モンティ・ホール問題または三囚人問題の拡張とその確率操作シミュレーション

JRF

(http://jrf.cocolog-nifty.com/software/2016/08/post.html)

初版：2016 年 8 月 27 日
第 2.1 版：2019 年 1 月 19 日

概要

近年、J. ローゼンハウスの『モンティ・ホール問題テレビ番組から生まれた史上最も議論を呼んだ確率問題の紹介と解説』という本が出た [1]。この本は、モンティ・ホール問題について詳しく、この本が出る前に私が考えていた問題の拡張についても書かれてしまっていた。この本を読んで、一旦は、私がこれ以上、この問題について何か書く必要はないかと思った。が、それからしばらくして、私がやったことも少しは意味があるかと思い直し、私の思考の跡を備忘的に記録するため、ちょっとした確率モデルのシミュレーションに関する論考を足して、レポートにしておくことにした。

1 序

テレビのショー。三つのドアが用意され、その一つには賞品 (アタリ) が、残り二つにはヤギ (ハズレ) が隠れている。賞品のドアを、プレイヤーはもちろん知らないが、司会者は知っている。ショーの間、ドアの向こうでスタッフが小細工することもない。アタれば賞品がもらえるが、ヤギはもらえない。

プレイヤーが、最初にドアの一つを指定する。司会者はそのドア以外のドアでヤギ (ハズレ) のドアを一つあける。もう一度、プレイヤーは選択を求められる。最初に指定したドアのままでも、残ったもう一つのドアへの変更でも選択できる。ドアが開かれ、最終的な選択だけにもとづき、アタリ、ハズレが決まる。

さぁ、最後の選択で、最初の選択のドアに固執するべきか、それとも司会者が残したドアに乗り替えるべきか？　そもそも、その選択に確率的な差があるのか？

……というのが、モンティ・ホール問題という確率の問題である。モンティ・ホールというのがテレビの司会者の名前なのだそうだ。さて、皆さんはどう考えたのだろうか？
私は、最初、この問題を知ったとき、情報の操作だけで確率が変わるのはおかしいので、残ったドアに賞品がある確率はイーブン、$\frac{1}{2}$ 同志だと考えた。しかし、答えは意外なことに、最初の選択のドアに固執すると確率は $\frac{1}{3}$ のままで、司会者が残したドアに乗り替えると、確率は $\frac{2}{3}$。なんと二倍も確率に差が出るということになる。

これはヒッカケのヒッカケ問題のように思う。司会者が変更を可能だというのだから、一見、司会者が出した新しい提案に乗ったほうが有利かもしれないとまずは思う。これが第一のヒッカケで、落ち着いて考えれば、選択に関する物理的な何かが変わったわけではないので、確率がイーブンだと思う。しかし、これが第二のヒッカケで、本当は、ちゃんと計算すると情報の操作だけで確率が変わっている。…という構造をしている。

ちなみに、私はこの問題と出会ったのは「モンティ・ホール問題」としてではなく、「三囚人問題」と呼ばれるタイプの問題だった。それは、次のような問題である。

三人の囚人 A, B, C と看守がいる。囚人のうちの一人は恩赦になるが、残り二人は処刑されることがわかっている。

誰が恩赦になるか知っている看守に対し、A が「B と C のうち少なくとも一人が処刑されるのは確実だから、二人のうち一人だけ処刑される者の名前を教えても、私についての情報をもらしたことにはならないだろう。一人の名前を教えてくれないか」と頼んだところ、看守は納得して「B は処刑される」と答えた。

それを聞いて A は「これで釈放されるのは自分か C であるから自分の助かる確率は $\frac{1}{3}$ から $\frac{1}{2}$ に増えた」と喜んだ。この A の主張は正しいだろうか？

……と、これが「三囚人問題」で、計算などをしてみると上のモンティ・ホール問題と同じ問題であることがわかる。

このモンティ・ホール問題について、近年、J. ローゼンハウスの『モンティ・ホール問題テレビ番組から生まれた史上最も議論を呼んだ確率問題の紹介と解説』という本が出た[1]。この本は、モンティ・ホール問題について詳しく、主観確率や客観確率の違いなどについても書かれている。さらに、私がこの本が出る前に考えていた問題の拡張についても書かれてしまっていて、この本があるんだから、私がこれ以上、この問題について何か書く必要はないな……と一旦は思ってしまった。

しかし、最近、私は暇にあかせて、シミュレーションについて関心を向けたとき、このモンティ・ホール問題の関連で何か言えないかと考えるようになった。同時に私が考えてきたことも一応、まとめておいたほうが良いなと気力が回復してきたので、本稿を書くこ

とにした。

題に「確率操作」の用語が含むのは、モンティ・ホール問題につまづいたものが考えがちな、この問題を使って何らかの間違いが物理学など世にはびこっていることを示せるのではないか、そこを利用＝確率操作して何かがなせるのではないか……という考えが私にもあったからだ。ただ、本稿は正直、「確率操作」手法については何も述べるに致らなかったというのが公正なところだろう。その願いを「確率操作」で検索するものに託そうというのがこの題の意図の一つである。

2　モンティ・ホール問題または三囚人問題の拡張

前節でモンティ・ホール問題とはどのような問題かすでに述べたが、まずはやや詳しくして問題を繰り返そう。後に拡張を考えるので、前述の本のローゼンハウスにならって、これを問題の**パターン１**とする。

まず三囚人問題の形で述べよう。これは上に載せたままだ。

> 三人の囚人 A, B, C と看守がいる。囚人のうちの一人は恩赦になるが、残り二人は処刑されることがわかっている。誰が恩赦になるか知っている看守に対し、A が「B と C のうち少なくとも一人が処刑されるのは確実だから、二人のうち一人だけ処刑される者の名前を教えても、私についての情報をもらしたことにはならないだろう。一人の名前を教えてくれないか」と頼んだところ、看守は納得して「B は処刑される」と答えた。それを聞いて A は「これで釈放されるのは自分か C であるから自分の助かる確率は $\frac{1}{3}$ から $\frac{1}{2}$ に増えた」と喜んだ。この A の主張は正しいだろうか？

この A, B, C という記号を使って、相当するモンティ・ホール問題を書くと次のようになる。

> 扉 A, B, C があり、二つの扉の後ろにはハズレが、一つの扉の後ろにはアタリがある。プレイヤーが最初に扉 A を選択したあと、司会者が、残った扉 B, C のうちハズレとわかっているほうの扉 B を開けた。プレイヤーは、再選択を許され最初に選択した A か、司会者が残した C かを選べる。どちらを選んだほうが有利か。

ローゼンハウスの本を読めばわかるように、司会者が選択するところはデリケートな表現が必要で、A がアタリで B, C が外れの場合のみ B, C からランダムに選び、そうでない場合は、ハズレのほうを常に選ぶといったふうに言わないといけない。

そして、この答えだが、まず標本空間 Ω を書いて考えてみよう。[賞の位置] $\times$ [最初の選択] $\times$ [司会者の選択] ということで、$\Omega = \{A, B, C\} \times \{A, B, C\} \times \{$ 左, 右 $\}$ を考えよう。ここで司会者の選択は開くほうの扉を示すとし、最初の選択が A のとき 左 は C で 右 は B、最初の選択が B のとき 左 は A で 右 は C、最初の選択が C のとき 左 は B で 右 は A をそれぞれ表すとする。

このうち司会者の選択までが終ったという事象 R を考える。賞の位置が A で最初の選択が B ならば、司会者の選択はアタリを開かないよう A つまり 左 ではありえない。同様に賞の位置が A で最初の選択が C ならば、司会者の選択は A つまり 右 ではありえない。そうやって徐外していくと R は次のような集合になる。

$$\begin{aligned} R = \{&\langle A, A, \text{左} \rangle_C, \langle A, A, \text{右} \rangle_B, \\ &\langle A, B, \text{右} \rangle_C, \\ &\langle A, C, \text{左} \rangle_B, \\ &\langle B, A, \text{左} \rangle_C, \\ &\langle B, B, \text{左} \rangle_A, \langle B, B, \text{右} \rangle_C, \\ &\langle B, C, \text{右} \rangle_A, \\ &\langle C, A, \text{右} \rangle_B, \\ &\langle C, B, \text{左} \rangle_A, \\ &\langle C, C, \text{左} \rangle_B, \langle C, C, \text{右} \rangle_A\}。 \end{aligned}$$

ここでそれぞれの要素を $\langle P, F, D \rangle_M$ という変数で参照することにする。P は賞の位置、F が最初の選択、D が最初の選択を除いた左右のどちらを開けるかで、M が司会者が開けたドアを示す。ここから最初の選択 F が A で M が B であるとき残った C がアタリである確率を求めることができる。

$$\begin{aligned} P(P = C | F = A \wedge M = B \wedge R) &= \frac{|\{\langle C, A, \text{右} \rangle_B\}|}{|\{\langle A, A, \text{右} \rangle_B, \langle C, A, \text{右} \rangle_B\}|} \\ &= \frac{1}{2}。 \end{aligned}$$

さて、この論証は実は間違っている。どうして間違っているかというのを説明するのは難しいが、正しいはずの方法と比較してみるとよい。「正しいはずの方法」、それはコンピュータによるシミュレーション、いわゆるモンテ・カルロ法である。この問題をシミュレーションするプログラミング言語 Perl による実装 `monty_hall_1.pl` は、だいたい次のようになる。

```
our $TRIALS = 10000;
```

```
our $FIX = 0;
my $Homo = 0;
my $Hetero = 0;
my $Valid = 0;

MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my $prize = int(rand(3));
    my $first_choice = int(rand(3));
    my $monty_left;

    if ($first_choice == $prize) {
      $monty_left = int(rand(3 - 1));
      $monty_left++ if $monty_left >= $first_choice;
    } else {
      $monty_left = $prize;
    }

    next if $FIX && ! ($first_choice == 0 && $monty_left == 2);

    $Valid++;

    $Homo++ if $first_choice == $prize;
    $Hetero++ if $monty_left == $prize;
  }
}

REPORT:
{
  printf("Valid Trials = %d (/ %d)\n", $Valid, $TRIALS);
  if ($Valid > 0) {
    printf("Homo = %d (%g)\nHetero = %d (%g)\n",
           $Homo, $Homo / $Valid,
           $Hetero, $Hetero / $Valid);
  } else {
    printf("Homo = 0\nHetero = 0\n");
  }
}
```

本来は、プログラミングの警告を制御する文や、コマンドラインオプションを処理するコードが、さらに入っていたのだが、それらは省略して掲載した。今後、プログラムを掲載する際には、さらに変数定義部と REPORT 部分などを必要に応じて省略し MAIN_LOOP

部のみを掲載することにする。

さて、プログラムはとても単純で、賞の位置 $prize と最初の選択 $first_choice をランダムに選んだあと、司会者が開く場所ではなく司会者が開かずに残しておく扉 $monty_left をランダムに決め、$first_choice に重ならないようにしているだけである。

$FIX が真のときは、最初の選択が 0 で司会者が残したのが 2 である場合の他は有効 (Valid) な試行としては数えないようにしている。$FIX は通常は偽で、オプション --fix が指定されたときのみ真となる。なお、問題の対称性から、$FIX をしてみても確率には影響がないはずである。

そして、最初の選択にアタリがあると $Homo という変数をプラスし、司会者が残したものに賞があると $Hetero という変数をプラスする。

この試行を $TRIALS 回だけ行って、結果を REPORT 部分で表示している。

実行すると次のようになる。

```
$ perl monty_hall_1.pl
Valid Trials = 10000 (/ 10000)
Homo = 3319 (0.3319)
Hetero = 6681 (0.6681)

$ perl monty_hall_1.pl --fix
Valid Trials = 1706 (/ 10000)
Homo = 519 (0.30422)
Hetero = 1187 (0.69578)
```

乱数を使っているので、毎回数値が変わるが、だいたい最初の選択に固執した場合の確率は $\frac{1}{3} = 0.3333\cdots$、司会者が残したほうに乗り替えた場合の確率は $\frac{2}{3} = 0.6666\cdots$ となる。 これは上の $\frac{1}{2}$ という計算と合わない。

簡単なシミュレーションのほうが正しいとすると、間違っているのは上の R を使った論証である。プログラムをよく見てみると、乱数を使っているところで、最初の選択が賞の位置を当てていない場合 ($first_choice != $prize のとき) は $\{$ 左, 右 $\}$ の部分の乱数データを使っていないことに注意せねばならない。これを集合的に書こうと思ったら、その部分は使わないようにしないといけない。

つまり、標本空間を書くときに $\langle P, F, D\rangle_M$ のうち、最初の選択と賞の位置が違う ($P \neq F$) ならば、D が左か右かは使わずに M を決定しなければならないということなのだ。$\langle A, B,$ 右 $\rangle$ は含まれて $\langle A, B,$ 左 $\rangle$ は含まれないとは考えない。$\langle A, B,$ 右 $\rangle_C$ と $\langle A, B,$ 左 $\rangle_C$ が含まれると考えるのである。Ω を書くと次のようになる。

$$\begin{aligned}\Omega = \{&\langle A, A, \text{左} \rangle_C, \langle A, A, \text{右} \rangle_B,\\ &\langle A, B, \text{左} \rangle_C, \langle A, B, \text{右} \rangle_C,\\ &\langle A, C, \text{左} \rangle_B, \langle A, C, \text{右} \rangle_B,\\ &\langle B, A, \text{左} \rangle_C, \langle B, A, \text{右} \rangle_C,\\ &\langle B, B, \text{左} \rangle_A, \langle B, B, \text{右} \rangle_C,\\ &\langle B, C, \text{左} \rangle_A, \langle B, C, \text{右} \rangle_A,\\ &\langle C, A, \text{左} \rangle_B, \langle C, A, \text{右} \rangle_B,\\ &\langle C, B, \text{左} \rangle_A, \langle C, B, \text{右} \rangle_A,\\ &\langle C, C, \text{左} \rangle_B, \langle C, C, \text{右} \rangle_A\}。\end{aligned}$$

そして、最初の選択 F が A で M が B であるとき残った C がアタリである確率を求めると、

$$\begin{aligned}P(P = C|F = A \wedge M = B) &= \frac{|\{\langle C, A, \text{左} \rangle_B, \langle C, A, \text{右} \rangle_B\}|}{|\{\langle A, A, \text{右} \rangle_B, \langle C, A, \text{左} \rangle_B, \langle C, A, \text{右} \rangle_B\}|}\\ &= \frac{2}{3}\end{aligned}$$

となって、シミュレーションの結果と合うことになる。

ところで、上の R を使った論証はまるきり間違った論証だったのだろうか？　あれは何を意味していたのだろう。その論証に相当するのが問題の**パターン２**で、三囚人問題の場合、次のようになる。

> 三人の囚人 A,B,C と看守がいる。囚人のうちの一人は恩赦になるが、残り二人は処刑されることがわかっている。はじめ、誰が恩赦になるか看守はまだ知らないが、もうすぐわかる。わかる前に A がこうもちかけた。「B と C のうち少なくとも一人が処刑されるのは確実だから、二人のうち一人だけ処刑される者の名前を教えても、私についての情報をもらしたことにはならないだろう。一人の名前を教えてくれないか」この主張はパターン１と同じである。さらに A はこう続けた「ただし、処刑されるべき者を (看守が) 今、思い浮かべ、その者が処刑される場合のみ、そう教えて欲しい。」その後、看守は誰が恩赦になるか知ったあと「B が処刑される」と A に告げた。A が釈放される確率は上がったか？

私は、このパターン２を自力で考えついて、これはモンティ・ホール問題では書けないので、三囚人問題を中心に考えていくべきだ……などと一時期思っていたのだが、ローゼ

ンハウスの本を読んで、モンティ・ホール問題にもこれと同様の問題があることを知った。モンティ・ホール問題では、ローゼンハウスの本を引用して、次のようになる。

> モンティは３枚の同一の扉を見せる。１枚には車が隠れており、２枚には山羊が隠れている。あなたは１枚の扉を選ぶが、それはまだ開けない。しかし今回は、モンティは車がどこにあるか知らない。あなたが選ばなかった２枚の扉からランダムに選んで開けるだけだ。その扉には山羊がいた。そこでモンティはあなたに、最初に選んだ扉のままにするか、残った扉に変えるかの選択の機会を提供する。どうすればいいだろう。

上までの言葉づかいでは、「モンティ」が「司会者」、「車」が「アタリ」、「山羊」が「ハズレ」である。

この問題のシミュレーション monty_hall_2.pl は次のようになる。

```
MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my $prize = int(rand(3));
    my $first_choice = int(rand(3));
    my $monty_left;

    $monty_left = int(rand(3 - 1));
    $monty_left++ if $monty_left >= $first_choice;

    next if ! ($prize == $monty_left || $prize == $first_choice);

    next if $FIX && ! ($first_choice == 0 && $monty_left == 2);

    $Valid++;

    $Homo++ if $first_choice == $prize;
    $Hetero++ if $monty_left == $prize;
  }
}
```

先ほどのプログラムよりも簡単になっていると言えるだろう。ただし、$prize が司会者の開けた扉にあった場合 (すなわち $prize == $monty_left でも $prize == $first_choice でもない場合)、その試行はお流れになって、$Valid にカウントされないようになっている。

結果は次のようになる。

```
$ perl monty_hall_2.pl
Valid Trials = 6645 (/ 10000)
Homo = 3276 (0.493002)
Hetero = 3369 (0.506998)

$ perl monty_hall_2.pl --fix
Valid Trials = 1123 (/ 10000)
Homo = 546 (0.486198)
Hetero = 577 (0.513802)
```

これは上の R を使った論証で求めた確率 $\frac{1}{2} = 0.5$ に等しい。条件に合わねければ消すという操作がプログラムでも行われているのが、ちょうど R を作るときに要素を除外していったのと対応している。

ここで、ベイズの定理を使った確率の計算についても言及しておこう。扉 A にアタリ (車) がある事象を C_A と書こう。司会者が扉 B を開く事象を M_B、また今後のために逆に司会者が残した扉が C である事象を L_C と書こう。つまり、C_A の条件下では M_B と L_C は違いはない。今後は論証に便利なように M_X と L_X を使い分けていく。今回は M_X のほうを使う。

まず、パターン 1 について考える。アタリが出る確率は皆同じだから

$$P(C_A) = P(C_B) = P(C_C) = \frac{1}{3}。$$

C_A がすでにわかっているとき、M_B と M_C は等確率で起こる。C_C がわかっているなら常に M_B で、C_B ならば M_B であることはありえない。まとめると、

$$P(M_B|C_A) = \frac{1}{2}、P(M_B|C_B) = 0、P(M_B|C_C) = 1。$$

ベイズの定理を使って、

$$\begin{aligned} P(C_C|M_B) &= \frac{P(M_B|C_C) \cdot P(C_C)}{P(M_B|C_A) \cdot P(C_A) + P(M_B|C_B) \cdot P(C_B) + P(M_B|C_C) \cdot P(C_C)} \\ &= \frac{1 \cdot \frac{1}{3}}{\frac{1}{2} \cdot \frac{1}{3} + 0 \cdot \frac{1}{3} + 1 \cdot \frac{1}{3}} \\ &= \frac{2}{3}。 \end{aligned}$$

次に、パターン 2 について考えよう。そもそもアタリが出る確率は上と同じで

$$P(C_A) = P(C_B) = P(C_C) = \frac{1}{3}。$$

C_A がすでにわかっているとき、M_B と M_C は等確率で起こるのは同じ。C_B ならば M_B であることはありえないのも同じ。しかし、C_C がわかっているときは、M_B と M_C が等確率で起こるからそこだけ違う。まとめると、

$$P(M_B|C_A) = \frac{1}{2}、P(M_B|C_B) = 0、P(M_B|C_C) = \frac{1}{2}。$$

$$\begin{aligned} P(C_C|M_B) &= \frac{P(M_B|C_C) \cdot P(C_C)}{P(M_B|C_A) \cdot P(C_A) + P(M_B|C_B) \cdot P(C_B) + P(M_B|C_C) \cdot P(C_C)} \\ &= \frac{\frac{1}{2} \cdot \frac{1}{3}}{\frac{1}{2} \cdot \frac{1}{3} + 0 \cdot \frac{1}{3} + \frac{1}{2} \cdot \frac{1}{3}} \\ &= \frac{1}{2}。 \end{aligned}$$

ベイズの定理を使った結果は、シミュレーションの答えと合っている。

さて、納得できただろうか？　納得しがたい人に受け容れてもらう方法として、ローゼンハウスは、扉の数を増やした場合について言及している。

> 扉が 100 枚ある。プレイヤーが 1 枚選んだあと、残り 99 枚の中から 98 枚ハズレのものを司会者が選んで開けた。司会者が残した 1 枚に選択を変えてもいいという。選択を変えるべきか？

これを試すのが次のプログラム monty_hall_3_1.pl である。

```
MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my $prize = int(rand($DOORS));
    my $first_choice = int(rand($DOORS));
    my $monty_left;

    if ($first_choice == $prize) {
      $monty_left = int(rand($DOORS - 1));
      $monty_left++ if $monty_left >= $first_choice;
    } else {
      $monty_left = $prize;
```

```
    }

    next if $FIX && ! ($first_choice == 0 && $monty_left == $DOORS - 1);

    $Valid++;

    $Homo++ if $first_choice == $prize;
    $Hetero++ if $monty_left == $prize;
  }
}
```

monty_hall_3_1.pl は monty_hall_1.pl に対応するものだが、同様に monty_hall_2.pl に対応する monty_hall_3_2.pl も作った。これらを実行すると次のようになる。

```
$ perl monty_hall_3_1.pl --door=100
Valid Trials = 10000 (/ 10000)
Homo = 101 (0.0101)
Hetero = 9899 (0.9899)

$ perl monty_hall_3_2.pl --door=100
Valid Trials = 172 (/ 10000)
Homo = 79 (0.459302)
Hetero = 93 (0.540698)
```

ベイズの定理を使って monty_hall_3_1.pl の場合の確率を求めてみよう。

扉を 1 から 100 まで番号付けして扉 1 にアタリがある事象を C_1 と書く。司会者が扉 2 を残す事象を L_2 と書く。最初に扉 1 が選択されたとする。そうしても問題の対称性により一般性を失わないだろう。

扉 i について $P(C_i) = \frac{1}{100}$ である。C_1 のときは残りからランダムに選ばれるので、$P(L_2|C_1) = \frac{1}{99}$。1 でも 2 でもない i について C_i ならば司会者は残り全部の扉を開けるので $P(L_2|C_i) = 0$ である。逆に C_2 ならばそこを残すので、$P(L_2|C_2) = 1$ である。ベイズの定理を使って、

$$
\begin{aligned}
P(C_2|L_2) &= \frac{P(L_2|C_2) \cdot P(C_2)}{P(L_2|C_1) \cdot P(C_1) + P(L_2|C_2) \cdot P(C_2) + \sum_{i \neq 1,2} P(L_2|C_i) \cdot P(C_i)} \\
&= \frac{1 \cdot \frac{1}{100}}{\frac{1}{99} \cdot \frac{1}{100} + 1 \cdot \frac{1}{100} + \sum_{i \neq 1,2} 0 \cdot \frac{1}{100}} \\
&= \frac{99}{100}。
\end{aligned}
$$

シミュレーションの答えとだいたい一致している。

ただ、この扉を 100 枚にする方法では、私は納得しづらかった。私がむしろ納得したのは扉に得点を与えてそれを集計する方法である。

つまり、扉に例えばそれぞれ (2.0, 1.0, 0.0) というポイントを与え、最終的に選ばれた扉のポイントを集計していくというものである。司会者の選択は、最初に選ばれた扉以外の扉のうち、もっともポイントが大きい扉を残すことで行う。ポイントが大きい扉が複数ある場合にはそのうちでランダムで選ぶ。モンティ・ホール問題の設定は、一つアタリで他がハズレなので (1.0, 0.0, 0.0) とポイントを割り振ったときに相当する。

そのプログラム monty_hall_4.pl は次のようなものだ。

```
our @POINT = (2.0, 1.0, 0.0);

sub rand_sort {
  my (@a) = @_;
  my @r;
  return () if @a == 0;
  while (@a > 1) {
    my $i = int(rand(@a));
    push(@r, splice(@a, $i, 1));
  }
  push(@r, pop(@a));
  return @r;
}

MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my @score = rand_sort(@POINT);
    my $first_choice = int(rand(3));
    my $monty_left;

    my @tmp = grep {$first_choice != $_} (0 .. 2);
    @tmp = rand_sort(@tmp); ## $POINT[] が等しい場合には、
                            ## 等確率で $monty_left に残るように。
    @tmp = sort {$score[$a] <=> $score[$b]} @tmp;
    $monty_left = $tmp[@tmp - 1];

    $Valid++;
```

```
        my $last_choice_1 = $first_choice;
        my $last_choice_2 = $monty_left;
        my $last_choice_3 = (int(rand(2)))? $first_choice : $monty_left;

        $Score1 += $score[$last_choice_1];
        $Score2 += $score[$last_choice_2];
        $Score3 += $score[$last_choice_3];
      }
    }
```

Perl では @a などで配列を表す。ここで関数 rand_sort は配列を取ってランダムにその並び方を変えて返す関数である。grep {$first_choice != $_} (0 .. 2) は (0, 1, 2) という配列の中から最初の選択で選ばれなかったもの ($first_choice != $_ なる $_) を配列にして返している。

sort でポイントが小さいもの順に並べ、配列の最後にある最大の要素の番号を $monty_left としている。Perl では、配列を表す @tmp が、数値を欲する文脈 (コンテクスト) で使われたときは、配列の要素数を返すという決まりがある。よってここで @tmp - 1 は常に $2-1=1$ となり、Perl は配列の番号が 0 からはじまるので @tmp の最後の要素を表すことになる。

rand_sort を sort の前にかませてあるのは、@Point の中に同じ数値が含まれていても、ランダムにそれを取得できるようにするためである。

@Point はオプション --point によって指定でき、次のような実行結果が得られる。

```
$ perl monty_hall_4.pl --point=2,1,0
Valid Trials = 10000 (/ 10000)
Score1 = 10073 (1.0073)
Score2 = 16645 (1.6645)
Score3 = 13358 (1.3358)

$ perl monty_hall_4.pl --point=1,0,0
Valid Trials = 10000 (/ 10000)
Score1 = 3336 (0.3336)
Score2 = 6664 (0.6664)
Score3 = 4956 (0.4956)
```

ポイントが 2 か 1 か 0 のとき、司会者には 0, 1 か 0, 2 か 1, 2 かの組が等確率で残りそれぞれその最大値 1 か 2 か 2 の要素が $monty_left に選ばれるのだから、そのスコアの平均は $\frac{1+2+2}{3} = \frac{5}{3} = 1.666\cdots$ となり、上と合っている。

ポイントを 1 か 0 か 0 にした例が、モンティ・ホール問題にあたる。常に司会者が残した扉を選ぶ選択がこの場合も有効で、司会者には 0, 0 か 0, 1 か 0, 1 かの組が等確率

で (または 0, 0 の倍の確率で 0, 1 が) 起き、そのスコアの平均は $\frac{2}{3}$ になる。

ここまでくると問題が少々複雑になっても対応できる。次のような問題を考える。

> 扉が 6 枚ある。最初に選ばれた 2 枚を残し、残り 4 枚の中からハズレのものを司会者が 1 枚選んで開け、全体の残り 5 枚の中から好きなように選んでもらう。確率はどのようになるか。

シミュレーションは次のように書く。

```
MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my $prize = int(rand(6));
    my @monty_left;

    my @tmp = grep {$prize != $_} (2, 3, 4, 5);
    @tmp = rand_sort(@tmp); ## 司会者は $tmp[0] を開くとする。
    @monty_left = grep {$tmp[0] != $_} (2, 3, 4, 5);

    my $last_choice_1 = int(rand(2));
    my $last_choice_2 = $monty_left[int(rand(@monty_left))];

    $Valid++;

    $Score1 ++ if $last_choice_1 == $prize;
    $Score2 ++ if $last_choice_2 == $prize;
  }
}
```

最初に 2 枚選んだところが上では 0 と 1 番に対応し、残りの 4 枚が 2, 3, 4, 5 番に対応する。結果は次のようになる。

```
$ perl monty_hall_5.pl
Valid Trials = 10000 (/ 10000)
Score1 = 1669 (0.1669)
Score2 = 2185 (0.2185)
```

ベイズの定理を用いて求める。扉 1, 2 が最初に選択され、扉 3 が司会者によって開けられたとする。C_i が扉 i にアタリがある事象、M_3 が扉 3 が司会者によって開けられた事象とする。

まず、i にかかわらず $P(C_i) = \frac{1}{6}$ である。扉 1 (または 2) にアタリがあれば、残り 4 枚の条件は同じだから $P(M_3|C_1) = \frac{1}{4}$。扉 3 にアタリがあれば M_3 は起こらないので

$P(M_3|C_3) = 0$。扉 4 (または 5 か 6) にアタリがあれば、残り 3 枚の条件は同じだから $P(M_3|C_4) = \frac{1}{3}$。ゆえに、

$$\begin{aligned} P(C_4|M_3) &= \frac{P(C_4|M_3) \cdot P(C_4)}{\sum P(M_3|C_i) \cdot P(C_i)} \\ &= \frac{\frac{1}{3} \cdot \frac{1}{6}}{2 \cdot \frac{1}{4} \cdot \frac{1}{6} + 0 \cdot \frac{1}{6} + 3 \cdot \frac{1}{3} \cdot \frac{1}{6}} \\ &= \frac{2}{9} = 0.2222 \cdots 。 \end{aligned}$$

同様にして $P(C_1|M_3) = \frac{1}{6} = 0.1666 \cdots$。シミュレーション結果と合っている。

司会者に残った扉のほうが多かろうが、複数の扉が残ろうが、司会者が選別対象としたもののほうを選んだほうが、アタリの確率は大きくなる。

さて、パターン 2 の三囚人問題において、看守が教えてくれたからいいものの、看守が「教えない」という選択をしたという場合、B か C のどちらかが助かるわけで、そうすると残りの A はかならず処刑されることになる。これは看守が情報をもらしたに等しい。それを避けるために逆に A が釈放されても教えない場合をムリヤリ作ったとしたらどうなるだろうか？　A は次のようにいったとしよう。これが**パターン３**になる。

> 「たまたま決めた者が処刑されるなら教え、そうでなければ教えないで下さい。ただそれだけでは、教えないことが私が処刑されることを意味することになります。そこで、私が恩赦になる場合も教えない場合をつくるために、私が恩赦であることがわかったらサイコロを振り、偶数の目が出たときは、今決めた者が処刑されたとしても教えないようにしてください。」

さて、これにモンティ・ホール問題を対応させようとしたら次のようになる。

> 最初の扉が出る前にモンティがサイコロを振ってその目をカードに書いてふせておく。プレイヤーが最初の扉を選んだとき、そこにアタリがあれば、司会者はカードに書かれているのが偶数であることを表にすることによって、その回のチャレンジをなかったことにできる。

さて、ベイズの定理でこれをどう解いたものだろうか。実は、最初の $P(C_A) = \frac{1}{3} \cdot \left(1 - \frac{1}{2}\right) = \frac{1}{6}$ とし、$P(C_B) = P(C_C) = \frac{1}{3}$ とする。$P(C_B) = \frac{1}{3} \cdot \left(1 + \frac{1}{2} \cdot \frac{1}{2}\right)$ などではないというのが注意を要するところである。

あとはパターン 2 の場合と同じで、

$$P(M_B|C_A) = \frac{1}{2}、P(M_B|C_B) = 0、P(M_B|C_C) = \frac{1}{2}。$$

ベイズの定理を使って

$$\begin{aligned} P(C_C|M_B) &= \frac{P(M_B|C_C) \cdot P(C_C)}{P(M_B|C_A) \cdot P(C_A) + P(M_B|C_B) \cdot P(C_B) + P(M_B|C_C) \cdot P(C_C)} \\ &= \frac{\frac{1}{2} \cdot \frac{1}{3}}{\frac{1}{2} \cdot \frac{1}{6} + 0 \cdot \frac{1}{3} + \frac{1}{2} \cdot \frac{1}{3}} \\ &= \frac{2}{3} = 0.6666 \cdots 。 \end{aligned}$$

プログラムで書くと monty_hall_2.pl からほんの少しの変更でいい。次が、その monty_hall_6.pl である。

```
our $REJECTION_POSITIVE = 0.5;

MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my $prize = int(rand(3));
    my $first_choice = int(rand(3));
    my $monty_left;

    next if $first_choice == $prize && rand(1) < $REJECTION_POSITIVE;

    $monty_left = int(rand(3 - 1));
    $monty_left++ if $monty_left >= $first_choice;

    next if ! ($prize == $monty_left || $prize == $first_choice);

    next if $FIX && ! ($first_choice == 0 && $monty_left == 2);

    $Valid++;

    $Homo++ if $first_choice == $prize;
    $Hetero++ if $monty_left == $prize;
  }
}
```

サイコロで偶数の目が出るのは 0.5 の確率だから$REJECTION_POSITIVE の初期値を

0.5 にしている。(今後、同じ変数名が出てくるが、今後のデフォルト値は 0 である。)

実行すると次のようになる。

```
$ perl monty_hall_6.pl
Valid Trials = 5014 (/ 10000)
Homo = 1665 (0.33207)
Hetero = 3349 (0.66793)
```

ベイズの定理を使って求めた確率とだいたい一致している。パターン 1 と同じ確率になってしまったが、$Home か $Hetero しかないので、一方が減れば一方が増えてしまうのは当然の理である。

ここで、パターン 1 とパターン 2 の混合を考える。確率 q でパターン 1 とパターン 2 を切り替えるのである。さらに、上のパターン 3 の$REJECTION_POSITIVE を p として同時に用いる。そして、ドア数を d として扉を番号 $1, \ldots, d$ で区別し、monty_hall_3_1.pl のように、最初の選択のあと残りの $d-1$ 枚の扉のうち一枚を残して全部を司会者が開けるとする。

ベイズの定理を用いた確率の計算からまず示そう。最初に扉 1 が選ばれ、司会者が扉 2 を残したとする。扉 1 がアタリである事象を C_1、司会者が扉 2 を残した事象を L_2 などと表すとする。

まず、アタリの確率はパターン 3 の議論を通じて次のようになる。

$$P(C_1) = \frac{1}{d}(1-p)、P(C_2) = P(C_i) = \frac{1}{d}。$$

ただし、$i \neq 1, 2$。扉 1 にアタリがあれば、残りの扉はどれも同じで $P(L_2|C_1) = \frac{1}{d-1}$。扉 1 でも 2 でもない i について、C_i であれば扉 2 が残るはずがないので $P(L_2|C_i) = 0$。確率 q でパターン 2 のようになり、確率 $1-q$ でパターン 1 のようになるとすると、

$$P(L_2|C_2) = q \cdot \frac{1}{d-1} + (1-q) \cdot 1。$$

だから、

$$\begin{aligned} P(C_2|L_2) &= \frac{P(L_2|C_2) \cdot P(C_2)}{\sum P(C_j) \cdot P(L_2|C_i)} \\ &= \frac{(\frac{q}{d-1} + (1-q)) \cdot \frac{1}{d}}{\frac{1}{d-1} \cdot \frac{1}{d}(1-p) + (\frac{q}{d-1} + (1-q)) \cdot \frac{1}{d} + 0} \\ &= \frac{(d-1) - (d-2)q}{(d-p) - (d-2)q}。 \end{aligned}$$

たとえば $d = 3$、$p = 0.25$、$q = 0.5$ のとき $P(C_2|L_2) = \frac{2}{3} = 0.6666\cdots$ となる。
これをプログラムすると次の monty_hall_7.pl のようになる。

```
our $DOORS = 3;
our $PRISONER_PROTEST = 0;
our $REJECTION = 0.0;
our $REJECTION_POSITIVE = 0;

MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my $prize = int(rand($DOORS));
    my $first_choice = int(rand($DOORS));
    my $monty_left;

    $PositiveInvalid++ if $first_choice == $prize;
    next if rand(1) < $REJECTION;
    next if $first_choice == $prize && rand(1) < $REJECTION_POSITIVE;

    $monty_left = int(rand($DOORS - 1));
    $monty_left++ if $monty_left >= $first_choice;

    if (! ($prize == $monty_left || $prize == $first_choice)) {
      next if rand(1) < $PRISONER_PROTEST;
      $monty_left = $prize;     # if $prize != $first_choice;
    }

    next if $FIX && ! ($first_choice == 0 && $monty_left == $DOORS - 1);

    $Valid++;
    $PositiveInvalid-- if $first_choice == $prize;

    $Homo++ if $first_choice == $prize;
    $Hetero++ if $monty_left == $prize;
  }
}
```

なお、ここで $REJECTION_POSITIVE が p で$PRISONER_PROTEST が q で、$DOORS が d にあたる。この他に全体を無視 (Invalid に) する確率 $REJECTION も使えるようにしている。

これまで出てきた $Home と $Hetero の他に$PositiveInvalid という変数をカウン

トしている。これは、最初の選択がアタリ、つまり「釈放」のとき、無視 (Invalid に) された数で、パターン３の議論では、これがないのが問題になったのであった。

オプションでは変数名が少し変わって、$REJECTION_POSITIVE を--rejection で、$REJECTION を --ignore で、$PRISONER_PROTEST を --prisoner で指定する。

実行結果は次のようになる。

```
$ perl monty_hall_7.pl --rejection=0.25 --prisoner=0.5
Valid Trials = 7455 (/ 10000)
Positive Invalid Trials = 863 (0.339096)
Homo = 2483 (0.333065)
Hetero = 4972 (0.666935)
```

これはベイズの定理で求めた例と一致する。

さらに、これはローゼンハウスの本でも出てきたアイデアだが、扉のアタリ・ハズレの割り当てが等確率ではなく扉ごとに違った比率で指定される場合を考える。

この場合、最早、対称性はないので、プログラムを作るときは、$first_choice をランダムにではなく、直接 0 番に指定してシミュレーションする。プログラム monty_hall_8.pl を先に示そう。

```
our @TENDENCY = (3.0, 1.0, 2.0);
our @M_TENDENCY = (1.0, 1.0, 1.0);

sub sum_array {
  my (@a) = @_;
  my $sum = 0.0;
  foreach my $x (@a) {
    $sum += $x;
  }
  return $sum;
}

sub rand_tendency {
  my (@tend) = @_;
  my $tmp1 = rand(sum_array(@tend));
  my $tmp = 0.0;
  for (my $i = 0; $i < @tend - 1; $i++) {
    $tmp += $tend[$i];
    if ($tmp1 < $tmp) {
      return $i;
    }
  }
```

```
  return @tend - 1;
}

MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my $prize = rand_tendency(@TENDENCY);
    my $first_choice = 0;
    my $monty_left;

    $PositiveInvalid++ if $first_choice == $prize;
    next if rand(1) < $REJECTION;
    next if $first_choice == $prize && rand(1) < $REJECTION_POSITIVE;

    my @tmp = @M_TENDENCY;
    splice(@tmp, $first_choice, 1);
    $monty_left = rand_tendency(@tmp);
    $monty_left++ if $monty_left >= $first_choice;

    if (! ($prize == $monty_left || $prize == $first_choice)) {
      next if rand(1) < $PRISONER_PROTEST;
      $monty_left = $prize;     # if $prize != $first_choice;
    }

    next if $FIX && ! ($first_choice == 0 && $monty_left == $DOORS - 1);

    $Valid++;
    $PositiveInvalid-- if $first_choice == $prize;

    $Homo++ if $first_choice == $prize;
    $Hetero++ if $monty_left == $prize;
  }
}
```

関数 sum_array は配列の数値を全て足し合わせたものを返す。関数 rand_tendency(@tend) は、配列 @tend の数値の比率に従ってランダムに配列の中の要素の番号を返す関数である。上の splice は、@tmp の中の $first_choice (このプログラムでは 0) から要素一個を除いた配列を @tmp とし、今回は使っていないが、返り値としては除かれた要素を返す。

このプログラムでは、アタリ・ハズレの割り当ての他に、司会者の選択についても比率に従った選択がなされるように書かれている。アタリ・ハズレの割り当ての

比率を @TENDENCY、オプションでは --tendency で指定し、司会者の選択の比率を @M_TENDENCY、オプションでは --mtendency で指定する。

実行結果は例えば次のようになる。

```
$ perl monty_hall_8.pl --tendency=3,1,2 --mtendency=1,1,3 \
                       --rejection=0.25 --prisoner=0.5
Valid Trials = 7683 (/ 10000) (=:= 7708.33)
Positive Invalid Trials = 1298 (0.560207)
Homo = 3755 (0.488741 =:= 0.486486)
Hetero = 3928 (0.511259 =:= 0.513514)

$ perl monty_hall_8.pl --tendency=3,1,2 --mtendency=1,1,3 --fix
Valid Trials = 7093 (/ 10000) (=:= 7083.33)
Positive Invalid Trials = 1253 (0.431029)
Homo = 3847 (0.542366 =:= 0.529412)
Hetero = 3246 (0.457634 =:= 0.470588)
```

ここでは 扉 1, 2, 3 のアタリ・ハズレの比率を 3 : 1 : 2 に、司会者の選択の比率を 1 : 1 : 3 に指定している。

ベイズの定理を使った計算は結果だけ書いて行こう。アタリ・ハズレの比率は扉 i について $\frac{a_i}{\sum a_j}$、司会者の選択の比率は扉 i について $\frac{b_i}{\sum b_j}$ とする。

$FIX を使って固定した場合と全体で $Homo、$Hetero を見た場合とで結果が異なる。

まず $FIX した場合、

$$P(L_d) = (1-p)\frac{b_d}{\sum_{i\neq 1} b_i}\frac{a_1}{\sum a_i} + \frac{a_d}{\sum a_i}\left(\frac{b_d}{\sum_{i\neq 1} b_i}\cdot q + 1 - q\right)、$$

$$P(C_1|L_d) = \frac{(1-p)\frac{b_d}{\sum_{i\neq 1} b_i}\frac{a_1}{\sum a_i}}{P(L_d)}。$$

なお、$P(C_d|L_d) = 1 - P(C_1|L_d)$ である。

上の出力では、$a_1 : a_2 : a_3 = 3 : 1 : 2$、$a_1 : a_2 : a_3 = 1 : 1 : 3$、$d = 3$、$p = 0$、$q = 0$ であるから、$P(L_d) = \frac{17}{24} = 0.7083\cdots$、$P(C_1|L_d) = \frac{9}{17} = 0.5294\cdots$ となる。なお、$P(L_d)$ に試行数を掛けたものが Valid な数となる。ここでは出力と計算結果が合っている。

次に $FIX しない場合、

$$P(\bigcup_{i\neq 1} L_i) = (1-p)\frac{a_1}{\sum a_i} + \left(1 - \frac{a_1}{\sum a_i}\right)(1-q) + \frac{\sum_{i\neq 1} a_i b_i}{\sum a_i \sum_{i\neq 1} b_i} q、$$

$$P(C_1|\bigcup_{i\neq 1} L_i) = \frac{(1-p)\frac{a_1}{\sum a_i}}{P(\bigcup_{i\neq 1} L_i)}。$$

上の出力では $a_1 : a_2 : a_3 = 3 : 1 : 2$、$a_1 : a_2 : a_3 = 1 : 1 : 3$、$d = 3$、$p = 0.5$、$q = 0.25$ であるから、$P(\bigcup_{i\neq 1} L_i) = \frac{37}{48} = 0.7708\cdots$、$P(C_1|\bigcup_{i\neq 1} L_i) = \frac{18}{37} = 0.4864\cdots$ となる。なお、$P(\bigcup_{i\neq 1} L_i)$ に試行数を掛けたものが `Valid` な数となる。ここでは出力と計算結果が合っている。

3 簡単な確率モデル

前節では、モンティ・ホール問題の拡張を論じたが、ここからはそれを用いた簡単な確率モデルを作っていく。

ここからは、プログラムでは扉の数が 3 以上の場合にも対応しているように見えるかもしれないが、扉の数は 3 に限定して考えていくことにする。

まず、アタリ・ハズレの割り当ての比率 $a_1 : a_2 : a_3$ の各 a_i を 0 から 1 までの乱数で決めたとき、どういう戦略をとればアタリの回数を多くできるかを考える。$a_1 \leq a_2 \leq a_3$ と仮定しても一般性を失わないだろう。

最良の選択は、おそらく最初に比率が最も小さい a_1 を選択して、司会者に a_2 と a_3 の高い比率どうしのうちから選択肢を減らしてもらい、司会者が残したほうに乗り替えるものだと予想ができる。さらに $a_1 + a_2 \leq a_3$ ぐらいに差がついた a_3 ならば、最大の a_3 をまず選んで最初の選択に固執するのも悪くはないかもしれない。

`monty_sim_1_1.pl` は、最初の選択は同確率でランダムに選び、必ず司会者の残したほうに乗り替えるという選択をする。`monty_sim_1_2.pl` は、上の最良選択と言った最初の選択は比率が最も小さかったものを選び、必ず司会者が残したほうを選ぶという選択をする。`monty_sim_1_3.pl` は、最初の選択で最も大きい比率のものを選び、最初の選択に固執する。`monty_sim_1_4.pl` は、$a_1 + a_2 \leq a_3$ の場合は、最初の選択で a_3 を選んで、最初の選択に固執するが、それ以外の場合は、最も小さい比率を最初の選択で選んで、必ず司会者の残したほうを選択するという戦略をとる。

実行結果は次のようになる。

```
$ monty_sim_1_1.pl
Valid Trials = 10000 (/ 10000)
Score = 6610 (0.661)
```

```
$ monty_sim_1_2.pl
Valid Trials = 10000 (/ 10000)
Score = 8497 (0.8497)

$ monty_sim_1_3.pl
Valid Trials = 10000 (/ 10000)
Score = 5299 (0.5299)

$ monty_sim_1_4.pl
Valid Trials = 10000 (/ 10000)
Score = 6990 (0.699)
First Choice = 4981 (0.4981)
```

見ると、最初の選択に固執するのはマズイことがわかる。monty_sim_1_4.pl でもダメだったのは、$a_1 + a_2 \leq a_3$ であったとしても、a_1 を最初の選択としてから、司会者に a_2、a_3 のうちから残してもらったほうが有利になるからである。

上の実行結果はモンティ・ホール問題のパターン 1 の場合だが、最初の選択と司会者の残す選択に差がなくなりがちなパターン 2 の場合どうなるだろうか。それを試してみる。

```
$ monty_sim_1_1.pl --prisoner=1
Valid Trials = 6692 (/ 10000)
Positive Invalid Trials = 0 (0)
Score = 3355 (0.501345)

$ monty_sim_1_2.pl --prisoner=1
Valid Trials = 5776 (/ 10000)
Positive Invalid Trials = 0 (0)
Score = 4234 (0.733033)

$ monty_sim_1_3.pl --prisoner=1
Valid Trials = 7620 (/ 10000)
Positive Invalid Trials = 0 (0)
Score = 5282 (0.693176)

$ monty_sim_1_4.pl --prisoner=1
Valid Trials = 7036 (/ 10000)
Positive Invalid Trials = 0 (0)
Score = 5057 (0.718732)
First Choice = 4042 (0.574474)
```

実行結果は興味深い。monty_sim_1_1.pl よりも monty_sim_1_2.pl のほうが成績がいいのは、比率の高いほうから最後の選択が行われるからだろう。スコア (確率) でみると、monty_sim_1_2.pl のほうが Valid な試行が少なくなるが、monty_1_4.pl よりこ

こでも成績がいい。

monty_sim_1_3.pl と monty_sim_1_4.pl についてだが、スコア (確率) としては monty_sim_1_4.pl のほうが大きい。monty_sim_1_4.pl の First Choice は最初の選択に固執した割合を示していて、これが意外に大きいことがわかる。ただ、最初の選択に固執する場合は、monty_sim_1_3.pl も同じなので、それほど大きな差がないときに比率が最小のものを選択して司会者に残りを選別してもらっている分、成績が良くなるのは、monty_sim_1_2.pl が monty_1_4.pl より成績がいいことからわかる。

monty_sim_1_4.pl のプログラムだけ示しておこう。

```
MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my @tendency;
    for (my $i = 0; $i < $DOORS; $i++) {
      $tendency[$i] = rand(1);
    }

    my @rank = rand_sort(0 .. (@tendency - 1));
    @rank = sort {$tendency[$a] <=> $tendency[$b]} @rank;

    my $prize = rand_tendency(@tendency);
    my $fix_first;
    my $first_choice;
    my $monty_left;
    my $last_choice;
    {
      my $tmp = 0.0;
      for (my $i = 0; $i < $DOORS - 1; $i++) {
        $tmp += $tendency[$rank[$i]];
      }
      if ($tmp <= $tendency[$rank[$DOORS - 1]]) {
        $fix_first = 1;
        $first_choice = $rank[$DOORS - 1];
      } else {
        $fix_first = 0;
        $first_choice = $rank[0];
      }
    }

    $PositiveInvalid++ if $first_choice == $prize;
    next if rand(1) < $REJECTION;
    next if $first_choice == $prize && rand(1) < $REJECTION_POSITIVE;
```

```
      my @tmp = @M_TENDENCY;
      splice(@tmp, $first_choice, 1);
      $monty_left = rand_tendency(@tmp);
      $monty_left++ if $monty_left >= $first_choice;

      if (! ($prize == $monty_left || $prize == $first_choice)) {
        next if rand(1) < $PRISONER_PROTEST;
        $monty_left = $prize;     # if $prize != $first_choice;
      }

      next if $FIX && ! ($first_choice == 0 && $monty_left == $DOORS - 1);

      if ($fix_first) {
        $last_choice = $first_choice;
        $Fix_First++;
      } else {
        $last_choice = $monty_left;
      }

      $Valid++;
      $PositiveInvalid-- if $first_choice == $prize;

      $Score += ($fix_first)? $BONUS : 1 if $last_choice == $prize;
    }
  }
```

大きくは monty_hall_8.pl をもとにしていて、比率をランダムに決めているところと、戦略を決めているところが新しいところになる。なお、First Choice と表示していたのは、$Fix_First がカウントしているものである。

ここまでアタリ・ハズレの比率 $a_1 : a_2 : a_3$ がわかっている前提でやってきたが、これがわからないがだいたいの傾向はつかめるという場合を考えてみたい。比率を決めるのが乱数のままなら、それはムチャな要求なので、比率は前の試行などを元に一定のアルゴリズムで決まるようにする。

そのアルゴリズムだが、私は昔 balance_mail_redirect.pl という秘書などへのメール配送時にランダムだが最終的な配送数はできるだけ平等に割り振るというプログラムを書いたことがある。そのアルゴリズムを流用しよう。式で表すと、まず第 n 試行について扉 i に対し中間的変数 $B_{n,i}$ を考える。$B_{1,i} = 0$ とする。第 n 試行でのアタリの扉の位

置を P_n で表すと、

$$B_{n+1,i} = \begin{cases} B_{n,i} \cdot F - \frac{2}{3} & (P_n = i)、 \\ B_{n,i} \cdot F + \frac{1}{3} & (\text{otherwise}) \end{cases}$$

とした上で、$a_i = P_0^{B_{n,i}}$ とする。ここで F と P_0 は定数で、プログラム上ではそれぞれ $FORGET、$POW として扱われ、オプションでは --forget と--power でそれぞれ指定する。

このアルゴリズムの特徴を見てみよう。次のようなプログラム test_bl_2.pl を作る。

```
MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my @tendency;
    for (my $i = 0; $i < $DOORS; $i++) {
      $tendency[$i] = exp(log($POW) * $Balance[$i]);
    }
    my $prize = rand_tendency(@tendency);

    $Occur[$prize]++;
    if (!defined $Prev) {
      $Prev = $prize;
      $Length = 1;
      $Point[1] = 0 if ! defined $Point[1];
      $Point[1]++;
    } elsif ($Prev != $prize) {
      $Prev = $prize;
      $Length = 1;
      $Point[1] = 0 if ! defined $Point[1];
      $Point[1]++;
    } else {
      $Length++;
      $Point[$Length - 1]--;
      $Point[$Length] = 0 if ! defined $Point[$Length];
      $Point[$Length]++;
    }

    for (my $i = 0; $i < @Balance; $i++) {
      if ($i == $prize) {
        $Balance[$i] = $Balance[$i] * $FORGET - (@Balance - 1) / @Balance;
      } else {
        $Balance[$i] = $Balance[$i] * $FORGET + 1 / @Balance;
      }
    }
```

```
  }
}

REPORT:
{
  for (my $i = 0; $i < @Occur; $i++) {
    printf("$i: $Occur[$i] (%g)\n", $Occur[$i] / $TRIALS);
  }
  print "\n";
  for (my $i = 0; $i < @Point; $i++) {
    print "$i: $Point[$i]\n";
  }
}
```

つまり、扉がどれだけアタリになったかを扉ごとに出力してから、アタリが同じ扉でどれだけの長さ連続したかを長さごとに出力する。

比較のために上の

```
$tendency[$i] = exp(log($POW) * $Balance[$i]);
```

を

```
$tendency[$i] = rand(1);
```

と元に戻した test_bl_1.pl も作る。なお、test_bl_ の bl は balance の略である。

実行結果は次のようになる。

```
$ perl test_bl_1.pl
0: 3380 (0.338)
1: 3237 (0.3237)
2: 3383 (0.3383)

0: 0
1: 4326
2: 1535
3: 487
4: 174
5: 53
6: 17
7: 8
8: 3

$ perl test_bl_2.pl --power=1.5 --forget=1
0: 3333 (0.3333)
1: 3334 (0.3334)
```

```
2: 3333 (0.3333)

0: 0
1: 4879
2: 1653
3: 462
4: 87
5: 15
6: 1

$ perl test_bl_2.pl --power=3 --forget=1
0: 3333 (0.3333)
1: 3333 (0.3333)
2: 3334 (0.3334)

0: 0
1: 5641
2: 1660
3: 302
4: 32
5: 1

$ perl test_bl_2.pl --power=1.5 --forget=0.8
0: 3306 (0.3306)
1: 3319 (0.3319)
2: 3375 (0.3375)

0: 0
1: 5128
2: 1617
3: 407
4: 84
5: 15
6: 1

$ perl test_bl_2.pl --power=3 --forget=0.8
0: 3336 (0.3336)
1: 3326 (0.3326)
2: 3338 (0.3338)

0: 0
1: 6414
2: 1556
```

```
3: 142
4: 12
```

$POW はもちろん $FORGET もアタリの連続を少なくする方向に作用している。まったく忘れない場合は、かなり昔のロスまで覚えているため、その分、連続を長くしてしまうのだろう。

このアルゴリズムに対して戦略を検討するために monty_sim_2_1.pl、monty_sim_2_2.pl、monty_sim_2_3.pl、monty_sim_2_4.pl、monty_sim_2_5.pl と作って調べてみたのだが、どうも良くない。良い性質が出て来ない。実は、最初、$FORGET はなしでやってみて、あとから導入したのだが、それでも良くならない。

$FORGET は 1 以上にすると連続がとんでもない長さまで出る場合が出てきてどうも不具合があるが、欲しいのは、そのような過去のロスが強化されて反映されるようなものである……と考えて、アルゴリズムをいじったのが次のようなものになる。

$$B_{n+1,i} = \begin{cases} 0 & (P_n = i)、\\ B_{n,i} \cdot F + \frac{1}{3} & (\text{otherwise})。 \end{cases}$$

なお、n、i、P_n、P_0、F の意味は同じで、$B_{1,i} = 0$ とするのも同じである。

これを試すのは test_bl_2.pl の以下の部分を

```
for (my $i = 0; $i < @Balance; $i++) {
  if ($i == $prize) {
    $Balance[$i] = $Balance[$i] * $FORGET - (@Balance - 1) / @Balance;
  } else {
    $Balance[$i] = $Balance[$i] * $FORGET + 1 / @Balance;
  }
}
```

次のように置き換えた test_bl_3.pl を作る。

```
for (my $i = 0; $i < @Balance; $i++) {
  if ($i == $prize) {
    $Balance[$i] = 0;
  } else {
    $Balance[$i] = $Balance[$i] * $FORGET + 1 / @Balance;
  }
}
```

実行すると次のようになる。

```
$ perl test_bl_3.pl --power=1.5 --forget=1.5
0: 3304 (0.3304)
```

```
1: 3333 (0.3333)
2: 3363 (0.3363)

0: 0
1: 5898
2: 1494
3: 295
4: 51
5: 5

$ perl test_bl_3.pl --power=3 --forget=1.5
0: 3301 (0.3301)
1: 3354 (0.3354)
2: 3345 (0.3345)

0: 0
1: 7114
2: 1243
3: 128
4: 4

$ perl test_bl_3.pl --power=1.5 --forget=3
0: 3356 (0.3356)
1: 3338 (0.3338)
2: 3306 (0.3306)

0: 0
1: 6657
2: 1466
3: 137

$ perl test_bl_3.pl --power=1.5 --forget=0.8
0: 3313 (0.3313)
1: 3323 (0.3323)
2: 3364 (0.3364)

0: 0
1: 5162
2: 1497
3: 417
4: 115
5: 23
6: 3
```

$FORGET を 1 以上にすれば、かなりアタリの連続が少なくなることがわかる。もちろん、test_bl_2.pl ではバランスが重視されていたので扉ごとのアタリの数はそろっていたが、test_bl_3.pl ではそれは失なわれている。

このアルゴリズムに対して戦略を考えていく。アタリ・ハズレの比率は直接わからないが、連続してアタリということは少ないことがわかっている。前にアタリであったものを最初の選択にし、司会者が残したものを最後の選択とするようにすれば、イイ線行くのではないかと考えた。

それを試すのが次の monty_sim_2_7.pl である。

```
our $BONUS = 1;
our $POW = 1.5;
our $FORGET = 1.5;
our $CHALLENGE_LENGTH = 2;
our $MEMORY_SIZE = 20;

MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my @tendency;
    for (my $i = 0; $i < $DOORS; $i++) {
      $tendency[$i] = exp(log($POW) * $Balance[$i]);
    }

    my $prize = rand_tendency(@tendency);
    my $fix_first;
    my $first_choice;
    my $monty_left;
    my $last_choice;

    my $cont_length = 0;
    for (my $i = 0; $i < @Memory; $i++) {
      last if $Memory[0] != $Memory[$i];
      $cont_length++;
    }

    if ($cont_length >= $CHALLENGE_LENGTH) {
      $fix_first = 0;
      $first_choice = $Memory[0];
    } else {
      $fix_first = 0;
      $first_choice = int(rand($DOORS));
```

```
  }

  $PositiveInvalid++ if $first_choice == $prize;
  next if rand(1) < $REJECTION;
  next if $first_choice == $prize && rand(1) < $REJECTION_POSITIVE;

  my @tmp = @M_TENDENCY;
  splice(@tmp, $first_choice, 1);
  $monty_left = rand_tendency(@tmp);
  $monty_left++ if $monty_left >= $first_choice;

  if (! ($prize == $monty_left || $prize == $first_choice)) {
    next if rand(1) < $PRISONER_PROTEST;
    $monty_left = $prize;     # if $prize != $first_choice;
  }

  next if $FIX && ! ($first_choice == 0 && $monty_left == $DOORS - 1);

  if ($fix_first) {
    $last_choice = $first_choice;
    $Fix_First++;
  } else {
    $last_choice = $monty_left;
  }

  $Valid++;
  $PositiveInvalid-- if $first_choice == $prize;

  $Score += ($fix_first)? $BONUS : 1 if $last_choice == $prize;
  $Score_First += 1 if $last_choice == $prize && $fix_first;

  for (my $i = 0; $i < @Balance; $i++) {
    if ($i == $prize) {
      $Balance[$i] = 0;
    } else {
      $Balance[$i] = $Balance[$i] * $FORGET + 1 / @Balance;
    }
  }
  unshift(@Memory, $prize);
  while (@Memory >= $MEMORY_SIZE) {
    pop(@Memory);
  }
}
```

```
}
```

オプション --challenge で指定できる $CHALLENGE_LENGTH 分の長さがあれば、一つ前のアタリの位置を最初の選択とし、そうでなければランダムに最初の選択をし、必ず司会者が残したほうを選ぶという戦略をとる。

比較対照のためにアルゴリズムを使わず rand(1) で比率を決める場合の monty_sim_2_1.pl も作る。

実行結果は次のようになる。

```
$ perl monty_sim_2_1.pl --challenge=1
Valid Trials = 10000 (/ 10000)
Score = 6628 (0.6628)

$ perl monty_sim_2_1.pl --challenge=2
Valid Trials = 10000 (/ 10000)
Score = 6672 (0.6672)

$ perl monty_sim_2_7.pl --challenge=1
Valid Trials = 10000 (/ 10000)
Score = 7690 (0.769)

$ perl monty_sim_2_7.pl --challenge=2
Valid Trials = 10000 (/ 10000)
Score = 7049 (0.7049)

$ perl monty_sim_2_7.pl --challenge=3
Valid Trials = 10000 (/ 10000)
Score = 6718 (0.6718)
```

アタリの長さは 1 のものだけ見れば十分なようである。

次に、アタリが最近出ていないものは、アタリが出る確率が高くなると、このアルゴリズムでは言える。それを試すのが monty_sim_2_6.pl で次のようになる。

```
MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my @tendency;
    for (my $i = 0; $i < $DOORS; $i++) {
      $tendency[$i] = exp(log($POW) * $Balance[$i]);
    }

    my $prize = rand_tendency(@tendency);
    my $fix_first;
```

```
my $first_choice;
my $monty_left;
my $last_choice;

my $cont_length = 0;
for (my $i = 0; $i < @Memory; $i++) {
  last if $Memory[0] != $Memory[$i];
  $cont_length++;
}
my $rcont_length = 0;
my $rest;
{
  my %tmp;
  for (my $i = 0; $i < $DOORS; $i++) {
    $tmp{$i} = 1;
  }
  for (my $i = 0; $i < @Memory; $i++) {
    delete $tmp{$Memory[$i]} if exists $tmp{$Memory[$i]};
    my @tmp = %tmp;
    if (@tmp == 0) {
      $rest = $Memory[$i];
      last;
    }
    $rcont_length++;
  }
  if (! defined $rest) {
    my @tmp = rand_sort(keys %tmp);
    $rest = pop(@tmp);
  }
}

if ($rcont_length >= $CHALLENGE_LENGTH) {
  $fix_first = 1;
  $first_choice = $rest;
} else {
  $fix_first = 0;
  $first_choice = (@Memory)? $Memory[0] : int(rand($DOORS));
}

$PositiveInvalid++ if $first_choice == $prize;
next if rand(1) < $REJECTION;
next if $first_choice == $prize && rand(1) < $REJECTION_POSITIVE;
```

```
      my @tmp = @M_TENDENCY;
      splice(@tmp, $first_choice, 1);
      $monty_left = rand_tendency(@tmp);
      $monty_left++ if $monty_left >= $first_choice;

      if (! ($prize == $monty_left || $prize == $first_choice)) {
        next if rand(1) < $PRISONER_PROTEST;
        $monty_left = $prize;     # if $prize != $first_choice;
      }

      next if $FIX && ! ($first_choice == 0 && $monty_left == $DOORS - 1);

      if ($fix_first) {
        $last_choice = $first_choice;
        $Fix_First++;
      } else {
        $last_choice = $monty_left;
      }

      $Valid++;
      $PositiveInvalid-- if $first_choice == $prize;

      $Score += ($fix_first)? $BONUS : 1 if $last_choice == $prize;
      $Score_First += 1 if $last_choice == $prize && $fix_first;

      for (my $i = 0; $i < @Balance; $i++) {
        if ($i == $prize) {
          $Balance[$i] = 0;
        } else {
          $Balance[$i] = $Balance[$i] * $FORGET + 1 / @Balance;
        }
      }
      unshift(@Memory, $prize);
      while (@Memory >= $MEMORY_SIZE) {
        pop(@Memory);
      }
    }
  }
```

$CHALLENGE_LENGTH が、ここでは、アタリが最近どれだけの長さ出ていない場合、最初の選択にその扉を指定し最初の選択に固執するかを決めている。ただし、monty_sim_1_*.pl のところで見たように、最初の選択に固執するのはどのみち有利ではないため、最初の選択のオッズを上げる……ボーナス$BONUS をスコアに加算するよう

に設定できるようにした。

比較のために monty_sim_2_1.pl と同様に、monty_sim_2_6.pl の比率指定部分をアルゴリズムを使わず、rand(1) とランダムにした monty_sim_2_3.pl も作っておく。

実行結果は次のようになる。

```
$ perl monty_sim_2_3.pl --challenge=2
Valid Trials = 10000 (/ 10000)
Score = 3251 (0.3251)
First Choice = 9998 (0.9998)

$ perl monty_sim_2_3.pl --challenge=3
Valid Trials = 10000 (/ 10000)
Score = 4026 (0.4026)
First Choice = 7806 (0.7806)

$ perl monty_sim_2_3.pl --challenge=4
Valid Trials = 10000 (/ 10000)
Score = 4804 (0.4804)
First Choice = 5512 (0.5512)

$ perl monty_sim_2_6.pl --challenge=2
Valid Trials = 10000 (/ 10000)
Score = 4933 (0.4933)
First Choice = 9998 (0.9998)
  Score(First) = 4931 (0.493199)

$ perl monty_sim_2_6.pl --challenge=3
Valid Trials = 10000 (/ 10000)
Score = 6163 (0.6163)
First Choice = 6290 (0.629)
  Score(First) = 3457 (0.549603)

$ perl monty_sim_2_6.pl --challenge=4
Valid Trials = 10000 (/ 10000)
Score = 7096 (0.7096)
First Choice = 3045 (0.3045)
  Score(First) = 1941 (0.637438)
```

アルゴリズムの効果でスコアが高く出ることがわかる。

ボーナスについても試してみよう。

```
$ perl monty_sim_2_6.pl --challenge=2 --bonus=1.7
Valid Trials = 10000 (/ 10000)
Score = 8341 (0.83419)
```

```
First Choice = 9998 (0.9998)
  Score(First) = 4907 (0.490798)

$ perl monty_sim_2_6.pl --challenge=3 --bonus=1.7
Valid Trials = 10000 (/ 10000)
Score = 8446 (0.84467)
First Choice = 6459 (0.6459)
  Score(First) = 3511 (0.543583)

$ perl monty_sim_2_6.pl --challenge=4 --bonus=1.7
Valid Trials = 10000 (/ 10000)
Score = 8460 (0.84604)
First Choice = 3001 (0.3001)
  Score(First) = 1922 (0.640453)

$ perl monty_sim_2_6.pl --challenge=2 --bonus=2
Valid Trials = 10000 (/ 10000)
Score = 9819 (0.9819)
First Choice = 9998 (0.9998)
  Score(First) = 4909 (0.490998)

$ perl monty_sim_2_6.pl --challenge=3 --bonus=2
Valid Trials = 10000 (/ 10000)
Score = 9618 (0.9618)
First Choice = 6305 (0.6305)
  Score(First) = 3455 (0.547978)

$ perl monty_sim_2_6.pl --challenge=4 --bonus=2
Valid Trials = 10000 (/ 10000)
Score = 9004 (0.9004)
First Choice = 3044 (0.3044)
  Score(First) = 1939 (0.636991)
```

元の確率が $\frac{1}{3}$ と $\frac{2}{3}$ で 2 倍の差なのだから、ボーナスが 2 を越えると、最初の選択に固執するほうが有利になるため、$CHALLENGE_LENGTH を長くして最初の選択への固執の回数が減ればスコアが下がっていくことが予想される。一方、チャレンジが有利であれば、ボーナスが 2 より小さくても十分、効果が期待でき、チャレンジの確率、すなわち Score(First) で現れている確率が増えるごとにスコアが増えていくと予想される。$BONUS が 1 から 2 の間に $CHALLENGE_LENGTH の長さにかかわらず増えも減りもしない数値が存在しそうだが、その意味がどれほど重要かはよくわからない。

次に、人気度と呼ぶパラメータを導入する。扉 i について Q_i は 1 から -1 をとる数

で、比率計算を先の (二番目の) $B_{n,i}$ を用いて

$$a_i = P_0^{B_{n,i}} P_Q^{Q_i}$$

とする。P_Q も定数である。つまり、アルゴリズム的変化に定数部分があるようにするのである。

この人気度を導入した test_bl_3.pl に対応する test_bl_5.pl は次のようになる。

```
our @POPULARITY = (1.0, 0.0, -1.0);
our $P_POW = 1.5;

MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my @tendency;
    for (my $i = 0; $i < $DOORS; $i++) {
      $tendency[$i] = exp(log($POW) * $Balance[$i]
                          + log($P_POW) * $POPULARITY[$i]);
    }
    my $prize = rand_tendency(@tendency);

    $Occur[$prize]++;
    if (!defined $Prev) {
      $Prev = $prize;
      $Length = 1;
      $Point[1] = 0 if ! defined $Point[1];
      $Point[1]++;
    } elsif ($Prev != $prize) {
      $Prev = $prize;
      $Length = 1;
      $Point[1] = 0 if ! defined $Point[1];
      $Point[1]++;
    } else {
      $Length++;
      $Point[$Length - 1]--;
      $Point[$Length] = 0 if ! defined $Point[$Length];
      $Point[$Length]++;
    }

    for (my $i = 0; $i < @Balance; $i++) {
      if ($i == $prize) {
        $Balance[$i] = 0;
      } else {
```

```
            $Balance[$i] = $Balance[$i] * $FORGET + 1 / @Balance;
          }
        }
      }
    }
```

実行結果は次のようになる。

```
$ perl test_bl_5.pl --popularity=1,0,-1 --forget=1.5 --power=1.5
0: 4112 (0.4112)
1: 3253 (0.3253)
2: 2635 (0.2635)

0: 0
1: 5802
2: 1422
3: 346
4: 70
5: 6
6: 1
```

扉ごとのアタリの数で大きく差が付いていることがわかるが、アタリの連続はそれほど続かないようだ。

また、人気度は、司会者の扉の選択にも影響するようにする。人気のない扉ほど選択されやすくしよう。比率 $b_1 : b_2 : b_3$ について、

$$b_i = P_M^{-Q_i}$$

とする。ただし、P_M は定数で、デフォルトは $P_M = 1.5$ にしている。

これに関し、アタリの連続の長さについてチャレンジする monty_sim_2_7.pl に相当する monty_sim_2_8.pl を作る。ただし、チャレンジするのは人気度が高い場合に限る。実行すると次のようになる。

```
$ perl monty_sim_2_8.pl --challenge=1
Valid Trials = 10000 (/ 10000)
Score = 7545 (0.7545)

$ perl monty_sim_2_8.pl --challenge=2
Valid Trials = 10000 (/ 10000)
Score = 7634 (0.7634)

$ perl monty_sim_2_8.pl --challenge=3
Valid Trials = 10000 (/ 10000)
Score = 7560 (0.756)
```

これを見ると、人気度が高いところでは、一度扉がアタリになったあと他のものが次に出そうだとはすぐには言えなくて、二度連続してアタリが出るまで、続いてアタリが出ることは少ないと考えないほうが良さそうである。

その結果を取り入れながら、monty_sim_2_6.pl に対応するものを作ったのが monty_sim_2_9.pl になる。実行すると次のようになる。

```
$ perl monty_sim_2_9.pl --challenge=4
Valid Trials = 10000 (/ 10000)
Score = 6897 (0.6897)
First Choice = 3448 (0.3448)
  Score(First) = 2041 (0.591937)

$ perl monty_sim_2_9.pl --challenge=5
Valid Trials = 10000 (/ 10000)
Score = 7489 (0.7489)
First Choice = 1390 (0.139)
  Score(First) = 1007 (0.72446)

$ perl monty_sim_2_9.pl --challenge=6
Valid Trials = 10000 (/ 10000)
Score = 7605 (0.7605)
First Choice = 496 (0.0496)
  Score(First) = 423 (0.852823)

$ perl monty_sim_2_9.pl --challenge=4 --bonus=1.2
Valid Trials = 10000 (/ 10000)
Score = 7323 (0.73236)
First Choice = 3488 (0.3488)
  Score(First) = 2053 (0.588589)

$ perl monty_sim_2_9.pl --challenge=5 --bonus=1.2
Valid Trials = 10000 (/ 10000)
Score = 7679 (0.76794)
First Choice = 1490 (0.149)
  Score(First) = 1072 (0.719463)

$ perl monty_sim_2_9.pl --challenge=6 --bonus=1.2
Valid Trials = 10000 (/ 10000)
Score = 7621 (0.76212)
First Choice = 464 (0.0464)
  Score(First) = 391 (0.842672)
```

長いチャレンジの結果が良く、その確率 (Score(First)) が $\frac{2}{3}$ を越えている。これな

らば、ボーナスなしでもいいかもしれない。

人気度を導入する際、司会者の選択の比率 $b_1 : b_2 : b_3$ も決定したのだった。それを活かせないかと考え、$PRISONER_PROTEST が 1 に近い場合は、もともとランダムに最初の選択をしたあと、人気度が高い (比率が低い) ものは最初の選択に固執するようにしてみた。それが monty_sim_2_10.pl である。ところが、人気度が低いものについて最初の選択に固執するようにした場合の monty_sim_2_11.pl を作ってみると、そちらのほうが成績が良い。人気度が高かろうが低かろうが最初の選択に固執するようにした monty_sim_2_12.pl は予想どおり前の二つほど成績はよくない。前節のベイズの定理にもとづいた式に戻って考えてみると、最初の選択の人気度が司会者の残したものの人気度より高ければ良さそうなので、それを試す monty_sim_2_13.pl を作ってみたが、monty_sim_2_11.pl より成績が悪い。成績の良し悪しといっても大した差でないのも事実であるが、いちおう、この先は monty_sim_2_11.pl のプランに沿ってシミュレーションを続けることにする。

実行結果は次のようになる。

```
$ perl monty_sim_2_10.pl --challenge=5 --prisoner=1
Valid Trials = 6124 (/ 10000)
Positive Invalid Trials = 0 (0)
Score = 3883 (0.634063)
First Choice = 1286 (0.209993)
  Score(First) = 954 (0.741835)

$ perl monty_sim_2_11.pl --challenge=5 --prisoner=1
Valid Trials = 6165 (/ 10000)
Positive Invalid Trials = 0 (0)
Score = 4087 (0.662936)
First Choice = 1513 (0.245418)
  Score(First) = 1133 (0.748843)

$ perl monty_sim_2_12.pl --challenge=5 --prisoner=1
Valid Trials = 6149 (/ 10000)
Positive Invalid Trials = 0 (0)
Score = 4020 (0.653765)
First Choice = 2257 (0.367052)
  Score(First) = 1492 (0.661054)

$ perl monty_sim_2_13.pl --challenge=5 --prisoner=1
Valid Trials = 6085 (/ 10000)
Positive Invalid Trials = 0 (0)
```

```
Score = 3987 (0.655218)
First Choice = 1686 (0.277075)
  Score(First) = 1224 (0.725979)
```

今度は、$PRISONER_PROTEST と $REJECTION_POSITIVE を動かしてみる。司会者が残したほうを選択して、アタリが出たら$PRISONER_PROTEST を増やし、ハズレたら減らす。最初の選択に固執して、アタリが出たら $REJECTION_POSITIVE を増やし、ハズレたら減らす。最初の選択に固執すると $PRISONER_PROTEST が少し回復する。司会者が残したほうを選択すると $REJECTION_POSITIVE が少し回復する。

そうやって書いた確率モデルが次の monty_sim_3_1.pl である。

```
our @POPULARITY = (1.0, 0.0, -1.0);
our $P_POW = 1.5;
our $M_POW = 1.5;
our $CHALLENGE_LENGTH = 5;
our @PP_PARAM = (0.2, 0.2, 0.05);
our @RP_PARAM = (0.1, 0.1, 0.05);

MAIN_LOOP:
{
  for (my $i = 0; $i < $TRIALS; $i++) {
    my @tendency;
    for (my $i = 0; $i < $DOORS; $i++) {
      $tendency[$i] = exp(log($POW) * $Balance[$i]
                          + log($P_POW) * $POPULARITY[$i]);
    }

    my $prize = rand_tendency(@tendency);
    my $fix_first;
    my $first_choice;
    my $monty_left;
    my $last_choice;

    my $cont_length = 0;
    for (my $i = 0; $i < @Memory; $i++) {
      last if $Memory[0] != $Memory[$i];
      $cont_length++;
    }
    my $rcont_length = 0;
    my $rest;
    {
      my %tmp;
```

```
  for (my $i = 0; $i < $DOORS; $i++) {
    $tmp{$i} = 1;
  }
  for (my $i = 0; $i < @Memory; $i++) {
    delete $tmp{$Memory[$i]} if exists $tmp{$Memory[$i]};
    my @tmp = %tmp;
    if (@tmp == 0) {
      $rest = $Memory[$i];
      last;
    }
    $rcont_length++;
  }
  if (! defined $rest) {
    my @tmp = rand_sort(keys %tmp);
    $rest = pop(@tmp);
  }
}

my $rand_first = 0;
if ($rcont_length >= $CHALLENGE_LENGTH) {
  $fix_first = 1;
  $first_choice = $rest;
} else {
  if (@Memory > 0 && $POPULARITY[$Memory[0]] > 0.5) {
    if ($cont_length >= 2) {
      $fix_first = 0;
      $first_choice = $Memory[0];
    } else {
      $fix_first = 0;
      $first_choice = int(rand($DOORS));
      $rand_first = 1;
    }
  } else {
    $fix_first = 0;
    if (@Memory) {
      $first_choice = $Memory[0];
    } else {
      $rand_first = 1;
      $first_choice = int(rand($DOORS));
    }
  }
}
```

```
$PositiveInvalid++ if $first_choice == $prize;
next if rand(1) < $REJECTION;
next if $first_choice == $prize && rand(1) < $REJECTION_POSITIVE;

my @m_tendency;
for (my $i = 0; $i < $DOORS; $i++) {
  $m_tendency[$i] = exp(- log($M_POW) * $POPULARITY[$i]);
}
my @tmp = @m_tendency;
splice(@tmp, $first_choice, 1);
$monty_left = rand_tendency(@tmp);
$monty_left++ if $monty_left >= $first_choice;

if (! ($prize == $monty_left || $prize == $first_choice)) {
  next if rand(1) < $PRISONER_PROTEST;
  $monty_left = $prize;     # if $prize != $first_choice;
}

next if $FIX && ! ($first_choice == 0 && $monty_left == $DOORS - 1);

if ($PRISONER_PROTEST >= 0.8
    && $REJECTION_POSITIVE < $PRISONER_PROTEST / 2) {
  if (! $fix_first && $rand_first && $POPULARITY[$monty_left] < -0.5) {
    $fix_first = 1;
  }
}

if ($fix_first) {
  $last_choice = $first_choice;
  $Fix_First++;
} else {
  $last_choice = $monty_left;
}

$Valid++;
$PositiveInvalid-- if $first_choice == $prize;

$Score += ($fix_first)? $BONUS : 1 if $last_choice == $prize;
$Score_First += 1 if $last_choice == $prize && $fix_first;

if ($fix_first) {
  if ($last_choice == $prize) {
    $REJECTION_POSITIVE += $RP_PARAM[0];
```

```
            $REJECTION_POSITIVE = 0.5 if $REJECTION_POSITIVE > 0.5;
            $REJECTION_POSITIVE = 0 if $REJECTION_POSITIVE < 0;
          } else {
            $REJECTION_POSITIVE -= $RP_PARAM[1];
            $REJECTION_POSITIVE = 0.5 if $REJECTION_POSITIVE > 0.5;
            $REJECTION_POSITIVE = 0 if $REJECTION_POSITIVE < 0;
          }
          $PRISONER_PROTEST -= $PP_PARAM[2];
          $PRISONER_PROTEST = 0 if $PRISONER_PROTEST < 0;
          $PRISONER_PROTEST = 1 if $PRISONER_PROTEST > 1;
        } else {
          if ($last_choice == $prize) {
            $PRISONER_PROTEST += $PP_PARAM[0];
            $PRISONER_PROTEST = 0 if $PRISONER_PROTEST < 0;
            $PRISONER_PROTEST = 1 if $PRISONER_PROTEST > 1;
          } else {
            $PRISONER_PROTEST -= $PP_PARAM[1];
            $PRISONER_PROTEST = 0 if $PRISONER_PROTEST < 0;
            $PRISONER_PROTEST = 1 if $PRISONER_PROTEST > 1;
          }
          $REJECTION_POSITIVE -= $RP_PARAM[2];
          $REJECTION_POSITIVE = 0.5 if $REJECTION_POSITIVE > 0.5;
          $REJECTION_POSITIVE = 0 if $REJECTION_POSITIVE < 0;
        }

        for (my $i = 0; $i < @Balance; $i++) {
          if ($i == $prize) {
            $Balance[$i] = 0;
          } else {
            $Balance[$i] = $Balance[$i] * $FORGET + 1 / @Balance;
          }
        }
        unshift(@Memory, $prize);
        while (@Memory >= $MEMORY_SIZE) {
          pop(@Memory);
        }
      }
    }
```

この他、比較対照のため、アタリ・ハズレの比率をアルゴリズムではなくランダムにしてみた monty_sim_3_2.pl、アルゴリズムは使いながら、戦略はまったくなし、つまり、最初の選択も乱数で、最後の選択も乱数でどちらか決める monty_sim_3_3.pl、アタリ・ハズレの比率も乱数、戦略もまったく乱数の monty_sim_3_4.pl、アルゴリズムを使い

ながら、戦略としては最もシンプルな前にアタリの出た扉を最初に選択し、必ず司会者の残したほうに乗り替える monty_sim_3_5.pl を作った。

実行してみると次のようになる。

```
$ perl monty_sim_3_1.pl --challenge=5
Valid Trials = 6831 (/ 10000)
Positive Invalid Trials = 64 (0.0201956)
Score = 4692 (0.686869)
First Choice = 1460 (0.213732)
  Score(First) = 1061 (0.726712)

$ perl monty_sim_3_2.pl --challenge=5
Valid Trials = 8048 (/ 10000)
Positive Invalid Trials = 97 (0.0496926)
Score = 4117 (0.511556)
First Choice = 3216 (0.399602)
  Score(First) = 1244 (0.386816)

$ perl monty_sim_3_3.pl --challenge=5
Valid Trials = 8255 (/ 10000)
Positive Invalid Trials = 134 (0.0767908)
Score = 4057 (0.49146)
First Choice = 4188 (0.507329)
  Score(First) = 1597 (0.381328)

$ perl monty_sim_3_4.pl --challenge=5
Valid Trials = 8133 (/ 10000)
Positive Invalid Trials = 152 (0.081414)
Score = 4117 (0.506209)
First Choice = 3991 (0.490717)
  Score(First) = 1589 (0.398146)

$ perl monty_sim_3_5.pl --challenge=5
Valid Trials = 6348 (/ 10000)
Positive Invalid Trials = 0 (0)
Score = 4152 (0.654064)
First Choice = 0 (0)
```

確かに、いろいろ調べて苦労して構築した monty_sim_3_1.pl が一番成績がいいが、最も戦略が簡単な monty_sim_3_5.pl と比べて大きな差があると言えるほどではない。

monty_sim_3_2.pl が 0.5 に近いスコアなのは、アルゴリズムに対応している戦略が、乱数にもあまり弱くないからである。

少し論を離れて、ジャンケンの対戦をするプログラムを考えよう。乱数だけのプログラムがかなり強いことが予想できるだろう。というか、学習とか無用で、最強かもしれない。しかし、負けるの覚悟で規則的に手を出すプログラムがあったとしよう。学習をするプログラムは、その規則的な相手には勝つことができる。乱数だけのプログラムは規則的な相手にもイーブンな戦いしかできない。

ここで一回の勝ち負けではなく、複数の対戦でのポイントというのを考えよう。規則的に手を出すプログラムが多くあれば、乱数だけのプログラムより学習をするプログラムのほうが勝ち星を多く獲得できるだろう。一見、愚かな規則性が、学習の効果を発揮する余地を作るのである。

今回のプログラムも、アルゴリズムによる規則性が `@Memory` を使った戦略を有効なものとしているのだ。乱数はここでもそこそこの成績を出すが、戦略にはかなわない。

`Valid Trials` が少ないのは、`$PRISONER_PROTEST` が 1 に近いところで動いているからだろう。つまり、常態はパターン２、最初の選択に固執するのと司会者が残すほうを選ぶのとが $\frac{1}{3}:\frac{1}{3}$ で、これが「自然な状態」であり、モンティ・ホール問題や三囚人問題で最初の直感であるところの確率はイーブンという予想は、これを意味していたのだ！……というのが結論として出したくて、私は確率モデルを形成してきたのである。しかし、そうなっているのは、私がそう作ったからに過ぎない。どうも論証に失敗したようである。

4　終論・反省

ということで、モンティ・ホール問題の拡張について備忘のために何か書いておくという目的は達せられたと思うが、確率操作シミュレーションすなわち確率モデルについてはさんざんな結果に終ったと言わざるを得ない。

確率モデルで「やらなかったこと」についていくつか弁解したい。

一つは、最初の選択に固執する場合のボーナスに関して。何回か待っているとボーナスが出るようにしようかと思ったが、`$CHALLENGE_LENGTH` を調べていて、何回か待っている間に、変化があるかどうか見ることにあまり意味を見出せないように思って、ボーナスについて考えるのをやめた。また、`$PRISONER_PROTEST` を動かすとき、それが高い状態だと最初の選択に固執するのにも少しは意味があるので、ボーナスなくてもやるスコアを上げるため最初の選択への固執が行われると考えたのも、ボーナスが必要ないと思った一因である。それともボーナスがあれば、何か良い戦略がありえただろうか？

もう一つは混合ゲームのようなものは目指さなかったという点に関して。最初の選択に固執するかセオリー通り司会者が残したほうに乗り替えるかの二択だから、戦略としてや

れることが少ない。別のゲームを同時にしていて、別のゲームをしている間にパラメータが回復したりすれば、また別のことも言えるかとも考えたが、複雑になるだけ、別のゲームの (おそらく単純な) 性質に支配されるだけではないかと思い、それも形にはしなかった。

さらに、本当は人気度 `@POPULARITY` もその他のパラメータのように正解・不正解によってパラメータをいろいろ変動させることを考えていたが、うまい変動のさせ方、その `@tendency` への反映の仕方が思いつかず、固定することになった。ここを何とかして、人気を上方にへばりつける戦略とかがどうたらこうたら……とかできるような気もしていたのだが、扉３つのうちの人気では意味がないかと思い、とりやめた。扉３つに１０人ぐらいの人物の割り当てるとかも、今考えればありえたが、話が複雑になるだけだろうし、見送る。

本稿では Perl プログラムのソースを多く載せた。ネットで流通すればよく印刷のことはあまり考えず、載せるかどうか迷ったら載せようという方針で多く載せた。図・表がなく、出力ベタ読みを求めるのも問題があるかもしれないが、これは著者が楽をするためにそれでヨシとした。

今さら、モンティ・ホール問題につまづく人がいるかはわからないが、本稿がそういう人の一助となれば幸いである。

なお、本稿の pdf は上で引用した Perl プログラムと一緒に zip されてネットに上がっているはずである。本稿の pdf だけを手に入れた方は、zip を別に探して入手していただきたい。

参考文献

[1] Jason Rosenhouse. 松浦 俊輔 訳. 『モンティ・ホール問題 テレビ番組から生まれた史上最も議論を呼んだ確率問題の紹介と解説』. 2013 年, 2009 年 (原著).

[2] 市川 伸一. 『確率の理解を探る ３囚人問題とその周辺』. 共立出版. 1998 年.