

各変数の意味は以下の通りです。- `X`：当期純利益 (net income) - `BE`：株主資本 (book of equity) - `OX`：事業利益 (operating income) - `NOA`：純事業資産 (net operating assets) - `NFO`：純金融負債 (net financial obligations) - `NFE`：純金融費用 (net financial expenses)

$RNOA_t$ は，ATO_t と PM_t とに分割できます。

$$\underbrace{\frac{OX_t}{NOA_{t-1}}}_{RNOA_t} = \underbrace{\frac{sales_t}{NOA_{t-1}}}_{ATO_t} \times \underbrace{\frac{OX_t}{sales_t}}_{PM_t}$$

いくつかの変数は，元のデータには含まれていないので，与えられたデータから計算する必要があります。`dplyr::mutate()` 関数を用いて新しい変数を作成し，データフレームに追加します。`lag()` で前期末 (つまり当期首) の値を取得するため，`group_by()` 関数で企業ごとにグループ化しています。

```
financial_data <- financial_data %>%
    group_by(firm_ID) %>% # 企業 ID ごとに以下の計算を行う。
    mutate(
        NOA = OA - OL, # 純事業資産 = 事業資産 - 事業負債
        RNOA = OX / lag(NOA), # 会計上の事業リターン
        PM = OX / sales, # 利ざや profit margin
        ATO = sales / lag(NOA), # 純事業資産回転率
        NFO = FO - FA, # 純金融負債 = 金融負債 - 金融資産
        lagged_FLEV = lag(NFO) / lagged_BE, #期首財務レバレッジ
```

```
        NBC = NFE / lag(NFO), # 債権者のリターン net borrowing cost
        ROE_DuPont = RNOA + lagged_FLEV * (RNOA - NBC) # 上級デュポン・
        ↪ モデルによる ROE
    ) %>%
    ungroup()
```

ROE の分解式が合っているかどうかを確認するため，all.equal() 関数を使って，第 1 引数と第 2 引数が等しいかどうかを判定してみます。普通に計算した ROE と上級デュポン・モデルの分解したものから計算した ROE_DuPoint との比較しています。

```
all.equal(financial_data$ROE, financial_data$ROE_DuPont)
```

```
[1] "Mean relative difference: 4.396878e-06"
```

となり，差の平均は 4.396878×10^{-6} となり，ほぼ 0 となっていることから，上級デュポン・モデルの分解式が正しいことが確認できます。完全にゼロにならない理由は計算の過程で生じる丸め誤差によるものです。

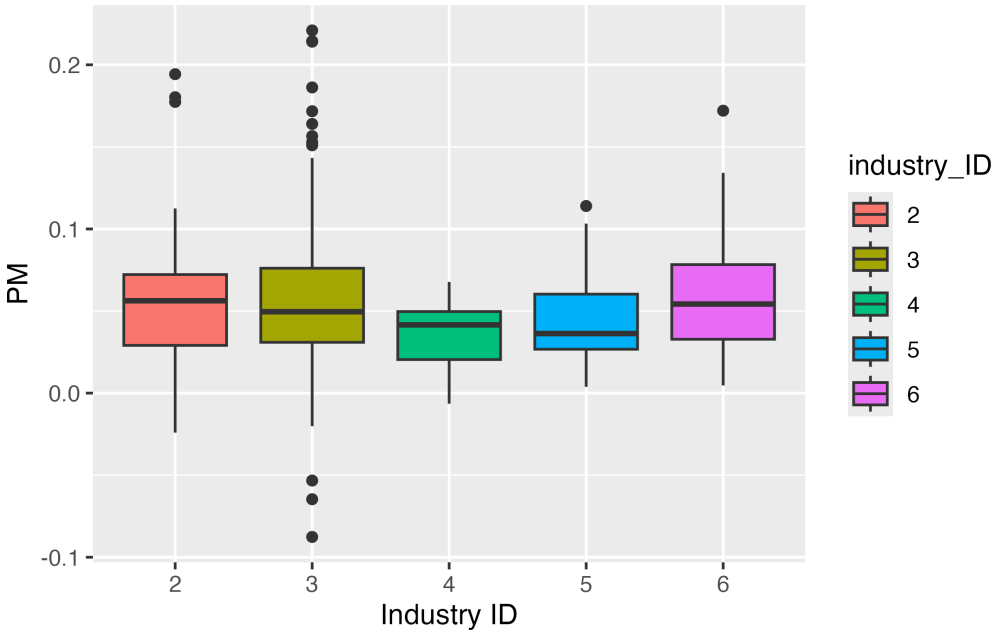
9.1　4.8.2 箱ひげ図による産業別比較

産業別で利ざや PM がどのように分布しているのかを調べるために，箱ひげ図 (box plot) を作ってみます。箱ひげ図は，データの分布を可視化するためのグラフで，第 1 四分位点，中央値，第 3 四分位点，(異常値をのぞく) 最大値，(異常値をのぞく) 最小値を表現できる，非常に情報量の多いグラフです。

ggplot2 パッケージの geom_boxplot() 関数を用いることで，データフレームから箱ひげ図を作図できます。

先に作成したデータフレーム financial_data を用いて，PM の箱ひげ図を作成してみましょう。あまり多くの箱ひげ図を作っても見づらくなるので，最終年度のデータで，産業 ID が 2〜6 までの企業に限定します。

```
df_2020 <- financial_data %>%
    filter(
        year == 2020, # 最終年度
        industry_ID %in% 2:6 # 産業コードが 2 から 6
    )
g <- ggplot(df_2020) +
  aes(x = industry_ID, y = PM, fill = industry_ID) + geom_boxplot() #
  ↪ 箱ひげ図
g <- g + labs(x = "Industry ID")
print(g)
```

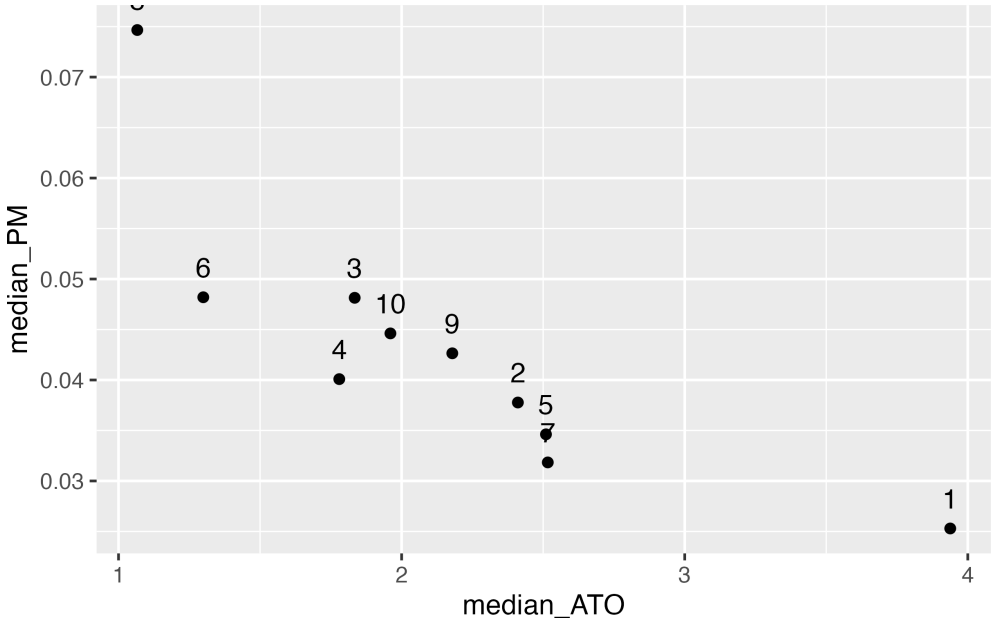


箱ひげ図から，産業ごとに利ざやの分布が異なることがわかります。とりわけ産業 3 は利ざやの散らばりが大きく，産業 4 は非常に散らばりが小さいことが分かります。

9.2　4.8.3 散布図による産業別比較

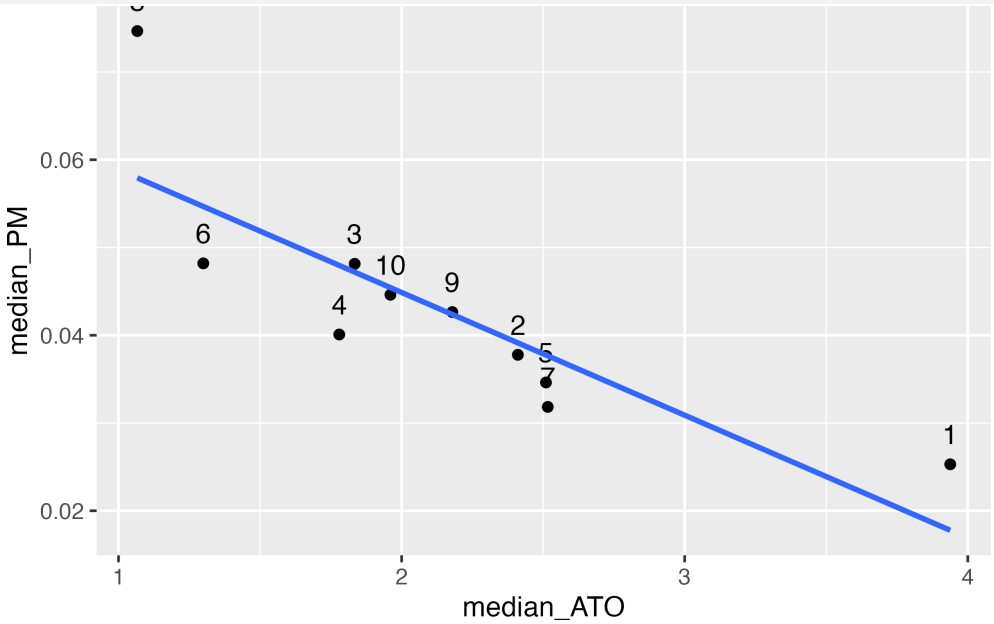
次に産業ごとに ATO(純事業資産回転率) と PM(売上高事業利益率) がどう分布しているか散布図を書いてみます。ここでは異常値の影響を受けにくい統計量である中央値 (median) を計算し，散布図を作成してみます。

```
df_ind_median <- financial_data %>%
    group_by(industry_ID) %>%
    summarise(
        median_ATO = median(ATO, na.rm = TRUE), # ATO の中央値
        median_PM = median(PM, na.rm = TRUE) # PM の中央値
    )

g <- ggplot(df_ind_median) +
    aes(x = median_ATO, y = median_PM, label = industry_ID) + # 散布図
    geom_point() + # 散布図
    geom_text(vjust=-1) # ラベル
print(g)
```



この産業ごとに計算された中央値のデータを用いて，線形回帰直線を引いて，ATO と PM の関係を見てみます。

```
g <- g + geom_smooth(method = "lm", se = FALSE) # 線形回帰直線を追加
print(g)
```

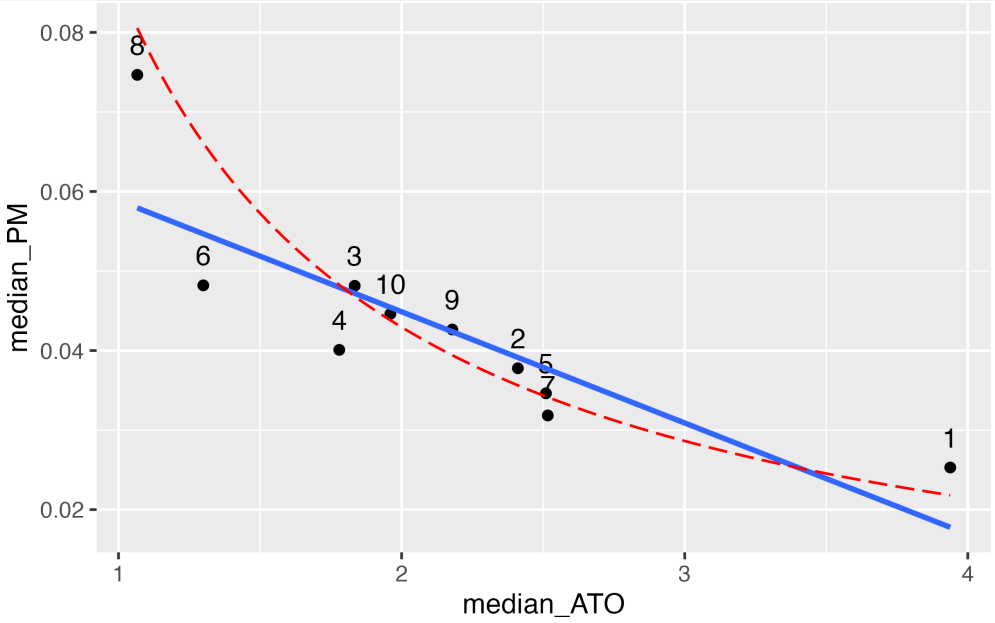


いい感じですが，会計学入門 (1.3.4 節) で学習した $ATO \times RM = RNOA$ という関係のとおり，データからも ATO と PM との間にトレードオフの関係があることが予想されています。もし理論どおりの関係であればデータは $ATO = RNOA/PM$ といった反比例の関係になるはずです。これを示すため，RNOA を一定としたときの ATO と PM の関係，つま

りを図に書き込んでみます。関数をグラフとして図に追加するために stat_function() 関数を用います。

stat_function() 関数の引数は，fun = function(x) x の関数系, linetype = “スタイル” とします。ここでは，function(x) で x の関数であることを指定し，median_RNOA / x としてます。

```
median_RNOA <- median(financial_data$RNOA, na.rm = TRUE) # 全データから
 ↪ 計算した RNOA の中央値

g <- g + stat_function(
    fun = function(x) median_RNOA / x,
    linetype = "longdash",
    color = "red") # 反比例の関数を追加
print(g)
```



かなりあてはまりが良さそうな線が引けました。このように，データを可視化することで，理論とデータの整合性を確認することができます。