2021 年度厚生労働行政推進調査事業費補助金（政策科学総合研究事業）
「医薬品・医療機器等の費用対効果評価における公的分析と公的意思決定方法に関する研究」

R を用いた医療経済評価モデリングの教材事例

分担研究者：森脇 健介
立命館大学 総合科学技術研究機構 医療経済評価・意思決定支援ユニット

1. はじめに

一般に、医療経済評価では長期的なアウトカムの推計のためにモデルを用いたシミュレーションが行われる。決定樹モデルやマルコフモデルなど費用効果分析で用いられるモデルは、様々なパラメータ入力や仮定に依存した数理モデルであり、そこから得られる結果は本質的に不確実である。一般に、医療経済評価のモデルの開発と報告においては透明性が欠如する傾向にある。このため、意思決定者や一般市民は、モデルの信頼性に疑問をもち、その結果を信用しないか誤用する可能性がある。科学的な意思決定には、透明性の確保が極めて重要であり、モデリングにおいてもその重要性は広く認識されている。

従来、モデリングにおいてはエクセルや TreeAge Pro などのソフトウェアが用いられてきたが、有償であることや計算効率の面で課題があり、また過度に複雑なモデルについては、透明性に問題があり、かつ、妥当性の検証が困難となることがある。こうした課題を背景に近年、フリーのソフトウェアである R を用いた医療経済評価のモデリングが普及しつつある。例えば、医療意思決定分析の透明性とオープンソースのソリューションに関心を持つ研究者で構成された団体である Decision Analysis in R for Technologies in Health (DARTH)は、R ベースの意思決定モデルを構築できるような教材を開発することを目的として、2016 年の SMDM でのショートコースを皮切りに年間 5 回程度のワークショップを実施している。また DARTH は R のコードやチュートリアル教材を公開している(https://darthworkgroup.com/)。その他、R for Health Technology Assessment (R for HTA)も同様の取り組みを行っている(https://r-hta.org/)。

モデリングにおいて R を利用するメリットとして下記が挙げられる。

1. 自由に使えるオープンソースであること
2. 任意のモデル構造を実装できる柔軟性があること
3. マウスを用いたポイント&クリックのステップがないこと
4. 計算効率が相対的に高いこと
5. 統計解析とモデリングの統合が可能であること
6. 作図(グラフ等)に柔軟性があること
7. モデルの透明性、再現性、共有性を促進すること

R の学習コストや外部パッケージの多用に伴うブラックボックス化等の課題があるもの

の、今後、教育・研究における R の活用は進むものと考えられる。また将来的には、日本を含め各国の医療技術評価のプロセスにおいても R ベースのモデリングが採用される可能性も考えられる。本研究課題では、DARTH が公開する R による費用効果分析の教材事例の一部をレビューし、解析のスクリプトと出力結果を紹介する。

2. RStudio 操作の基本

2.1 R と RStudio

R は無料で利用できる統計解析ソフトであり、標準的な統計解析のみならず、ユーザーによって開発された最新の統計解析のパッケージをインストールすることにより機能を拡張することも可能である。R 自体はスクリプト（R のプログラムのコード）を入力することにより操作する必要があるため、初学者とって学習のハードルが高いが、近年、R の操作を補助する無料ソフトウェアとして RStudio が利用可能となっている。RStudio では、マウスを用いたクリックによる操作が可能で、データの読み込みや統計解析、スクリプトの補完機能、グラフ出力が容易となっており、近年、医療技術評価における教育・研究においても活用されるに至っている。R および RStudio のインストールについては下記などを参照されたい。

R/RStudio のインストール法 高知工科大学 経済・マネジメント学群 矢内 勇生 氏 ホームページ https://yukiyanai.github.io/jp/resources/ masayukeeeee 氏　ホームページ https://zenn.dev/masayukeeeee/articles/96c7832de4dc40

2.2 RStudio の起動

R と RStudio が正常にインストールされたら、RStudio を起動することにより R を利用できる(R を直接起動する必要はない)。RStudio を起動したら、まず、画面左上の＋アイコンをクリックし、R Script を選択し、新規の R script を開く。RStudio の基本画面は、図のように 4 つのウィンドウ(ペイン：Pane)に分割されている。

2.3 各ペインの機能

①Source

- R のスクリプトを作成・編集・実行するウィンドウである。
- 「Run」アイコンをクリック、または、キーボードで Ctrl＋Enter を押下することで、カーソルがある行のスクリプトが実行される。
- スクリプトの範囲を選択することで指定した領域のスクリプトを実行することができる。

②Console

- 実行したスクリプトと結果が表示される。

③Environment

- Environment タブには、読み込んだデータや解析結果の出力等を格納した**「オブジェクト」**が表示される。オブジェクトをクリックするとその内容が Source ペインの新規タブにて表示される。
- History タブには、過去に実行したスクリプトが保存されており、「To source」アイコンをクリックすると、Source ペインにコピーされる。

④Plots

- Plots タブには、Source ペインで実行したグラフ描画の結果が表示される。グラフは「Export」メニューから形式等を指定して出力することが可能である。
- Help タブでは Source ペインで、?関数名や help(関数名)を実行することで、関数やパッケージに関するヘルプが表示される。
- Package タブには現在インストールされているパッケージの一覧が表示され、チェックボックスにチェックを入れることでそのパッケージが利用可能となる。Source ペインで、library(パッケージ名)を実行することでもパッケージが利用可能となる。

2.4 パッケージの利用

R では、色々な関数や最新の糖液解析を使うための**「パッケージ」**と呼ばれる関数のセットがある。多くのパッケージは CRAN などのウェブサイトに存在し、必要に応じて、ダウンロード、インストールして使うことができる。

手順としてはまず、以下のコードでパッケージをインストールする。

install.pakages("パッケージの名前")　※ダブルコーテーションがあることに注意

インストールが完了したら、次に以下のコードで、パッケージを呼び出し、RStudio 上で利用できる状態にする。

library(パッケージの名前)　※ダブルコーテーションがないことに注意

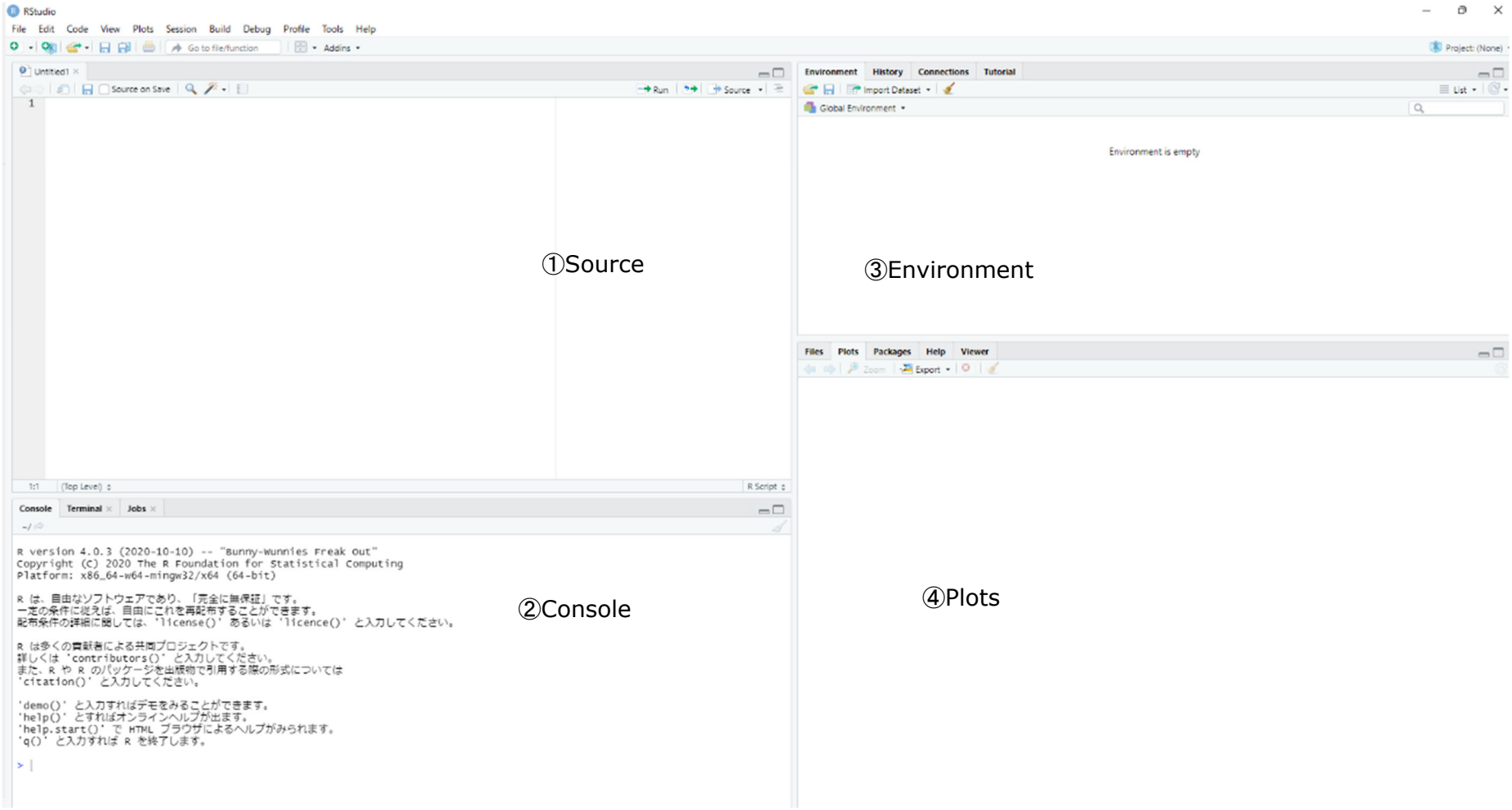

図 2-4-1. RStudio の基本画面

3. DARTH の教材使用の準備

使用ファイル：00_prep.R

まずパッケージ「pacman」をダウンロードし、他のパッケージを便利にインストールするために使用する。

```
if (!require('pacman')) install.packages('pacman'); library(pacman)
```

pacman ： ライブラリやパッケージに関連する関数を便利にラップし、直感的で一貫した方法で名前を付ける。低レベルの機能を組み合わせることで、ワークフローを高速化することを目指すものである。install.pakeges() や library() を羅列することなく、pacman::p_load() の一行でスマートに記載することができる。
参考　https://plaza.umin.ac.jp/shoei05/wp/index.php/2020/10/24/145/

require 関数:
参考　https://justdoit.hatenablog.jp/entry/2015/01/11/231208

CRAN から各種の必要パッケージを読み込む。

```
p_load("here", "devtools", "dplyr", "scales", "ellipse", "ggplot2", "lazyeval", "igraph", "truncnorm", "ggraph", "reshape2", "knitr", "markdown", "stringr")
```

GitHub からパッケージ「dampack」を読み込む。

```
install_github("DARTH-git/dampack", force = TRUE) # (Un)comment if there is a newer version
p_load_gh("DARTH-git/dampack")

dampack:
```

決定樹モデルの必要パッケージを読み込む。

```
install_github("DARTH-git/dectree", force = TRUE) # (Un)comment if there is a newer version
p_load_gh("DARTH-git/dectree")
```

マルコフモデルの必要パッケージを読み込む。

```
p_load("diagram")
```

生存時間分析の必要パッケージを読み込む。

```
p_load("gems", "flexsurv", "survHE", "msm", "mstate")
```

CALIBRATION と VOI(情報価値分析)については本稿では触れないため、必要パッケージの読み込みを省略する。

4. 決定樹モデルに基づく費用効果分析

4.1 演習シナリオ「ウイルス性脳炎の治療」

ウイルス性脳炎は、ヘルペスウイルス（HVE）とその他のウイルス（OVE）によって引き起こされることがある。ヘルペスウイルスは、ウイルス性脳炎の約52%を引き起こす。治療しない場合の合併症（重篤な後遺症）のリスクは、HVE では 71%、OVE ではわずか 1%といわれている。ビダラビンという薬剤は、HVE による合併症の可能性を 71%から 36%に減少させることができる。しかし、OVE 患者では、ビダラビンによる治療は重篤な副作用を伴い、合併症のリスクはベースラインの 1%から 20%に増加する。HVE の確定診断は脳生検で可能だが、この方法自体も 0.5%の確率で致命的となる。

あなたは、3 つの管理戦略（無治療、ビダラビン治療、HVE と診断された人への脳生検とビダラビン治療）に関連する医療費と利益を評価することを課せられている。効果は質調整生存年数（QALYs）で測定される。

合併症のないウイルス性脳炎の場合の医療費は 1,200 ドルだが、合併症が発生すると 9,000 ドルにまで上昇する。ビダラビンの治療費は 9,500 ドル、脳生検は 25,000 ドルの処置である。

合併症なくウイルス性脳炎から回復した患者の QALYs は平均 20 であるが、合併症が発生した患者の QALYs は平均 19 である。脳生検は患者負担の大きい処置であるため、それを受けた患者は生検の結果にかかわらず 1 回限り 0.01QALYs の損失を経験することになる。

パラメータは表 4-1-1 に要約される。本演習では、"decision_tree_HVE_solutions.Rmd "で提供されるコードのテンプレートを使用する。

表 4-1-1. パラメータ設定

Parameter	R name	Value
Prevalence of HVE	p_HVE	0.52
Probability of complications without treatment		
- HVE	p_HVE_comp	0.71
- OVE	p_OVE_comp	0.01
Probability of complications with vidarabine treatment		
- HVE	p_HVE_comp_tx	0.36
- OVE	p_OVE_comp_tx	0.20
Probability of complications due to brain biopsy	p_biopsy_death	0.005
Quality-adjusted life-years (QALYs)		
- Remaining QALYs without VE complications	q_VE	20
- Remaining QALYs with VE complications	q_VE_comp	19
- QALY loss due to brain biopsy	q_loss_biopsy	0.01
Healthcare costs		
- Cost of viral encephalitis without complications	c_VE	$1,200
- Cost of viral encephalitis with complications	c_VE_comp	$9,000
- Vidarabine treatment	c_tx	$9,500
- Brain biopsy	c_biopsy	$25,000
- Willingness-to-pay	wtp	$100,000/QALY

※DARTH 演習資料より引用

4.2 演習課題の例

1. 特に末端ノードでのアウトカム値（コストと QALYs）に着目し、決定樹の描画を行う。
2. コードテンプレートには、「治療なし」戦略の計算が含まれている。他の 2 つの戦略に対する期待される結果を計算するコードを記載せよ。
3. dampack パッケージの calculate_icers() 関数を使用して、10 万ドル/QALY の支払い意思額で、各戦略の増分コスト、QALYs、ICERs、Incremantal NMB を計算する。関数ドキュメントを見るには、「?calculate_icers()」と入力せよ。
4. plot()関数と calculate_icers()関数の出力を使って費用対効果のフロンティアをプロットする。

4.3 スクリプトの例

使用ファイル: decision_tree_HVE_solutions.Rmd

まずワークスペースから変数名等の全設定を削除しておく。

```
rm(list = ls())      # clear memory (removes all the variables from the workspace)
```

4.3.1 パッケージの読み込み

下記のスクリプトを実行することにより、必要なパッケージを読み込む。

```
{r, warning = FALSE, message = FALSE}
if (!require('pacman')) install.packages('pacman'); library(pacman) # use this package to conveniently install other packages
# load (install if required) packages from CRAN
p_load("here", "dplyr", "devtools", "scales", "ellipse", "ggplot2", "lazyeval", "igraph", "truncnorm", "ggraph", "reshape2", "knitr", "stringr")
# load (install if required) packages from GitHub
# install_github("DARTH-git/dampack", force = TRUE) # Uncomment if there is a newer version
# install_github("DARTH-git/dectree", force = TRUE) # Uncomment if there is a newer version
p_load_gh("DARTH-git/dampack", "DARTH-git/dectree")
```

4.3.2 関数の読み込み

必要となるパッケージや関数の読み込みは省略する。

4.3.3 パラメータの設定

パラメータの定義と入力値の設定を行う。

```
v_names_str <- c("No Tx", "Tx All", "Biopsy") # ストラテジーの名前
n_str <- length(v_names_str) # ストラテジーの数
wtp <- 100000 # 支払意思額閾値

# 確率
p_HVE <- 0.52 # HVE の有病率
p_HVE_comp <- 0.71 # 未治療の HVE を合併した場合
p_OVE_comp <- 0.01 # 未治療の OVE を合併した場合
p_HVE_comp_tx <- 0.36 # HVE を治療した場合の合併症
p_OVE_comp_tx <- 0.20 # OVE を治療した場合の合併症
p_biopsy_death <- 0.005 # 生検が原因で死亡する確率

# コスト
c_VE <- 1200 # 合併症がない場合のウイルス性脳炎の治療費
c_VE_comp <- 9000 # 合併症がある場合のウイルス性脳炎の治療費
c_tx <- 9500 # 治療費
c_biopsy <- 25000 # 脳生検の費用
```

```
# QALYs
q_VE <- 20 # VE 関連の合併症がない場合の残りの QALYs
q_VE_comp <- 19 # VE 関連合併症のある人の残り QALYs
q_loss_biopsy <- 0.01 # 脳生検による 1 回限りの QALY 損失
q_death_biopsy <- 0 # 生検中に死亡した人の残存 QALYs

# パラメータをリストに格納する
l_params_all <- as.list(data.frame(p_HVE, p_HVE_comp, p_OVE_comp,
p_HVE_comp_tx, p_OVE_comp_tx, p_biopsy_death.D), p_HVE_comp_tx,
p_OVE_comp_tx, p_biopsy_death,
                                   c_VE、c_VE_comp、c_tx、c_biopsy。
                                   q_VE, q_VE_comp, q_loss_biopsy))
# パラメータの名前をベクトルに格納する
v_names_params <- c('p_HVE', 'p_HVE_comp', 'p_OVE_comp',
'p_HVE_comp_tx', 'p_OVE_comp_tx', 'p_biopsy_death',
                    'c_VE', 'c_VE_comp', 'c_tx', 'c_biopsy', 'q_VE', 'q_VE_comp',
'q_loss_biopsy')
```

上記スクリプトを実行すると、Environment には図 4-3-3-1 のようなパラメータ設定が表示される。

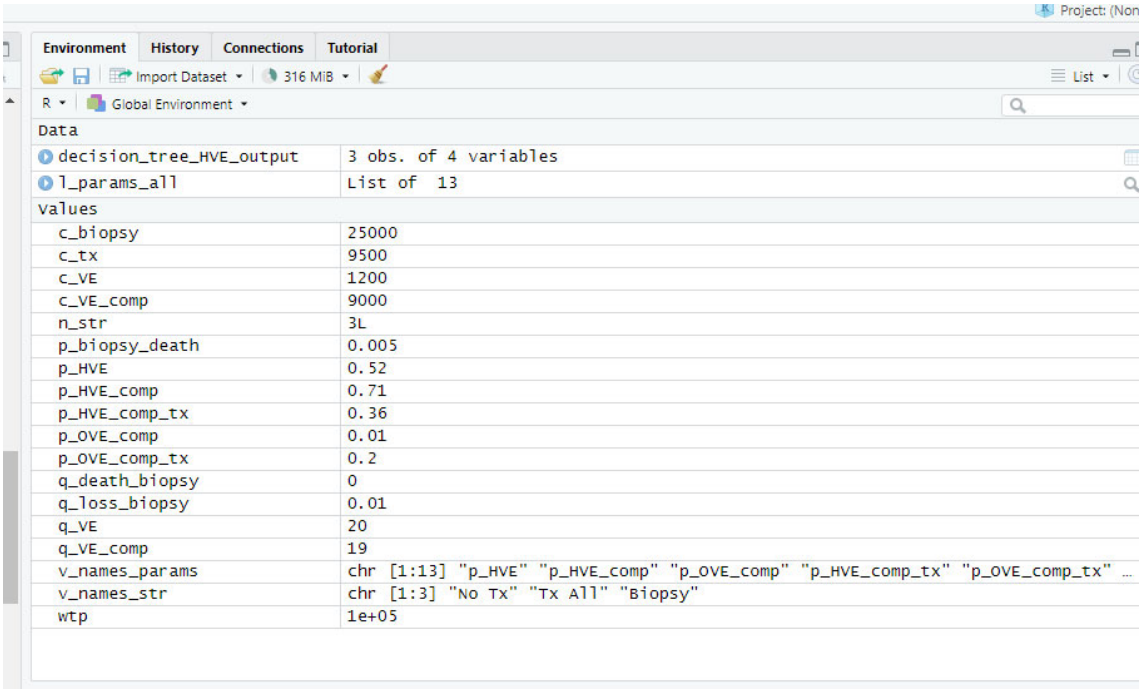

図 4-3-3-1. パラメータ設定

4.3.4 モデルの構築と分析

決定樹モデルを構築し、分析を実行する。

```
decision_tree_HVE_output <- with(as.list(l_params_all), {

 # 各戦略におけるシナリオが生起する確率のベクトルを作成する

 v_w_no_tx <- c( p_HVE * p_HVE_comp , # HVE, 合併症)
                 p_HVE * (1 - p_HVE_comp) , # HVE, 合併症なし
             (1 - p_HVE) * p_OVE_comp , # OVE, 合併症
             (1 - p_HVE) * (1 - p_OVE_comp)).      # OVE、合併症なし

 v_w_tx <- c( p_HVE * p_HVE_comp_tx , # HVE w/tx, 合併症)
                 p_HVE * (1 - p_HVE_comp_tx) , # HVE w/tx, 合併症なし
             (1 - p_HVE) * p_OVE_comp_tx , # OVE w/tx, 合併症
             (1 - p_HVE) * (1 - p_OVE_comp_tx)))   # 合併症なし。

 v_w_biopsy <- c(p_biopsy_death , # 生検による死亡
             # 生検死なし、HVE/tx あり、合併症あり
             (1-p_biopsy_death) * p_HVE * p_HVE_comp_tx ,
             # 生検による死亡なし、HVE/tx あり、合併症なし
             (1-p_biopsy_death) * p_HVE * (1-p_HVE_comp_tx) ,
             # 生検死なし、OVE、合併症あり
             (1-p_biopsy_death) * (1-p_HVE) * p_OVE_comp ,
             # 生検死なし、OVE、合併症なし
             (1-p_biopsy_death) * (1-p_HVE) * (1 - p_OVE_comp)))

 # 各戦略のアウトカム（QALYs）のベクトルを作成する

 v_qaly_no_tx <- c(q_VE_comp , # HVE, 合併症)
                  q_VE , # HVE, 合併症なし
                  q_VE_comp , # OVE, 合併症
                  q_VE) # OVE, 合併症なし

 v_qaly_tx <- c(q_VE_comp , # HVE, 合併症)
```

```
                q_VE , # HVE, 合併症なし
                q_VE_comp , # OVE, 合併症
                q_VE) # OVE, 合併症なし


v_qaly_biopsy <- -q_loss_biopsy + # 生検による損失
              c( q_death_biopsy , # 生検の合併症)
                q_VE_comp , # 生検による合併症なし, HVE w/tx, 合併症あり
                q_VE , # 生検費用なし、HVE あり、合併症なし
                q_VE_comp , # 生検同意なし, OVE, 合併症
                q_VE) # 生検なし, OVE, 合併症なし

# 各戦略のコストのベクトルを作成する

v_cost_no_tx <- c(c_VE_comp , # HVE, 合併症)
               c_VE , # HVE, 合併症なし
               c_VE_comp , # OVE, 合併症
               c_VE) # OVE, 合併症なし

v_cost_tx <- c_tx + # 治療に要した費用
             c(c_VE_comp , # HVE, 合併症
               c_VE , # HVE, 合併症なし
               c_VE_comp , # OVE, 合併症
               c_VE) # OVE、合併症なし


v_cost_biopsy <- c_biopsy + # 生検の費用
              c(0 , # 死亡時のコスト（ゼロ）
               c_VE_comp + c_tx , # 生検費用なし、HVE w/tx、合併症あり
               c_VE + c_tx , # 生検費用なし、HVE w/tx、合併症なし
               c_VE_comp , # 生検補正なし, OVE, 合併症
               c_VE) # 生検費用なし、OVE、合併症なし

# 各戦略の総 QALY を計算する
total_qaly_no_tx <- v_w_no_tx %*% v_qaly_no_tx
total_qaly_tx <- v_w_tx %*% v_qaly_tx
```

```
 total_qaly_biopsy <- v_w_biopsy %*% v_qaly_biopsy

 # 各戦略の総コストを計算する
 total_cost_no_tx <- v_w_no_tx %*% v_cost_no_tx
 total_cost_tx <- v_w_tx %*% v_cost_tx
 total_cost_biopsy <- v_w_biopsy %*% v_cost_biopsy

 # 総 QALYs のベクトル
 v_total_qaly <- c(total_qaly_no_tx, total_qaly_tx, total_qaly_biopsy)
 # 総費用のベクトル
 v_total_cost <- c(total_cost_no_tx, total_cost_tx, total_cost_biopsy)
 # nmb のベクトルを計算する
 v_nmb <- v_total_qaly * wtp - v_total_cost

 # アウトカムに名前を付ける
 names(v_total_qaly) <- v_names_str # 総 QALYs ベクトルの要素につける名前
 names(v_total_cost) <- v_names_str # 総コストベクトルの要素に対応する名前
 names(v_nmb) <- v_names_str # nmb ベクトルの要素に対応する名前

 df_output <- data.frame(Strategy = v_names_str,
                         Cost = v_total_cost,
                         Effect= v_total_qaly,
                         NMB = v_nmb)
 return(df_output)
})

# モデル出力
decision_tree_HVE_output
```

上記のスクリプトの実行の結果、Source には図 4-3-4-1 の出力が表示される。

Description: df [3 x 4]

	Strategy <chr>	Cost <dbl>	Effect <dbl>	NMB <dbl>
No Tx	No Tx	4117.20	19.62600	1958483
Tx All	Tx All	12908.96	19.71680	1958771
Biopsy	Biopsy	32599.41	19.69896	1937297

3 rows

図 4-3-4-1. アウトカムの推計結果

4.3.5 決定樹モデルの描画

決定樹を描画する。なお、決定樹の構成要素を入力したエクセルファイル(decision_tree_HVE_branches.xlsx)をあらかじめ作業中のフォルダに保存しておく必要がある。

```
{r, out.width = '100%', warning=F}.
branches <- read.csv('decision_tree_HVE_branches.csv', stringsAsFactors = F, header = T)
tree <- create_tree(branches)

plot_tree(tree, font.size = 3.15)
```

上記スクリプトを実行すると、Console に決定樹が出力される(図 4-3-5-1)。

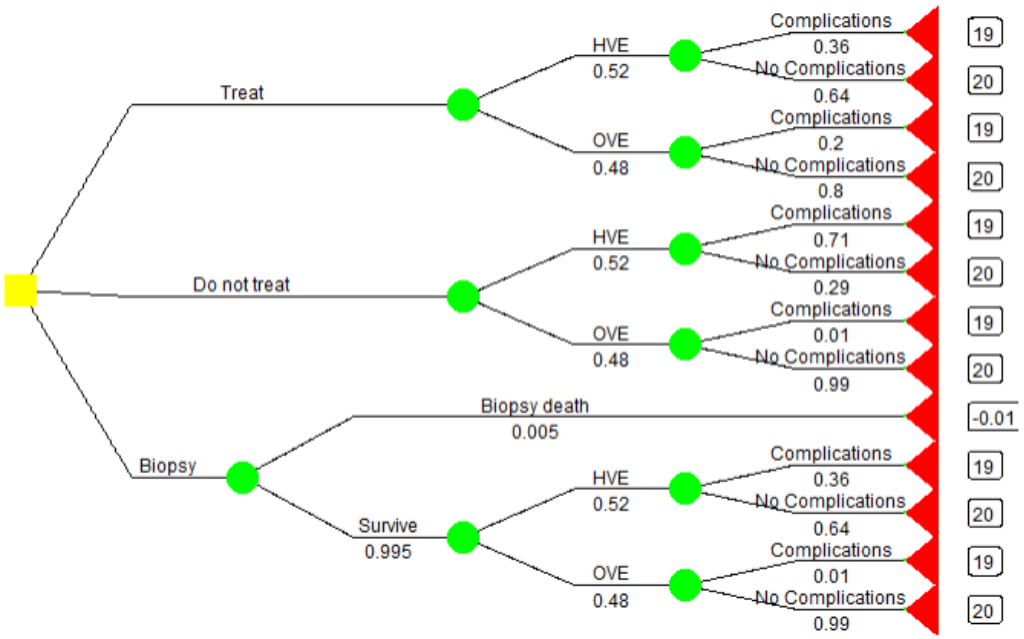

図 4-3-5-1. 決定樹モデルの構造

4.3.6 費用効果分析の実施

効率フロンティアの関係を踏まえて、複数の技術間の相対的な費用対効果を推計する。

```
decision_tree_HVE_cea <- calculate_icers(cost = decision_tree_HVE_output$Cost,
```

```
                                effect     = decision_tree_HVE_output$Effect,
                                strategies = decision_tree_HVE_output$Strategy)
decision_tree_HVE_cea
```

上記スクリプトを実行すると、Console に基本分析の結果が出力される(図 4-3-6-1)。No Tx 群と比較した Tx All 群の ICER は$96,826/QALY と推定された。なお、Biopsy 群は Tx All 群に比して劣位となった。

Description: icers [3 x 7]

Strategy <chr>	Cost <dbl>	Effect <dbl>	Inc_Cost <dbl>	Inc_Effect <dbl>	ICER <dbl>	Status <chr>
No Tx	4117.20	19.62600	NA	NA	NA	ND
Tx All	12908.96	19.71680	8791.76	0.0908	96825.55	ND
Biopsy	32599.41	19.69896	NA	NA	NA	D

3 rows

図 4-3-6-1. 基本分析の結果

4.3.7 効率フロンティア曲線の描画

4.3.6 の出力結果をもとに、効率性フロンティアを描画する。

```
plot(decision_tree_HVE_cea, effect_units = "QALYs", label="all")
```

上記スクリプトを実行すると、Console に図 4-3-7-1 が出力される。

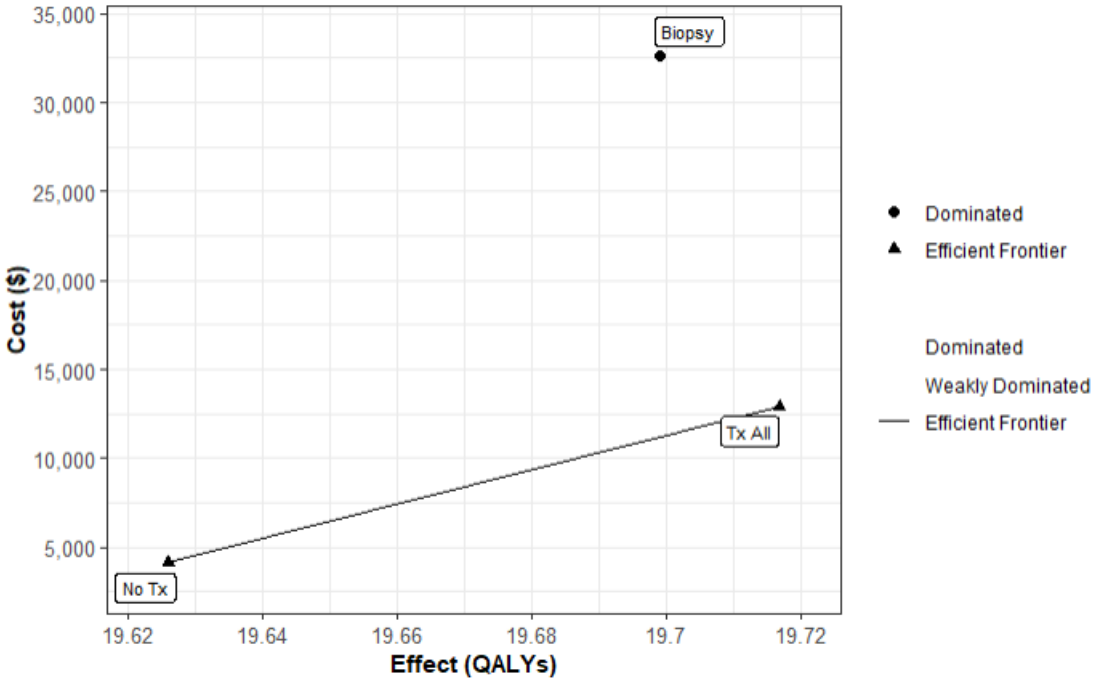

図 4-3-7-1. 効率性フロンティア

5. マルコフモデルに基づく費用効果分析

5.1 演習シナリオ「Sick-Siker モデル」

この演習では、平均年齢 25 歳の個人が罹患し、死亡率の増加、医療費の増加、QOL の低下をもたらす仮想の疾患をモデル化する。この病気には 2 つのレベルがあり、罹患者は最初、病気になるが、その後進行してより進行した病気になる可能性がある。このリサーチクエスチョンの病気に対しては、無治療と治療という 2 つの代替戦略が存在する。治療戦略では、病気あるいはより進行した病気の状態にある患者は、回復するか（病気の場合のみ回復する。進行した病気の患者は回復しない。）、死亡するまで治療される。治療費は、病気あるいはより進行した病気の状態の基本的な医療費に上乗せされる。治療は、病気の患者の QOL を向上させるが、より進行した病気の患者の QOL には影響を与えない。残念ながら、病気の患者と進行した病気の患者を確実に区別することはできないので、病気の人だけに治療の照準を合わせることはできない。あなたは、この治療の費用対効果を評価するよう求められている。

この病気をモデル化するために、Enns らによって最初に記述された Sick-Sicker モデルと呼ばれる状態遷移コホートモデルを用いる。Sick-Sicker モデルは 4 つの健康状態から構成されている。健康状態(H)、2 つの疾患状態、病気(S1)と進行した病気(S2)、そして死者(D)である（図 5-1-1）。すべての個体は健康な状態からスタートする。健康な個体は、時間の経過とともに病気を発症し、S1 に進行する可能性がある。S1 にいる個体は回復（状態 H に戻る）するか、さらに S2 に進行するか、死亡する可能性がある。S2 にいる個体は回復できない (すなわち、S1 にも H にも移行できない)。H の患者は、ベースラインの死亡リスクにさらされ、S1 と S2 の患者は、H の状態の患者と比較して死亡率が増加し、これはハザード比として表される。これらのハザード比は、S1 と S2 にいるときに死亡する確率を計算するために使用される。

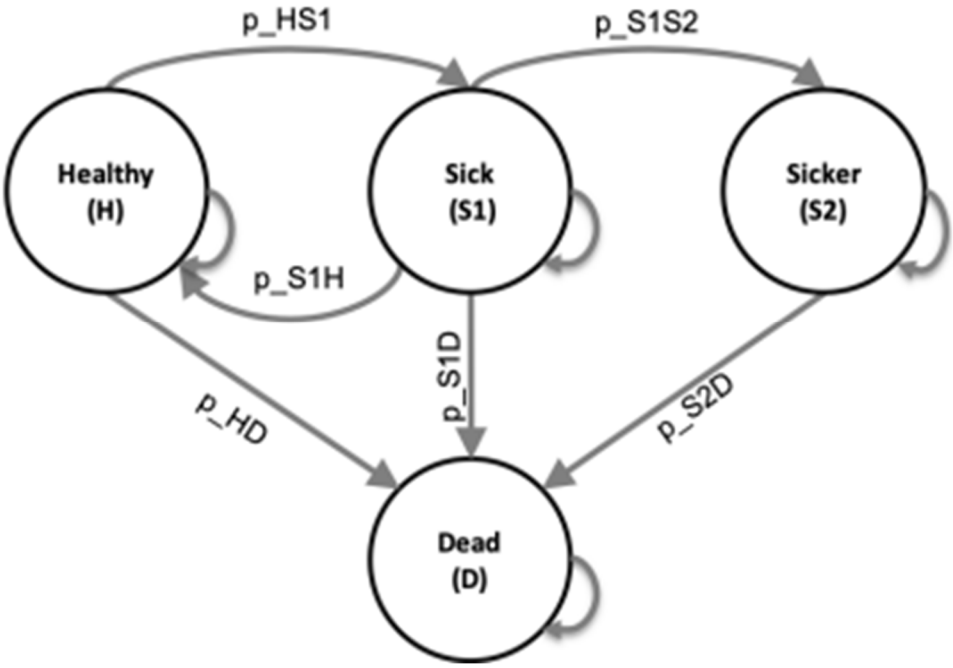

図 5-1-1. Sick-Sicker モデル

パラメータは表 5-1-1 に要約される。本演習では、"markov_sick-sicker_solutions.Rmd "で提供されるコードのテンプレートを使用する。

表 5-1-1. パラメータ設定

Parameter	R name	Value
Time horizon	n_t	30 years
Cycle length		1 year
Names of health states	v_n	H, S1, S2, D
Annual discount rate (costs/QALYs)	d_r	3%
Annual transition probabilities		
- Disease onset (H to S1)	p_HS1	0.15
- Recovery (S1 to H)	p_S1H	0.5
- Disease progression (S1 to S2) in the time-homogeneous model	p_S1S2	0.105
Annual mortality		
- All-cause mortality (H to D)	p_HD	0.005
- Hazard ratio of death in S1 vs H	hr_S1	3
- Hazard ratio of death in S2 vs H	hr_S2	10
Annual costs		
- Healthy individuals	c_H	$2,000
- Sick individuals in S1	c_S1	$4,000
- Sick individuals in S2	c_S2	$15,000
- Dead individuals	c_D	$0
- Additional costs of sick individuals treated in S1 or S2	c_trt	$12,000
Utility weights		
- Healthy individuals	u_H	1.00
- Sick individuals in S1	u_S1	0.75
- Sick individuals in S2	u_S2	0.50
- Dead individuals	u_D	0.00
Intervention effect		
- Utility for treated individuals in S1	u_trt	0.95

*Note: To calculate the probability of dying from S1 and S2, use the hazard ratios provided. To do so, first convert the probability of dying from healthy, p_HD, to a rate; then multiply this rate by the appropriate hazard ratio; finally, convert this rate back to a probability. Recall that you can convert between rates and probabilities using the following formulas: $r = -log(1-p)$ and $p = 1 - e^{(-rt)}$

※DARTH 演習資料より引用

5.2 演習課題の例

1. 無治療群と治療群について R でマルコフモデルを構築する。
2. 無治療群のコホートの生存曲線をプロットする。
3. 治療群と無治療群の費用対効果を推定する。
4. 興味のあるすべての結果を含む費用効果分析の表を作成する。

5.3 スクリプトの例

使用ファイル：markov_sick-sicker_solutions.Rmd

まずワークスペースから変数名等の全設定を削除しておく。

```
rm(list = ls())      # clear memory (removes all the variables from the workspace)
```

5.4 パッケージの読み込み

当該演習に必要となるパッケージを下記のスクリプトにより読み込む。

```
{r, warning = F, message = F}
if (!require('pacman')) install.packages('pacman'); library(pacman) # use this package to conveniently install other packages
# load (install if required) packages from CRAN
p_load("here", "dplyr", "devtools", "scales", "ellipse", "ggplot2", "lazyeval", "igraph", "truncnorm", "ggraph", "reshape2", "knitr", "stringr", "diagram")
# load (install if required) packages from GitHub
# install_github("DARTH-git/dampack", force = TRUE) Uncomment if there is a newer version
# install_github("DARTH-git/dectree", force = TRUE) Uncomment if there is a newer version
p_load_gh("DARTH-git/dampack", "DARTH-git/dectree")
```

5.5 関数の読み込み

特になし

5.6 パラメータの設定

下記のスクリプトにより、必要となるパラメータの定義と入力値を設定する。

```
# ストラテジー名
v_names_str <- c("No Treatment", "Treatment")

# ストラテジーの数
n_str <- length(v_names_str)

# マルコフモデルのパラメータ
年齢 <- 25 # ベースライン時の年齢
max_age <- 55 # フォローアップの最大年齢
n_t <- max_age - age # 時間軸、サイクル数
v_n <- c("H", "S1", "S2", "D") # モデルの 4 つの状態。健康(H)、病気(S1)。
                               # 病人(S2)、死人(D)
n_states <- length(v_n) # 健康状態の数

# 遷移確率(サイクル毎)
p_HD <- 0.005 # 健康なときに死亡する確率
p_HS1 <- 0.15 # 健康なときに病気になる確率
p_S1H <- 0.5 # 病気のときに健康にもどる確率
```

```
p_S1S2 <- 0.105 # 病気になったとき、より進行した病気になる確率
hr_S1 <- 3 # 病人対健康者の死亡のハザード比
hr_S2 <- 10 # 病人対健康者の死亡のハザード比
r_HD <- log(1 - p_HD) # 健康者の死亡率
r_S1D <- hr_S1 * r_HD # 病気の死亡率
r_S2D <- hr_S2 * r_HD # 進行した病気の死亡率
p_S1D <- 1 - exp(-r_S1D) # 病気の死亡確率
p_S2D <- 1 - exp(-r_S2D) # 進行した病気の死亡確率

# コストと効用の入力
c_H <- 2000 # 健康な状態を 1 サイクル維持するためのコスト
c_S1 <- 4000 # 病気の状態で 1 サイクルを過ごすためのコスト
c_S2 <- 15000 # 進行した病気状態で残り 1 サイクルのコスト
c_trt <- 12000 # 治療費(1 サイクルあたり)
c_D <- 0 # 死亡状態のコスト
u_H <- 1 # 健康なときの効用値
u_S1 <- 0.75 # 病気のときの効用値
u_S2 <- 0.5 # 進行した病気の効用値
u_D <- 0 # 死亡時の効用値
u_trt <- 0.95 # 治療を受けているときの効用値

# 割引係数
d_e <- d_c <- 0.03 # コストと QALYs の割引率は 3%で同じ。

# 割引率 d_c に基づいて、各サイクルのコストに対する割引の重みを計算する。
v_dwc <- 1 / (1 + d_e) ^ (0:n_t)
# 割引率 d_e に基づいて、各サイクルにおける有効性の割引重みを計算する。
v_dwe <- 1 / (1 + d_c) ^ (0:n_t)
```

5.7 マルコフモデルの作図

下記のスクリプトでマルコフダイアグラムを作成する。

```
m_P_diag <- matrix(0, nrow = n_states, ncol = n_states, dimnames = list(v_n,
v_n))
m_P_diag["H" , "S1"] = ""
m_P_diag["H" , "D" ] = ""
m_P_diag["H" , "H" ] = ""
```

```
m_P_diag["S1", "H" ] = ""
m_P_diag["S1", "S2"] = ""
m_P_diag["S1", "D" ] = ""
m_P_diag["S1", "S1"] = ""
m_P_diag["S2", "D" ] = ""
m_P_diag["S2", "S2"] = ""
m_P_diag["D", "D"  ] = ""
layout.fig <- c(3, 1)

plotmat(t(m_P_diag), t(layout.fig), self.cex = 0.5, curve = 0, arr.pos = 0.7,
        latex = T, arr.type = "curved", relsize = 0.9, box.prop = 0.8,
        cex = 0.8, box.cex = 0.9, lwd = 1)
```

上記の出力結果を図 5-7-1 に示す。

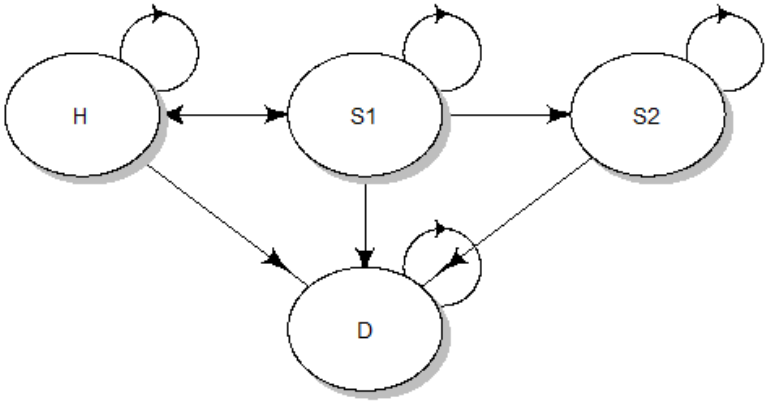

図 5-7-1. マルコフダイアグラム

5.8 行列とベクトルの定義と初期化

5.8.1 コホートトレース

各状態にあるコホートの割合を表すマルコフトレースのための行列を作成する。

```
# 各サイクル
m_M_notrt <- m_M_trt <- matrix(NA,
                               nrow = n_t + 1, ncol = n_states,
                               dimnames = list(paste("cycle", 0:n_t, sep = " "), v_n))
```

```
head(m_M_notrt) # 行列の最初の 6 行を表示する。

# コホートは健康な状態から始まる
m_M_notrt[1, ] <- m_M_trt[1, ] <- c(1, 0, 0, 0) # コホートトレースの最初のサイクルを開始させる。
```

5.8.2 遷移確率行列

無治療群に対する遷移確率行列を作成する。

```
m_P_notrt <- matrix(0,
                    nrow = n_states,
                    ncol = n_states,
                    dimnames = list(v_n, v_n)) # 行列の列と行に名前を付ける
m_P_notrt
```

遷移確率の行列に変数を代入する。

```
# 健康な状態からの移行
m_P_notrt["H", "H" ] <- 1 - (p_HS1 + p_HD)
m_P_notrt["H", "S1" ] <- p_HS1
m_P_notrt["H", "D" ] <- p_HD
# 病気の状態からの移行
m_P_notrt["S1", "H" ] <- p_S1H
m_P_notrt["S1", "S1"] <- 1 - (p_S1H + p_S1S2 + p_S1D)
m_P_notrt["S1", "S2"]の場合 <- p_S1S2
m_P_notrt["S1", "D" ] <- p_S1D
# 進行した病気からの移行
m_P_notrt["S2", "S2"] <- 1 - p_S2D
m_P_notrt["S2", "D" ] <- p_S2D
# from Dead
m_P_notrt["D", "D" ] <- 1

# 行の合計が 1 であることを確認する
rowSums(m_P_notrt)

# 無治療と同じ治療の遷移確率行列を作成する
m_P_trt <- m_P_notrt
```

5.9 マルコフモデルの実行

```
for (t in 1:n_t){ # サイクル数だけループさせる
  m_M_notrt[t + 1, ] <- t(m_M_notrt[t, ]) %*% m_P_notrt #マルコフトレースを推定します。
                                                  # 次のサイクル(t + 1)のために
  m_M_trt[t + 1, ] <- t(m_M_trt[t, ]) %*% m_P_trt # マルコフトレースを推定する。
                                                  # 次のサイクル(t + 1)のために
} # ループを閉じる

head(m_M_notrt) # 行列の最初の 6 行を表示する。
```

上記のスクリプトを実行すると、Console に下記が出力される。

```
              H       S1        S2         D
cycle 0 1.0000000 0.0000000 0.00000000 0.00000000
cycle 1 0.8450000 0.1500000 0.00000000 0.00500000
cycle 2 0.7890250 0.1837612 0.01575000 0.01146377
cycle 3 0.7586067 0.1881968 0.03427491 0.01892157
cycle 4 0.7351211 0.1853199 0.05235988 0.02719916
cycle 5 0.7138373 0.1807036 0.06925860 0.03620055
```

5.10 アウトカムの計算と表示

5.10.1 コホートトレース

```
# データのプロットを作成する
matplot(m_M_notrt, type = 'l',
        ylab = "Probability of state occupancy",
        xlab = "Cycle",
        main = "Cohort Trace")
# グラフに凡例を追加する
legend("topright", v_n, col = 1:n_states, lty = 1:n_states, bty = "n")
```

上記のスクリプトを実行すると、Console にコホートトレースが出力される(図 5-10-1)。

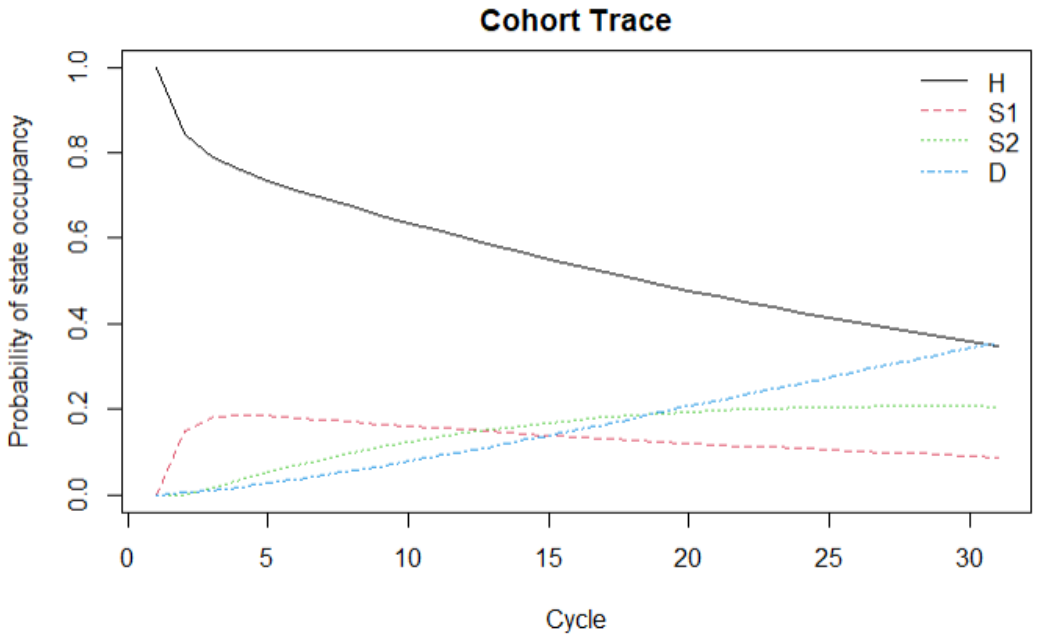

図 5-10-1-1. コホートトレース

5.10.2 全生存

```
# 無治療の場合の全生存（OS）確率を計算する
v_os_notrt <- 1 - m_M_notrt[, "D"].
# OS 確率の代替計算方法
v_os_notrt <- rowSums(m_M_notrt[, 1:3])
# OS を示す簡単なプロットを作成する
plot(0:n_t, v_os_notrt, type = 'l',
     ylim = c(0, 1),
     ylab = "Survival probability",
     xlab = "Cycle",
     main = "Overall Survival")
# グリッドを追加する
grid(nx = n_t, ny = 10, col = "lightgray", lty = "dotted", lwd = par("lwd")).Grid(nx = n_t, ny = 10, col = "lightgray", lty = "dotted", lwd = "lwd"),
     equilogs = TRUE)
```

上記のスクリプトを実行すると、Console に生存曲線が出力される(図 5-10-2-1)。

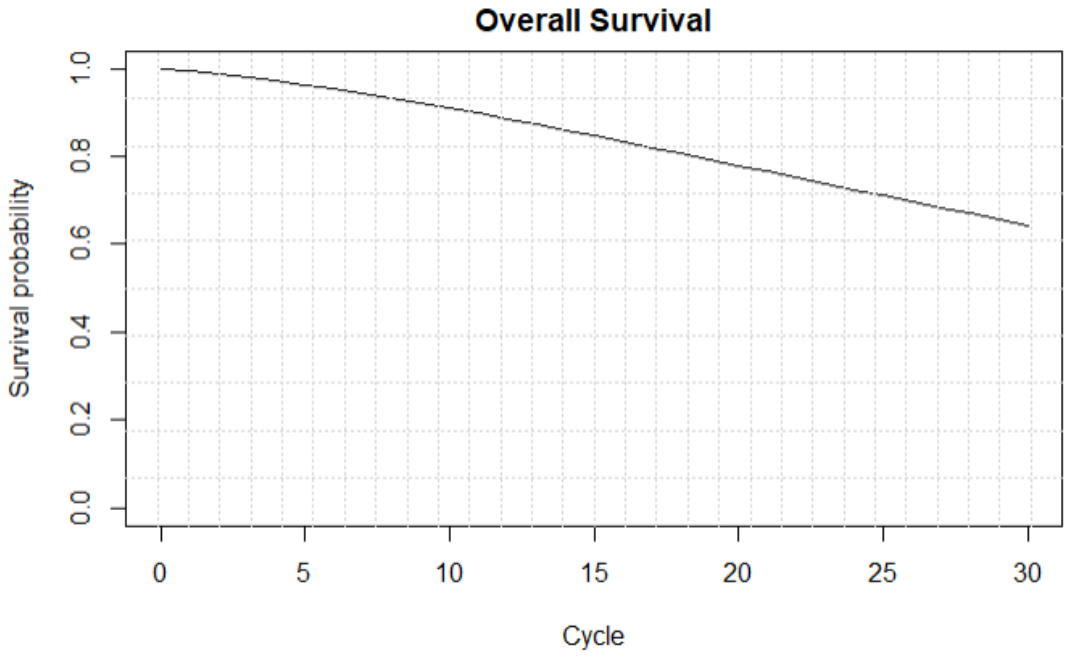

図 5-10-2-1. 生存曲線

5.10.3 期待生存年

```
v_le <- sum(v_os_notrt) # OS の確率の経年変化（つまり寿命）を合計したもの
```

期待生存年は 26 年と推計された。

5.10.4 有病割合

```
v_prev <- rowSums(m_M_notrt[, c("S1", "S2")]) / v_os_notrt
plot(v_prev,
     ylim = c(0, 1),
     ylab = "Prevalence",
     xlab = "Cycle",
     main = "Disease prevalence")
```

上記のスクリプトを実行すると、Console に有病割合の経時的変化が出力される(図 5-10-4-1)。

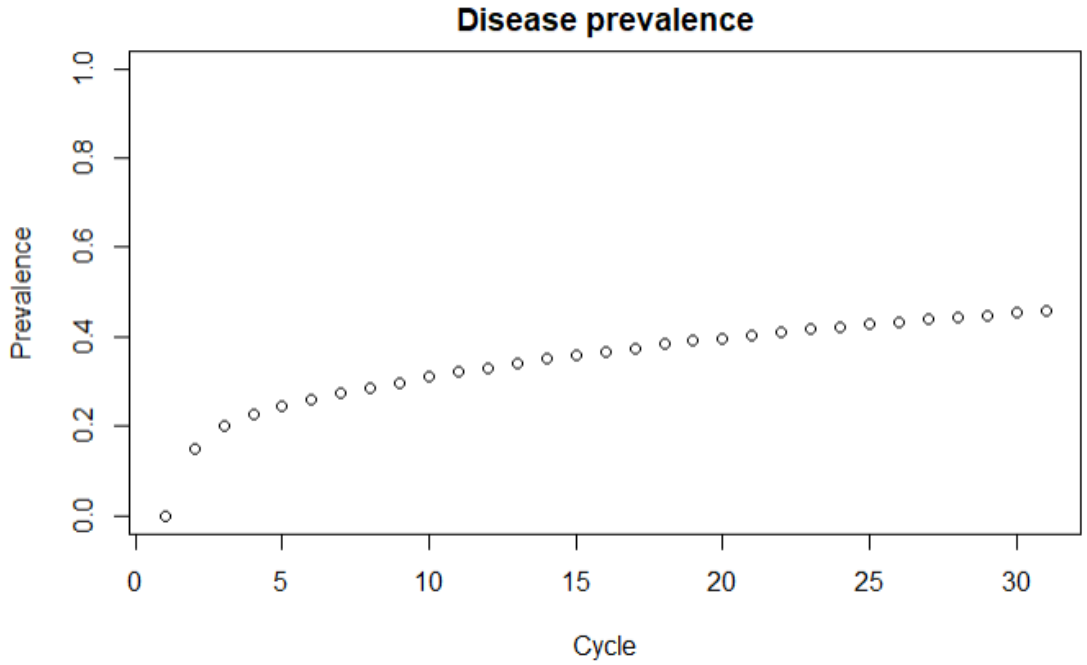

図 5-10-4-1. 有病割合の変化

5.10.5 S1 状態の割合

```
v_prop_S1 <- m_M_notrt[, "S1"] / v_prev
plot(0:n_t, v_prop_S1,
     xlab = "Cycle",
     ylab = "Proportion",
     main = "Proportion of sick in S1 state",
     col  = "black", type = "l")
```

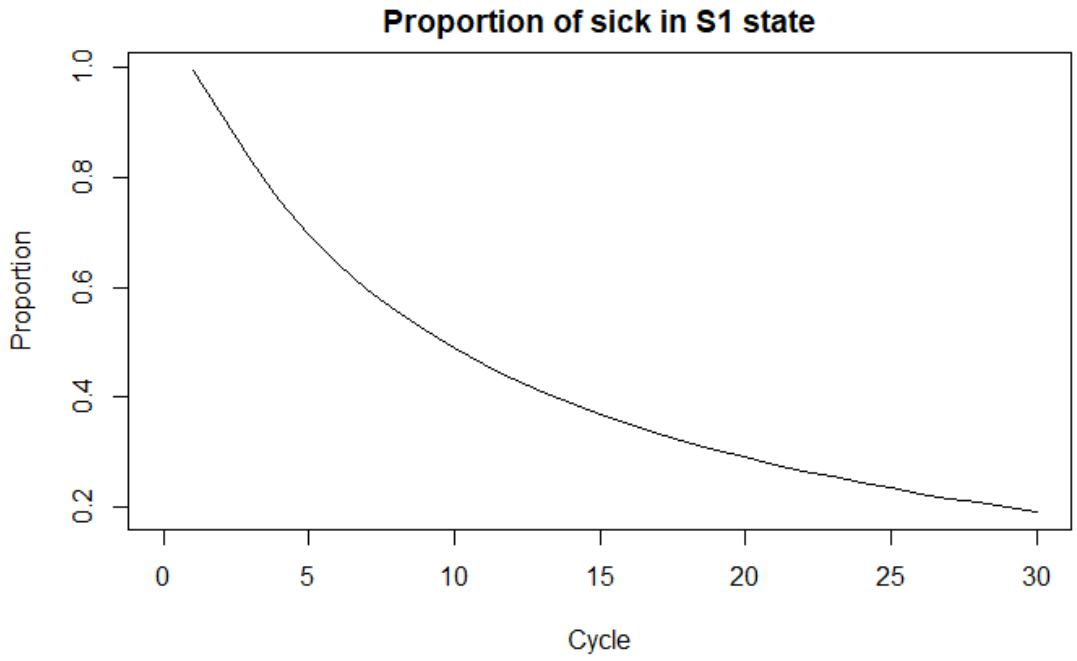

図 5-10-5-1. S1 状態の割合の変化

5.11 費用対効果の結果

以降のスクリプトで費用効果分析の結果を集計することができる。

```
# 処置別のコストと効用を持つベクトル
v_u_notrt <- c(u_H, u_S1, u_S2, u_D)
v_u_trt <- c(u_H, u_trt, u_S2, u_D)

v_c_notrt <- c(c_H, c_S1, c_S2, c_D)
v_c_trt <- c(c_H, c_S1 + c_trt, c_S2 + c_trt, c_D)
```

```
#各群の費用と QALY の平均
v_tu_notrt  <- m_M_notrt   %*%  v_u_notrt
v_tu_trt    <- m_M_trt     %*%  v_u_trt

v_tc_notrt  <- m_M_notrt   %*%  v_c_notrt
v_tc_trt    <- m_M_trt     %*%  v_c_trt
```

```
#各群の費用と QALY の平均（割引）
tu_d_notrt <- t(v_tu_notrt) %*% v_dwe
```

```
tu_d_trt <- t(v_tu_trt) %*% v_dwe

tc_d_notrt <- t(v_tc_notrt) %*% v_dwc
tc_d_trt <- t(v_tc_trt) %*% v_dwc

# ベクターに格納する
v_tc_d <- c(tc_d_notrt, tc_d_trt)
v_tu_d <- c(tu_d_notrt, tu_d_trt)

# コストと効果を割り引いたデータフレーム
df_ce <- data.frame(Strategy = v_names_str,
                    Cost     = v_tc_d,
                    Effect   = v_tu_d)
df_ce
```

上記スクリプトを実行すると、以下の結果が出力される。

Strategy	Cost	Effect
No Treatment	75976.15	15.83885
Treatment	141623.03	16.40041

```
#ICER の計算
df_cea <- calculate_icers(cost       = df_ce$Cost,
                    effect     = df_ce$Effect,
                    strategies = df_ce$Strategy)
df_cea
```

上記スクリプトを実行すると、以下の結果が出力される。ICER は$116,901/QALY と推計された。

Strategy	Cost	Effect	Inc_Cost	Inc_Effect	ICER	Status
No Treatment	75976.15	15.83885	NA	NA	NA	ND
Treatment	141623	16.40041	65646.88	0.561558	116901.4	ND

```
#効率フロンティアの描画
plot(df_cea, effect_units = "Quality of Life", xlim = c(15.6, 16.6))
```

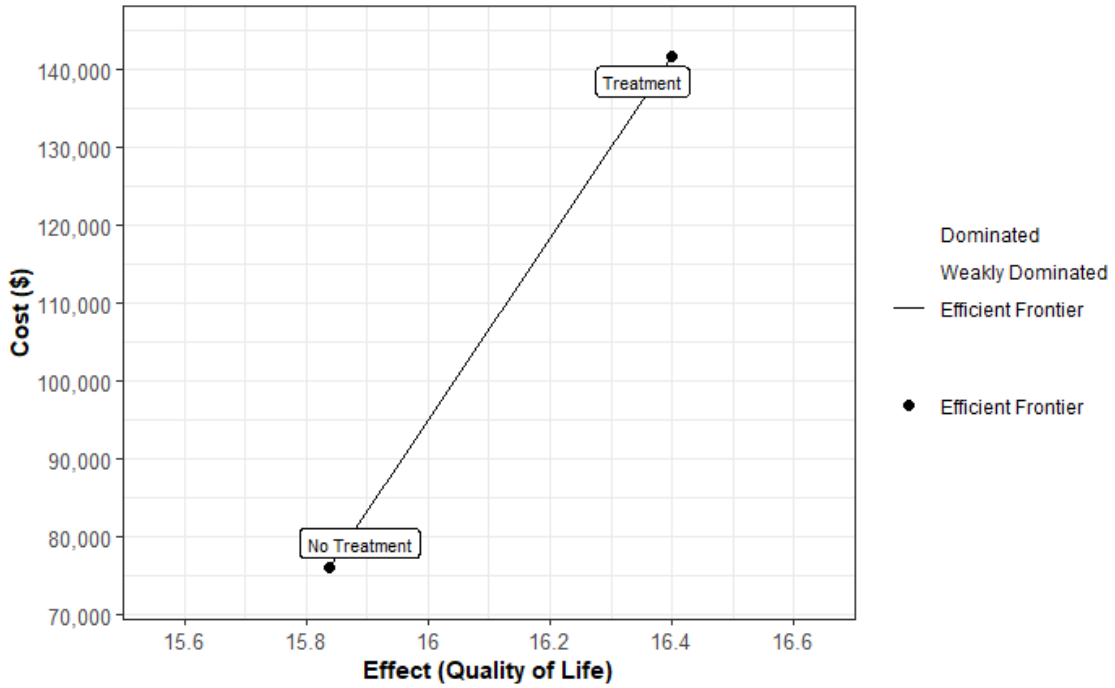

図 5-11-1. 費用効果平面図

6. PartSA 等による生存アウトカムの推計

進行・再発がんの費用効果分析において頻用される Partitioned survival analysis (PartSA) model 等について、R での実装スクリプト(”survival_3-state_solutions.Rmd”)を紹介する。

6.1 メモリの消去

使用ファイル：survival_3-state_solutions.Rmd

まずワークスペースから変数名等の全設定を削除しておく。

```
rm(list = ls())      # clear memory (removes all the variables from the workspace)
```

6.2 必要なパッケージの読み込み

```
if (!require('pacman')) install.packages('pacman'); library(pacman) # use this package to conveniently install other packages
# load (install if required) packages from CRAN
p_load("here", "dplyr", "devtools", "gems", "flexsurv", "survminer", "survHE", "ggplot2", "msm", "igraph", "mstate",  "reshape2", "knitr", "diagram")
```

6.3 必要な関数の読み込み

教材スクリプトに同封されている"survival_functions.R"を作業中のフォルダに保存し

ておく必要がある。下記のスクリプトにより、必要な関数を読み込む。

```
source("survival_functions.R")
```

6.4 パラメータの設定

分析の基本のパラメータを定義する。

```
v_n <- c("healthy", "sick", "dead") # 状態名
n_s <- length(v_n) # 状態の数
n_i <- 5000 # シミュレーション数
c_l <- 1 / 12 # サイクルの長さ (1 ヶ月)
n_t <- 30 # 年数 (20 年)
times <- seq(0, n_t, c_l) # 年単位での周期
set.seed(2020) # シードを設定する
```

次に、すべての遷移を表示し、番号を付けた遷移確率行列を作成する。

```
tmat <- matrix(NA, n_s, n_s, dimnames = list(v_n,v_n))
tmat["healthy", "sick"] <- 1
tmat["healthy", "dead"] <- 2
tmat["sick" , "dead"] <- 3
```

PartSA モデルの構造を図示する。出力結果は図 6-4-1 に示す。

```
layout.fig <- c(2,1)
plotmat(t(tmat), t(layout.fig), self.cex = 0.5, curve = 0, arr.pos = 0.76,
        latex = T, arr.type = "curved", relsize = 0.85, box.prop=0.8,
        cex = 0.1, box.cex = 0.7, lwd = 1)
```

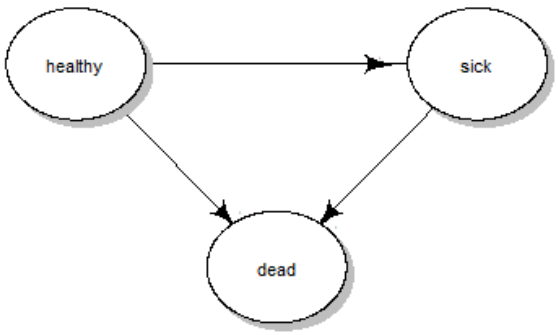

図 6-4-1. PartSA モデルの構造

6.5 生存曲線の描画

まず、全生存期間(OS)と無増悪生存期間(PFS)に関する模擬データを生成する。この例では、550 例の OS と PFS に関する時間(最長 60 年)と打ち切りの有無のデータが生成される。

```
source("data.R")
head(true_data)
head(sim_data)
head(status)
head(OS_PFS_data)
```

生存曲線を描画する。

```
fit_KM    <- survfit(Surv(time = OS_time, event = OS_status) ~ 1, data = OS_PFS_data,
                     type ="fleming-harrington")
plot(fit_KM, mark.time = T)
```

上記スクリプトを実行すると、OS のカプランマイヤー生存曲線が描画される(図 6-5-1)。

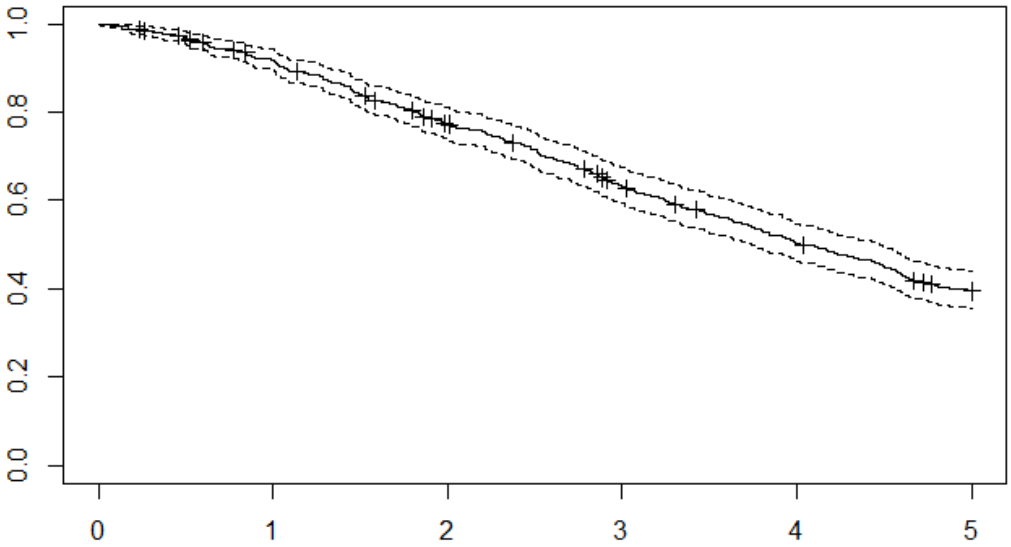

図 6-5-1. カプランマイヤー曲線

なお、以下のスクリプトにより、より美しい作図が可能である(図 6-5-2)。

```
ggsurvplot(
  fit_KM,
  data = OS_PFS_data,
  size = 1, # 線の大きさを変更する
  palette = c("orange2"), # カスタムカラーパレット
  conf.int = TRUE, # 信頼区間を追加する
  pval = TRUE, # p 値を追加する
  risk.table = TRUE, # リスクテーブルを追加する.
  risk.table.height = 0.25, # 複数のグループがある場合に変更すると便利。
  ggtheme = theme_bw(), # ggplot2 のテーマを変更する。
  xlab = 'Time in days',  # X 軸のラベルを変更する
 title    = "Survival curve for Progression-Free Survival (PFS)",
  subtitle = "Based on Kaplan-Meier estimates"
)
```

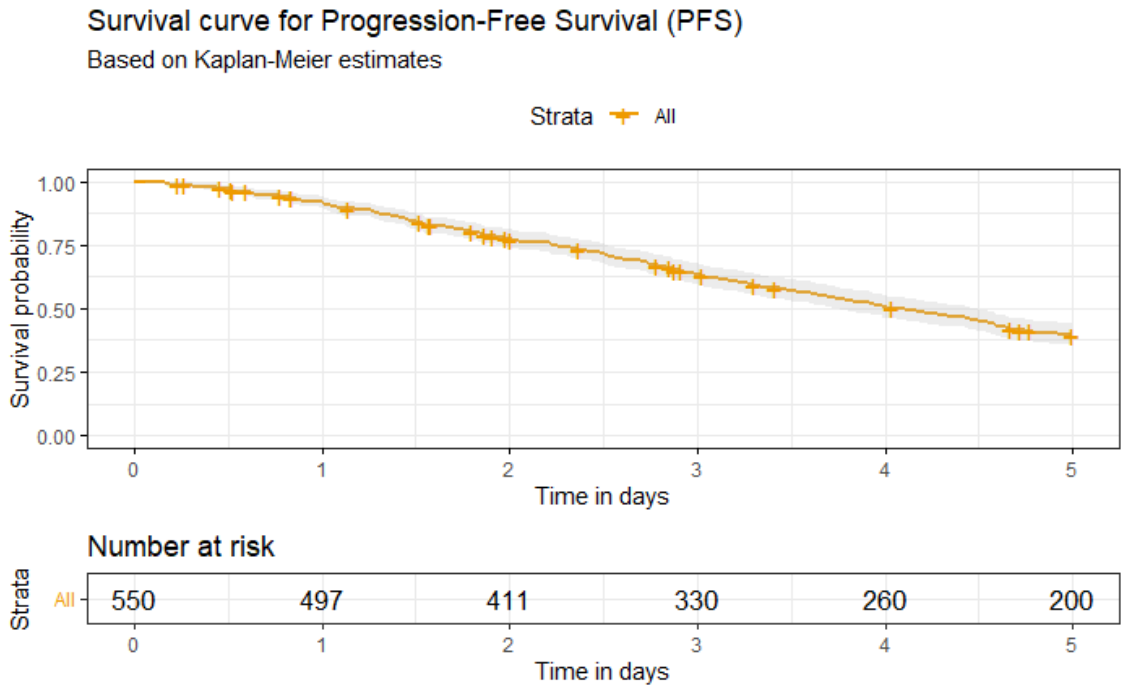

図 6-5-2. カプランマイヤー曲線

6.6 ParSA モデル

flexsurv 関数により Time to event データに任意のパラメトリック関数をあてはめることができる。以下は OS に対するワイブル関数のあてはめの例を示す(図 6-6-1)。

```
fit_weib <- flexsurvreg(Surv(time = OS_time, event = OS_status) ~ 1, data =
```

```
OS_PFS_data,  dist = "weibull")
plot(fit_weib)
```

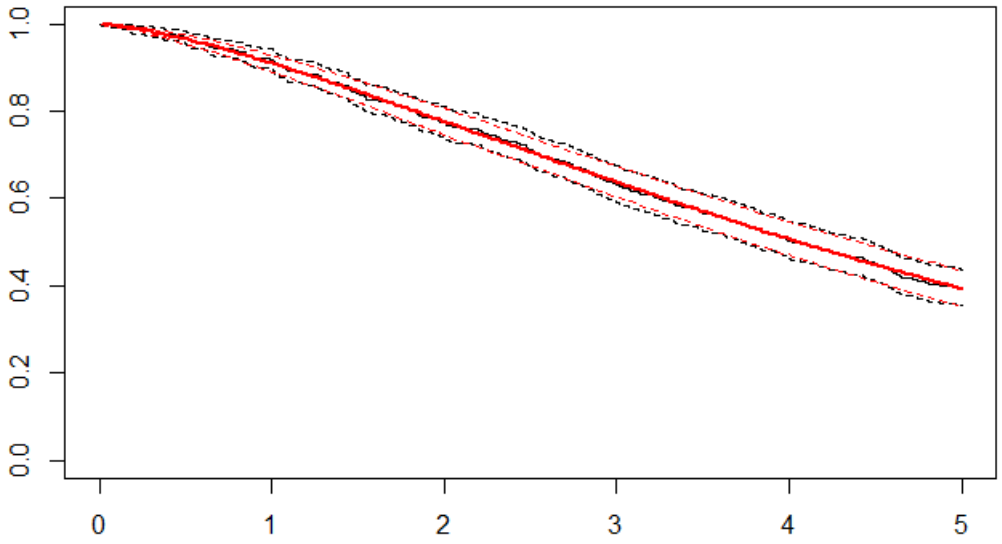

図 6-6-1. OS に対するワイブル関数のあてはめ

一般に PartSA では、OS、PFS のデータに対して標準的に用いられるパラメトリックモデルをあてはめ、AIC/BIC を抽出し、最も適合度の高いものをベストカーブの候補として選択する。すべてのカーブを重ね合わせた結果は図 6-6-2, 6-6-3 のように出力される。

```
fit_PFS    <-  fit.fun(time    =  "PFS_time",  status  =  "PFS_status",  data  =
OS_PFS_data,
                times = times, extrapolate = T)
fit_OS     <-  fit.fun(time    =  "OS_time",  status   =  "OS_status"  ,  data  =
OS_PFS_data,
                times = times, extrapolate = T)

best_PFS <- fit_PFS[["Weibull"]]
best_OS  <- fit_OS [["Weibull"]]
```

あてはめたカーブをもとに PartSA モデルを構築する。ここではワイブル関数をあてはめた場合の経時的なメンバーシップの変化が m_M_PSM に格納される。

```
m_M_PSM <- partsurv(best_PFS, best_OS, time = times)$trace
```

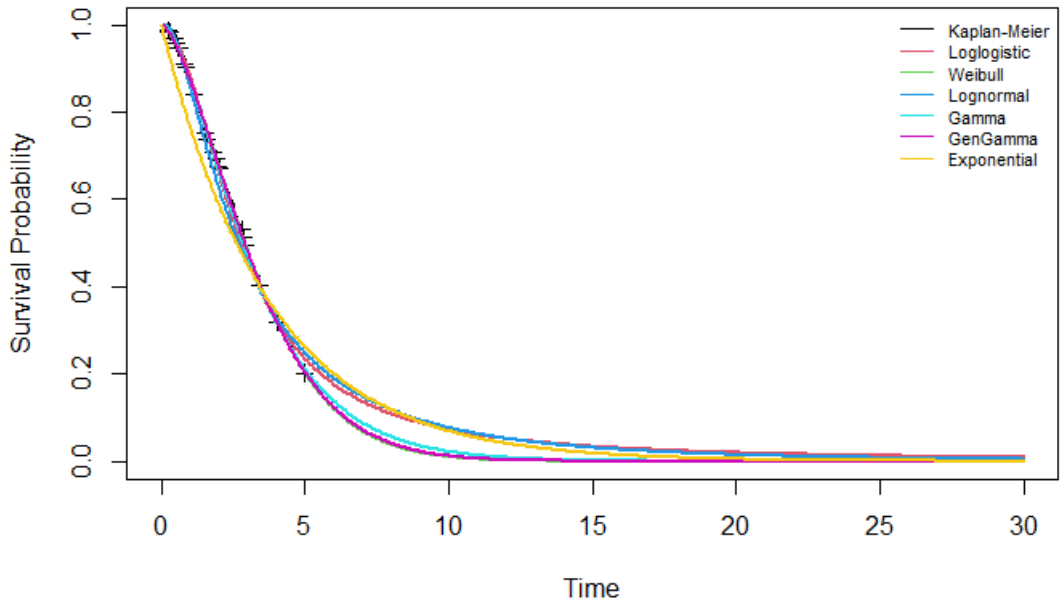

図 6-6-2. PFS の外挿結果

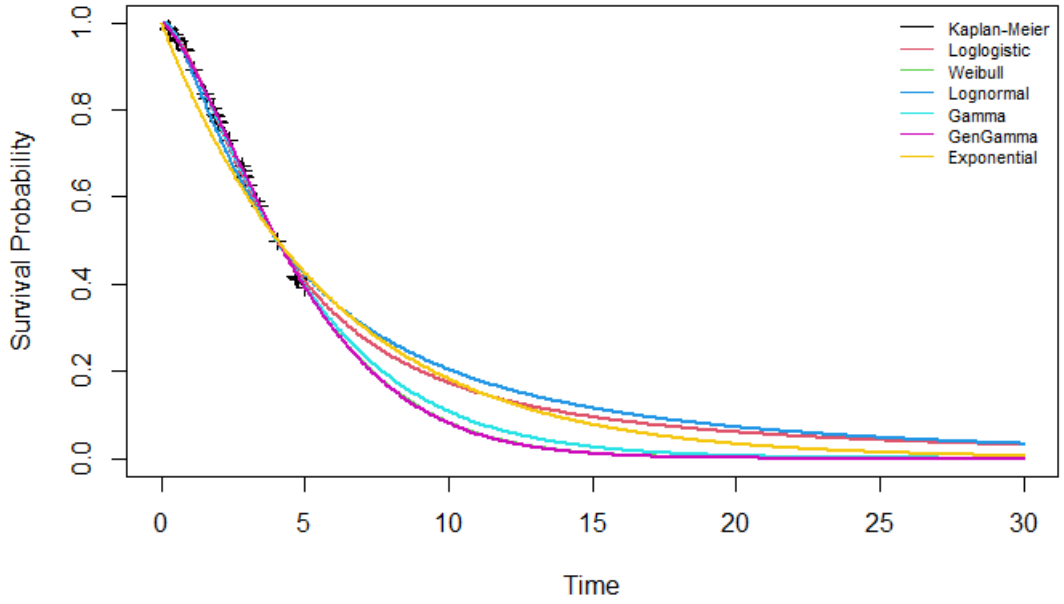

図 6-6-3. OS の外挿結果

6.7 多状態モデル(1)

無増悪、増悪、死亡の 3 状態よりも多い健康状態に一般化可能な多状態モデルの実装法

について紹介する。なお、flexsurv 関数は、ワイド形式ではなくロング形式のデータを必要とするため、mstate パッケージを使用してデータを変換する。

```
data_long <- msprep(time = sim_data, status = status, trans = tmat )
data_long$trans <- as.factor(data_long$trans) # trans を因子に変換する。

data_long$from  <- case_when(data_long$from == 1 ~ "healthy",
                        data_long$from == 2 ~ "sick",
                        data_long$from == 3 ~ "dead")

data_long$to    <- case_when(data_long$to == 1 ~ "healthy",
                        data_long$to == 2 ~ "sick",
                        data_long$to == 3 ~ "dead")
```

全てのパラメトリック多状態モデルを同時にデータにフィットさせ、AIC/BIC を抽出する。通常、AIC が最も低いものをベストカーブとして選択する。ここではログロジスティック関数を選択した場合が例示されている。スクリプトの出力結果を図 6-7-1 に示す。

```
fits <- fit.mstate(time ="time", status = "status", trans, data = data_long,
                   times = times, extrapolate = T )
best.fit <- fits[["Loglogistic"]]
```

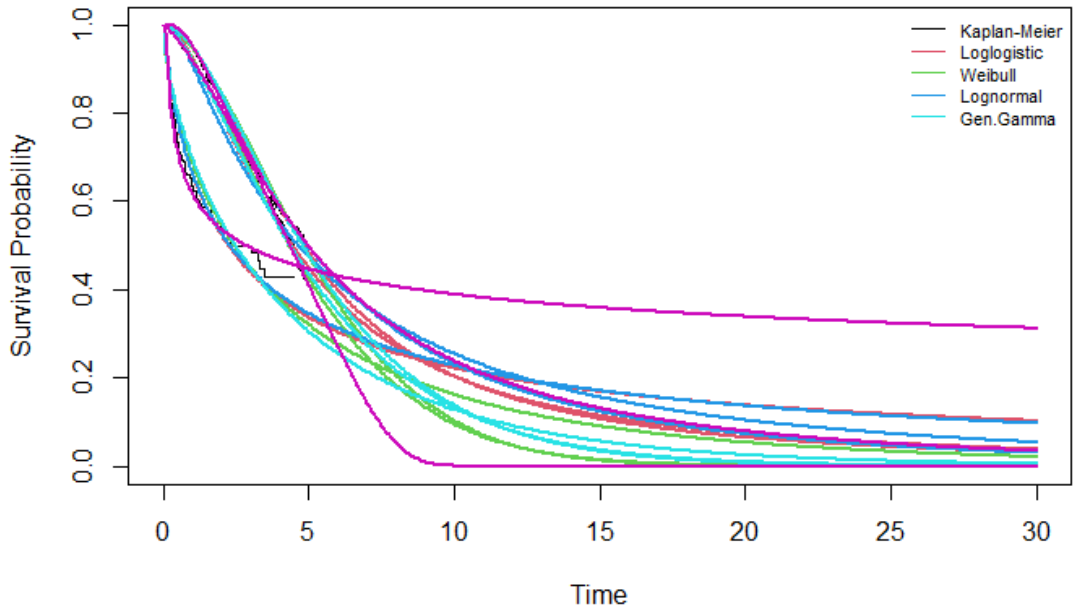

図 6-7-1. パラメトリック関数のあてはめ

先にフィットした多状態モデルから離散イベントシミュレーション(DES: Discrete Event Simulation)モデルを構築する。ここではログロジスティック関数を用いた場合が例示されている。スクリプトを実行すると、DES モデルに基づく経時的なメンバーシップの変化が m_M_DES に格納される。

```
DES_data <- sim.fmsm(best.fit, start = 1, t = n_years, trans = tmat, M = n_i)
m_M_DES  <- trace.DES(DES_data, n_i  = n_i , times = times, tmat = tmat)
```

6.8 多状態モデル(2)

多状態モデルは、各遷移に対して独立したパラメトリック関数をあてはめることができる。

まず、各遷移のサブセットを作成する

```
data_HS <- subset(data_long, trans == 1)
data_HD <- subset(data_long, trans == 2)
data_SD <- subset(data_long, trans == 3)
```

次に、各遷移に対して独立したモデルをあてはめ、最も低い AIC を持つものを選択する。

```
fit_HS <- fit.fun(time = "time", status = "status", data = data_HS, times = times,
                  extrapolate = T)
```

```
fit_HD <- fit.fun(time = "time", status = "status", data = data_HD, times = times,
                  extrapolate = T)
fit_SD <- fit.fun(time = "time", status = "status", data = data_SD, times = times,
                  extrapolate = T)

best.fit_HS <- fit_HS[["Gamma"]]
best.fit_HD <- fit_HD[["Weibull"]]
best.fit_SD <- fit_SD[["Lognormal"]]
```

上記スクリプトを実行すると、Console に出力が表示される(図 6-8-1~6-8-3)。

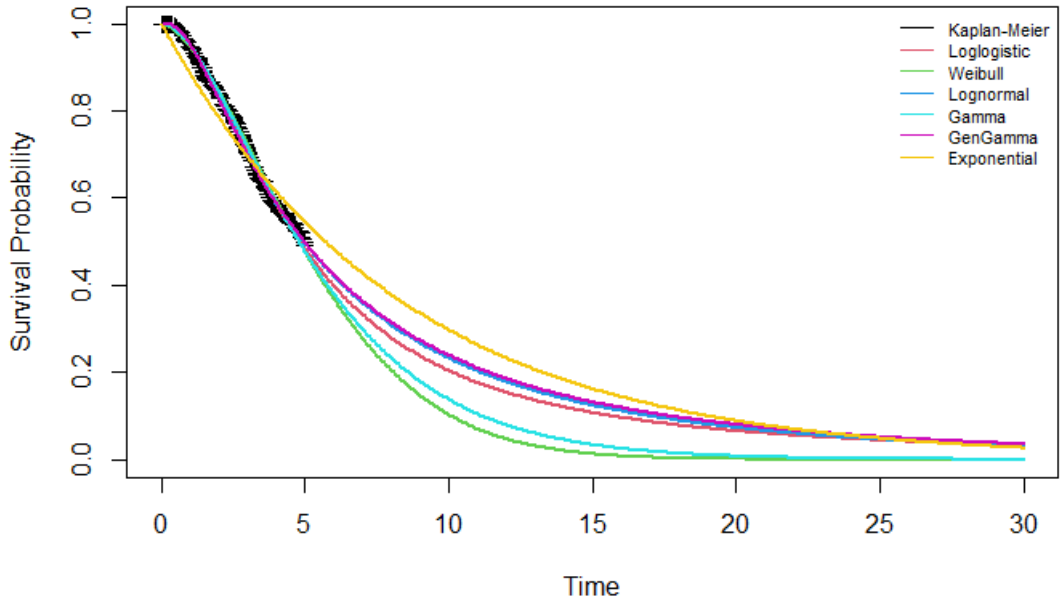

図 6-8-1. 健康からの死亡確率にガンマ関数をあてはめた結果

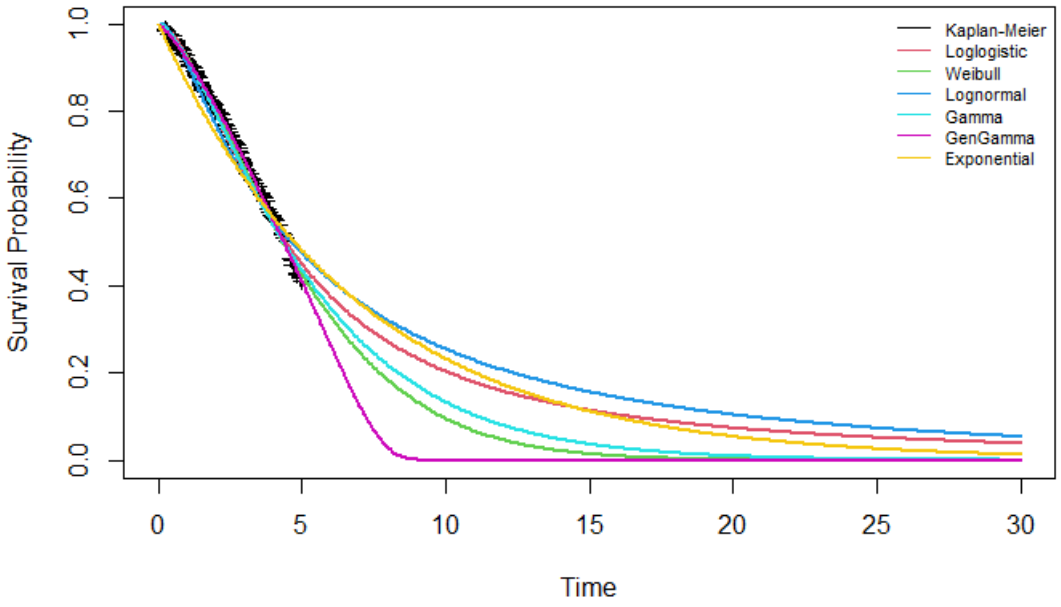

図 6-8-2. 健康からの罹患確率にワイブル関数をあてはめた結果

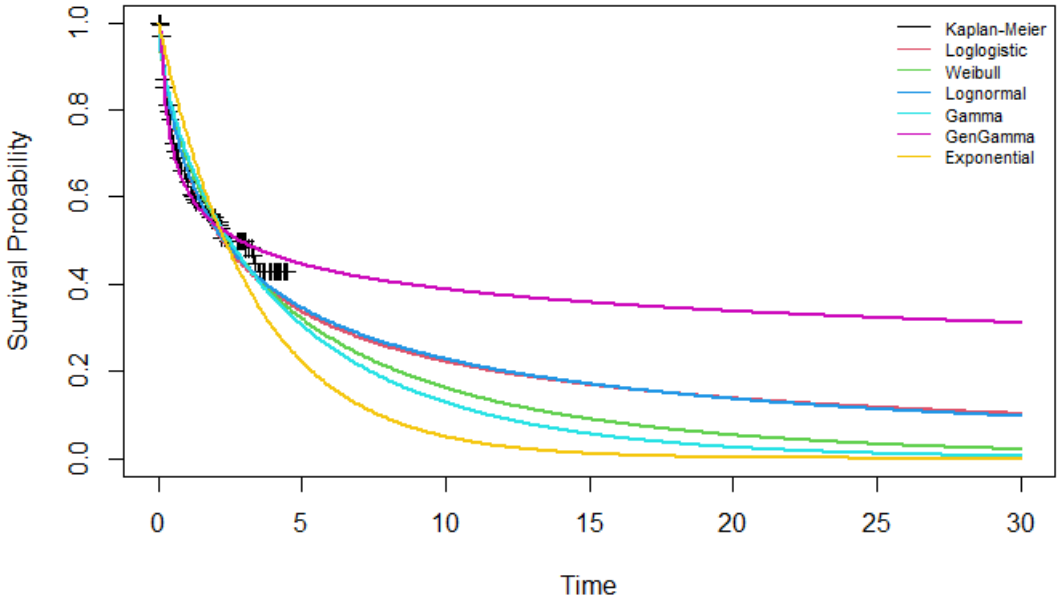

図 6-8-. 疾患からの死亡確率に対数正規関数をあてはめた結果

なお、DES の代替法にマイクロシミュレーションを適用することができる。マイクロシミュレーションを用いた場合、計算量が多くなるが、モデリングの自由度は高くなるメリ

ットがある。マイクロシミュレーションを実行するためには、単位時間当たりの遷移確率が必要である。

以下では、まず、最適なパラメトリック関数から遷移確率を抽出する。

```
p_HS <- flexsurvreg_prob(object = best.fit_HS, times = times)
p_HD <- flexsurvreg_prob(object = best.fit_HD, times = times)
p_SD <- flexsurvreg_prob(object = best.fit_SD, times = times)
```

仮想コホート患者は全員が "健康" 状態からスタートすると仮定する。

```
v_M_init  <- rep("healthy", times = n_i)
v_Ts_init <- rep(0, n_i) # モデル開始時に病気であった時間を表すベクトル
```

以下は、1 サイクルごとの遷移確率を生成する関数である。

```
Probs <- function(M_t, v_Ts, t){.
  # 引数
    # M_t: t サイクル目の健康状態(文字型変数)
    # v_Ts: 病気である期間を表すベクトル
    # t: 現在のサイクル
  # 返り値
    # そのサイクルの遷移確率

  # 状態遷移確率の行列を作成する
  m_p_t <- matrix(0, nrow = n_s, ncol = n_i)
  # 状態の名前を行に付与する
  rownames(m_p_t) <- v_n

  # m_p_t を適切な確率で更新する。
  # 健常時の遷移確率
   m_p_t[, M_t == "healthy"] <- rbind(1 - p_HD[t] - p_HS[t], p_HS[t], p_HD[t])
  # 病気のときの遷移確率
  m_p_t[, M_t == "sick"]    <- rbind(0, 1 - p_SD[v_Ts], p_SD[v_Ts])
  # 死亡時の遷移確率
  m_p_t[, M_t == "dead"]    <- rbind(0, 0, 1)
  return(t(m_p_t))
}
```

以下のスクリプトで、マイクロシミュレーションを実行する。外挿の結果は、m_M_Micro に格納される。

```
MicroSim <- function(n_i, seed = 1) {...
  # 引数
  # n_i: 個体数
  # seed: デフォルトは 1

  set.seed(seed) # シードを設定する

  # m_M は、各個体の時間経過に伴う健康状態情報を格納するために使用される
  times <- seq(0, n_t, c_l) # 年単位で見たサイクル

  m_M <- matrix(nrow = n_i, ncol = length(times) ,
                dimnames = list(paste("ind" , 1:n_i, sep = " "),
                                paste("year", times, sep = " ")))

  m_M[, 1] <- v_M_init # 個人 i の健康状態の初期値
  v_Ts <- v_Ts_init # 個人 i の発病からの経過時間を初期化する。

  # 1 から n_t までの周期でループを回す
  for (t in 1:(length(times)-1)) { { { { { { (t in 1:(length(times)-1))
    # 健康状態 t に基づき、サイクルの遷移確率を計算する。
    m_p <- Probs(m_M[, t], v_Ts, t)
    # 現在の健康状態をサンプリングし, その状態を行列 m_M に格納します.
    m_M[, t + 1] <- samplev(m_p, 1)

    # 発病からの時間を t + 1 だけ更新する
    v_Ts <- ifelse(m_M[, t + 1] == "sick", v_Ts + 1, 0)

    # シミュレーションの進捗状況を表示する
    if(t %in% seq(1,(length(times)),10)) { # 10%ごとに進捗を表示する
      cat('\r', paste(round(t/length(times)*100,0), "% done", sep = ""))
    } else if (t == (length(times)-1)) {cat('\r', paste("100% done"))}。

   # タイムポイントに対するループを閉じる
```

```
  # シミュレーションの結果をリストに格納する。
  results <- list(m_M = m_M)

  return(results) # 結果を返す

} # MicroSim 関数終了

# シミュレーションモデルの実行
Micro_data <- MicroSim(n_i, seed = 1)

# マイクロシミュレーショントレースの作成
m_M_Micro <- t(apply(Micro_data$m_M, 2, function(x) table(factor(x, levels =
v_n,
                                                    ordered = TRUE))
m_M_Micro <- m_M_Micro / n_i # 個体数の比率を計算する
colnames(m_M_Micro) <- v_n
rownames(m_M_Micro) <- paste("Cycle", times, sep = " ")
```

6.9 複数の外挿法の比較

以下では、これまでの全ての方法を比較する。

```
# 実データに対するトレースを計算する
m_M_data <- transitionProbabilities(generate$cohort, times =
times)@probabilities

# 視覚的にすべてのメソッドを比較する

matplot(times,m_M_data, type='l', lty = 1, col = 1, ylab= "proportion of cohort",
xlab = "Time",
        main = "Trace comparisons",xlim=c(0,25))
matlines(times, m_M_DES,   col = 2, lty = 1)
matlines(times, m_M_Micro, col = 3, lty = 1)
matlines(times, m_M_PSM,   col = 4, lty = 1)
legend("right", c("True Data", "DES","Microsim",  "PSM"),
        col = 1:4, lty = rep(1,4), bty= "n")
```

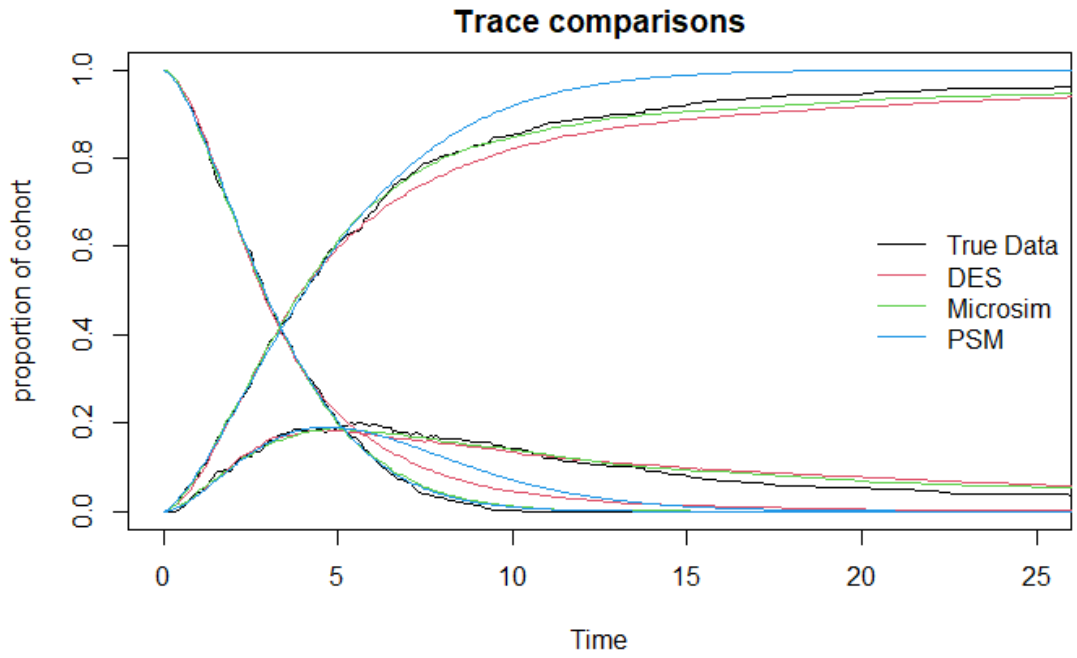

図 6-9-1. 全ての手法の比較

7. さいごに

近年、医療経済評価のモデリングにおいて R の活用を推奨する動きがみられる。複数の研究グループが R によるモデル構築のスクリプトやパッケージを開発、公開するとともに、ワークショップなどの教育活動が展開されている。現時点では医療技術評価のプロセスにおいて R ベースのモデルが取り扱われた事例は確認されていない。しかしながら、R ベースのモデルは従来のエクセルベースに比して、計算効率や透明性の点で優れていることから、将来的に R の利用が進むことも考えられる。今後、日本の費用対効果評価に関する教育・研究においても、DARTH や R in HTA が提供するスクリプトをベースに、実践的な教育機会を提供することが有用と考えられた。