2章
確率的プログラミング：コンピュータの助けを借りる

前章ではシンプルなA/Bテストを題材としてベイズ推論の流れ、そして統計的仮説検定の方法を学びました。ある確率でデータが0または1を取るというシンプルなベルヌーイ試行から出発し、確率計算を施すことでさまざまな知見を引き出すことができました。

しかし、常にこのように簡単に物事が運ぶとは限りません。今までは比較的簡単な計算で事後分布が導けましたが、対象となるデータの種類が異なればまた一から計算を展開する必要があります。たとえば、ECサイトでユーザあたりの購入点数を最大化したい場合にはデータは0以上の整数（0点、1点……）で表されます。一方、ユーザの滞在時間を最大化したい場合には0以上の連続値（5.3秒、10.8秒など）で表されると考えられるでしょう。このように、最適化したい対象が変わるたびに計算をするのは大変そうです。

そこで強力な武器になるのが1.5節でも触れた統計モデルです。データが生成される過程を統計モデルのかたちで記述することで、具体的な問題を抽象化し、他者と、そしてコンピュータと共有できるようになります。さらに統計モデルを**確率的プログラミング言語（probabilistic programming language, PPL）**で記述すると、コンピュータが事後分布を推論し、さまざまな統計量を計算して出力してくれます。つまり、分析者はひとまず具体的な計算のことは忘れて、目の前の問題を統計モデルに落とし込む創造的な作業に集中できるのです！

確率的プログラミングとは統計モデルをソースコードとして記述し、自動で推論を行う枠組みのことです。特にそのようなプログラミングをサポートする言語やライブラリのことを確率的プログラミング言語と呼びます。本章では、なかでもPythonで利用できる確率的プログラミングライブラリPyMC3を取り上げて、ソフトウェアドリブンなベイズ推論の方法を紹介します。この章を読み終えた頃には、クリック率以

外の指標を最適化するA/Bテストも自信を持って設計・分析できるようになっているでしょう。

2.1　統計モデルの記述とサンプリングの実行

まずは、ふたたびアリスとボブのレポートに戻って確率的プログラミングを使った統計的仮説検定をしてみましょう。最初にやることは、必要なモジュールを読み込むことです。PyMC3の可視化メソッドを最大限利用するためには、ベイズ統計モデルを解析するための便利な機能を提供するArviZモジュールもインストールする必要があります。Colabのデフォルトの実行環境ではArviZがインストールされていませんので、ArviZとPyMC3を利用するために最初のセルで下記を実行してください。

```
!pip install -U arviz==0.9.0 pymc3==3.9.3
```

ColabおよびJupyterノートブックでは、セルの冒頭に!をつけることでシェルコマンドを実行できます。このようにpipを使ってPythonパッケージをインストールするシェルコマンドを実行することで、指定したバージョンのPyMC3およびArviZが実行環境にインストールされ、ノートブックから利用できるようになります。

その後、以下のコードを実行してPyMC3その他必要なモジュールを読み込み、ModuleNotFoundErrorが現れないことを確認してください。ここではpymc3モジュールをpmとして読み込んでいます。

```
import numpy as np
from matplotlib import pyplot as plt
import pymc3 as pm
```

次に式(1.9)で示した統計モデルを記述していきます。今回扱う統計モデルの事前分布は一様分布で、尤度関数は二項分布ですから、そのことをそのままコードに落としていきます。まずは、アリスのレポートのデザインA案のクリック率の事後分布からのサンプルを求めましょう。

```
N = 40  # アリスのデザインA案の表示数
a = 2   # アリスのデザインA案のクリック数

with pm.Model() as model:
  theta = pm.Uniform('theta', lower=0, upper=1)
  obs = pm.Binomial('a', p=theta, n=N, observed=a)
```

```
    trace = pm.sample(5000, chains=2)
```

最初にpm.Modelメソッドを実行することで統計モデルオブジェクトmodelを生成します。このコンテキストの中で、統計モデルを構成する確率分布とその関係を記述していきます。pm.Uniformは一様分布を表すクラスで、ここでは確率変数thetaが取りうる範囲を$0 \leqq \theta \leqq 1$と指定しています。もちろん、この箇所を$\alpha = 1, \beta = 1$のベータ分布（1.6節でも述べたように、これは一様分布と一致するのでした）を事前分布として記述することも可能で、その場合は

```
theta = pm.Beta('theta', alpha=1, beta=1)
```

と書き直すことになります。ここでは、ひとまず事前分布がベータ分布で書けることを知らないものとして話を進めましょう。

次に、尤度関数は二項分布でしたからpm.Binomialでそのことを表現します。二項分布$\mathrm{Binomial}(\theta, N)$は試行の成功確率$\theta$と試行回数$N$の2つのパラメータで表されるのでした。それらをそれぞれpm.Binomialの引数pとnに渡して確率分布を設定します。最後に、この二項分布から生成される確率変数aすなわち合計クリック数はすでに観測されているので、observed引数に観測データaを渡します。

さて、pm.sampleを実行すると推論が実行され、記述した統計モデルの事後分布からの大量のサンプルが得られます。pm.sampleの第一引数は、抽出したいサンプルの数を指定します。ここでは5000個の乱数を生成することにします。chainsは並行して一連のサンプリングを行う回数を指定します。ここでは2としますので、合計$5000 \times 2 = 10000$個のサンプルが得られることになります。

上記のコードを著者の環境で実行したところ、プログラム実行中に以下のような表示が得られました。

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Sequential sampling (2 chains in 1 job)
NUTS: [theta]
 100.00% [6000/6000 00:02<00:00 Sampling chain 0, 0 divergences]
 100.00% [6000/6000 00:02<00:00 Sampling chain 1, 0 divergences]
Sampling 2 chains for 1_000 tune and 5_000 draw iterations
(2_000 + 10_000 draws total) took 5 seconds.
```

ここで注意したいのは、5000個のサンプルを2回得ているはずなのに、出力結果には6000と表示されていることです。これは、実はPyMC3が裏側で指定された数より

も1000回多い6000回乱数をサンプリングしていることを意味します。なぜ、余分な1000回のサンプリングを行うのでしょうか？ それは、推論の初期の段階で得られるサンプルは品質が悪いからです。

本書ではPyMC3の推論で用いられる具体的なアルゴリズムには触れませんが、大雑把に言うと、ある初期値から最適なパラメータを求めてうろうろと探索するような動きをしています。大量の乱数を生成して、どんどん状態を更新していくこのアルゴリズムは総称して**MCMC（Markov chain Monte Carlo, マルコフ連鎖モンテカルロ法）**と呼ばれます。モンテカルロ法（Monte Carlo method）は、乱数を大量に生成することでなんらかの問題を解く方法の総称です。一方、マルコフ連鎖（Markov chain）とは（離散的な）時間とともにどんどん状態が移り変わる過程のことで、将来の状態が現在の状態のみで規定されるという性質（マルコフ性）を持つものです。

MCMCはある初期値から最適なパラメータのまわりをうろつくように状態を遷移させていくのですが、その初期値は無作為に設定されるため、最適なパラメータとはかけ離れた値が設定される可能性もあります。したがって品質の良いサンプルを得るためには、初期値の影響ができるだけ残っていない、探索が適度に進んでから得られたものだけを使ったほうがいいのです。このように、探索の初期値の影響を取り除くために捨ててしまう期間のことを**バーンイン（burn-in）**といいます。PyMC3のデフォルトではこのバーンインが最初の1000サンプルに設定されているため、1000サンプルだけ多かったのです。あくまでこれは捨てるためのサンプルであって、最終的に得られるサンプルには残されません。具体的なMCMCのアルゴリズムについては[Kruschke14]の第7章などを参照してください。

MCMCによって得られたサンプルは`trace`に格納されます。PyMC3には、得られたサンプルを解釈するための便利なさまざまなメソッドがあり、`pm.traceplot`に`trace`を渡すと、MCMCによって得られたサンプルを可視化できます。

```
with model:
  pm.traceplot(trace)
```

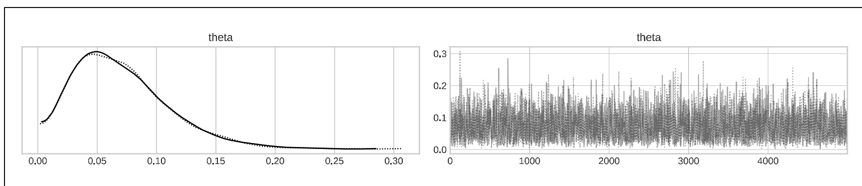

図2-1 `pm.traceplot`による、事後分布からのサンプルの可視化

図2-1の左側には対象の確率変数について推論された事後分布が図示されています。右側にはバーンイン以降に得られたサンプルの軌跡が図示されています。横軸にはサンプリングの回数、縦軸にはサンプルされた確率変数の値が示されています。この図を見るときに注目すべきポイントは、サンプルがある一定の分布に収束しているか否かです。確率変数のサンプルの軌跡がある帯を中心に広がっていれば、ある一定の分布の上でサンプルが得られていることがわかります。

一方で図2-2のようにサンプルの軌跡が蛇行していると、それはある一定の分布に定まっていないことを意味します。これはマルコフ連鎖が収束していない兆候であり、MCMCがうまくいっていないことを表しています。この場合は、十分に長いバーンイン期間を取る、事前分布を見直す、もしくは統計モデルそのものを見直すといった対策を講じる必要があります。

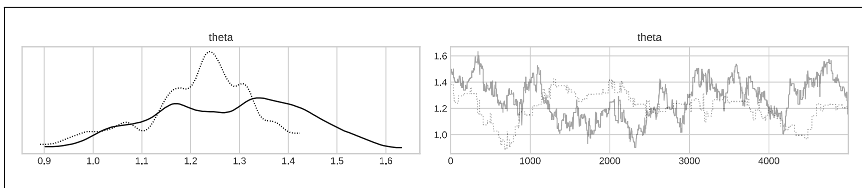

図2-2　マルコフ連鎖が収束せず、MCMCがうまくいっていない例

`pm.summary`にサンプルを渡すと、代表的な要約統計量が表示されます。ここでは95% HDIを考えるものとして、`hdi_prob`パラメータに0.95を渡しました。

```
with model:
  print(pm.summary(trace, hdi_prob=0.95))
```

	mean	sd	hdi_2.5%	hdi_97.5%
theta	0.071	0.039	0.009	0.149

`pm.summary`メソッドによって出力される要約統計量はPyMC3のバージョンによって異なります。ここでは代表的な統計量のみを抜き出して説明しています。