Санкт-Петербургский государственный университет

Страшко Владислав Алексеевич

Выпускная квалификационная работа

Разработка программного комплекса методов оценивания малых вероятностей

Уровень образования: магистратура

Направление 01.04.02 «Прикладная математика и информатика»

Основная образовательная программа ВМ.5688.2018 «Прикладная математика и информатика»

Профессиональная траектория «Статистическое моделирование»

Научный руководитель:

к. ф.-м. н., доцент А. И. Коробейников

Рецензент:

Доцент, Университет ИТМО

к. ф.-м. н., доцент А. А. Сергушичев

Санкт-Петербург

2020

Оглавление

Введение

Предметом исследования данной работы является оценка вероятностей редких событий. Обычно задача имеет простую постановку: оценить $P(\xi \in A)$ для некоторой случайной величины с известным нам распределением и некоторого множества A. При этом, как правило, интерес представляют такие множества A, для которых данная вероятность близка к нулю. Наивное решение такой задачи являет собой интегрирование по методу Монте-Карло, но при действительно малых вероятностях моделирование нужного распределения становится весьма трудоёмкой задачей. При этом оценки, полученные по независимой выборке, как правило, обладают большой дисперсией. Существуют различные методы уменьшения дисперсии оценки по методу Монте-Карло. Например, одним из них является метод существенной выборки, который позволяет получить интересующую оценку на основе выборки из некоторого специального распределения. Однако моделирование выборки из вообще говоря произвольного распределения — непростая задача. В данной работе рассмотрен метод Монте-Карло по схеме марковской цепи. Его идея состоит в получении оценок Монте-Карло не по независимой выборке, а по траектории некоторой марковской цепи. Мы рассмотрим различные алгоритмы, которые позволяют моделировать марковские цепи с заранее заданным инвариантным распределением, на основе траектории которых можно строить оценки. Оказывается, что класс марковских цепей, для которых оценки имеют состоятельность и асимптотическую нормальность, весьма широк. Большинство из таких алгоритмов так или иначе основаны на моделировании по методу Метрополиса-Гастингса [1]. Это очень простой в реализации и в то же время гибкий алгоритм, к тому же имеющий множество обобщений и адаптаций под различные задачи. Также очень часто для эффективного моделирования редких состояний некоторой случайной величины важно правильно подобрать плотность для моделирования существенной выборки. Алгоритм Ванга-Ландау решает ([2]) эту задачу. Это адаптивный алгоритм построения плотности некоторого специального вида, при помощи которой можно получить хорошие (в смысле дисперсии) оценки по методу Монте-Карло. Кроме того, весьма важно оценивать качество получаемых оценок, а также уметь строить для них доверительные интервалы. Для этого нужно оценивать дисперсию. В данной работе мы рассмотрим методы оценки дисперсии вдоль траектории марковской цепи. Таким образом, исходная задача разбивается на несколь-

ко последовательных подзадач, для каждой из которых рассматриваются известные алгоритмы.

Также мы увидим, что многие задачи оценивания вероятности формулируются некоторым общим образом, поэтому в данной работе рассматривается общий подход к решению задачи и предлагается небольшой фреймворк, предоставляющий подходящие инструменты.

Что касается непосредственно реализации, то в этом плане тоже возникает ряд проблем: есть много готовых решений, связанных с моделированием марковских цепей, но задача, рассматриваемая в этой работе имеет определённую специфику: пространство состояний дискретное и иногда очень большое. Поэтому хранение в явном виде пространства состояний неприемлемо. Это сразу делает почти все готовые решения неприменимыми для нашего случая. Поэтому в данной работе мы представляем расширенный вариант библиотеки PyMC3, который позволяет моделировать произвольные случайные объекты в дискретном пространстве состояний и на его основе получаем реализацию библиотеки для оценивания вероятностей редких событий.

В Главе 1 изложены необходимые теоретические факты, а также некоторые алгоритмы.

Глава 2 посвящена задаче вычисления вероятности максимального совпадения строк фиксированной длины над некоторым конечным алфавитом, причем ответ можно вычислить явно. Мы протестируем для этой задачи рассматриваемые в данной работе методы. В этом семестре был рассмотрен алгоритм параллельного темперирования, а также получены оценки по этому методу. Кроме того, добавлены графики, демонстрирующие поведение траекторий на бесконечности.

Глава 3 посвящена непосредственно реализации библиотеки. На данный момент проведён обзор существующих решений в рамках методов Монте-Карло по марковской цепи, таких как Stan, PyMC3. Кроме того, там рассматривается расширение библиотеки PyMC3, а также на его базе реализован пользовательский интерфейс, содержащий моделирование траектории, вычисление вероятности, вычисление дсиперсии и построение доверительного интервала После этого был воспроизведен пример из главы 2 с использованием новых инструментов.

Глава 1

Вспомогательные факты

1.1. Постановка задачи

Для начала формализуем исследуемую задачу. Пусть задано вероятностное пространство $(\Omega, \mathcal{F}, P)$ и на нем определена случайная величина ξ, принимающая значения в некотором пространстве $\mathcal{X}$. При этом будем считать, что распределение $\mathcal{P}$ этой случайной величины нам заранее известно. Далее, пусть задано некоторое множество A. Задача состоит в вычислении $P(\xi \in A)$. Известно, что

$$P(\xi \in A) = \mathbb{E}\left[\mathbb{I}_A(\xi)\right] = \int_X \mathbb{I}_A(x)\mathcal{P}(dx),$$

где $\mathbb{I}_A(x)$ — индикатор множества A. Далее мы рассмотрим различные методы оценивания этой величины.

1.2. Интегрирование методом Монте-Карло

Пусть ν — вероятностная мера, заданная на борелевской σ-алгебре $\mathcal{B}$ подмножеств $\mathcal{X} \subseteq \mathbb{R}^d$, и предположим, что нужно вычислить

$$\mathbb{E}_\nu\left[h(\xi)\right] = \int_{\mathcal{X}} h(x)\nu(dx), \tag{1.1}$$

где $h(x)$ — измеримая функция. Если ν имеет плотность $f(x)$ относительно меры Лебега, то (1.1) принимает вид

$$\mathbb{E}_f\left[h(\xi)\right] = \int_{\mathcal{X}} h(x)f(x)dx.$$

Теперь предположим, что у нас есть $X_1, \ldots, X_n$ — выборка из распределения ν. Тогда оценкой величины (1.1) по методу Монте-Карло будет

$$\bar{h}_n = \frac{1}{n}\sum_{i=1}^{n} h(X_i). \tag{1.2}$$

Оценка (1.2) является состоятельной по закону больших чисел. В частности, для оценки $P(\xi \in A)$ в качестве $h(x)$ возьмём $\mathbb{I}_A(x)$. Тогда оценкой будет

$$\bar{h}_n = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_A(X_i). \tag{1.3}$$

1.2.1. Оценка при помощи существенной выборки

Чтобы получить оценку величины (1.1), не обязательно иметь выборку из распределения ν. Рассмотрим равенство

$$\mathbb{E}_f\left[h(\xi)\right] = \int_{\mathcal{X}} h(x)f(x)dx = \int_{\mathcal{X}} h(x)\frac{f(x)}{g(x)}g(x)dx = \mathbb{E}_g\left[\frac{h(\xi)f(\xi)}{g(\xi)}\right],$$

и потребуем, чтобы из $g(x) = 0$, следовало $f(x) = 0$. Тогда оценка

$$\tilde{h}_n = \frac{1}{n}\sum_{i=1}^{n} h(X_i)\frac{f(X_i)}{g(X_i)} \tag{1.4}$$

будет оценкой интеграла (1.1) по методу существенной выборки. Здесь $X_1, \ldots, X_n$ — выборка из распределения с плотностью $g(x)$. При правильном подборе $g(x)$ оценки по существенной выборке обладают меньшей дисперсией. Действительно, если подобрать плотность $g(X_i) \propto h(X_i)f(X_i)$, то оценка будет близка к константе, и, соответственно будет обладать малой дисперсией.

1.3. Метод Монте-Карло по схеме марковской цепи

Иногда получить независимую выборку из некоторого распределения π не представляется возможным. В таком случае можно воспользоваться последовательностью зависимых величин $\{X_i\}$. Одним классов таких последовательностей являются марковские цепи.

1.3.1. Марковские цепи

Мы приведём здесь основные определения и теоретические результаты относительно марковских цепей, которые пригодятся в рамках данной работы. Более подробно теория изложена в [1].

Определение 1. *Марковской цепью называется последовательность случайных величин* $\{X_i : i = 0, 1, 2, \ldots\}$ *такая, что для любого измеримого множества* $A \in \mathcal{X}$ *выполняется* $P(X_{t+1} \in A | X_0 = x_0, \ldots, X_t = x_t) = P(X_{t+1} \in A | X_t = x_t)$ *для* $\mathcal{P}_{X_0,\ldots,X_t}$*-почти всех* $(x_0, \ldots, x_t)$ *для* $t \geq 0$. *Здесь* $\mathcal{P}_{X_0,\ldots,X_t}$ *— совместное распределение величин* $X_0, \ldots, X_t$.

Чтобы обращаться как с дискретными, так и с абсолютно непрерывными распределениями будем обозначать $\pi(dx)$ вероятностную меру, заданную на $(\mathcal{X}, \mathcal{B}_{\mathcal{X}})$. Для абсолютно непрерывной случайной величины X обозначим $f(x)$ её плотность — производную Радона-Никодима $\pi(dx)$ относительно меры Лебега. В дискретном случае $f(x)$ —

это производная Радона-Никодима относительно считающей меры. Обозначим $P_t(dx)$ — распределение X_t. Пусть задано начальное распределение по состояниям $P_0(dx)$. Также будем называть *переходным ядром* $P_t(x, dx)$ условное распределение X_{t+1} при условии $X_t = x$. Данные распределения связаны между собой соотношением

$$P_{t+1}(dy) = \int_{\mathcal{X}} P_t(dx)P_t(x, dy).$$

В рамках данной работы мы будем рассматривать только однородные по времени марковские цепи — такие, что переходное ядро не зависит от номера состояния: для любого $t > 0$ выполнено

$$P_t(x, dy) = P(x, dy).$$

Это означает, что марковская цепь «эволюционирует» по правилу

$$P_{t+1}(dy) = \int_{\mathcal{X}} P_t(dx)P(x, dy).$$

Таким образом, $P_t(dx)$ определяется единственным образом начальным распределением и переходным ядром $P(x, dy)$. Далее будем обозначать $P^n(x, \cdot)$ условное распределение X_n при условии $X_0 = x$.

Определение 2. *Инвариантное распределение марковской цепи — это такое распределение $\pi(dx)$, для которого выполняется*

$$\pi(dy) = \int_{\mathcal{X}} \pi(dx)P(x, dy), \tag{1.5}$$

где $P(x, dy)$ — ядро перехода. Также говорят, что в таком случае π и $P(x, dy)$ удовлетворяют условию баланса.

Например, если марковская цепь удовлетворяет условию $P_{t+1}(dx) = P_t(dx) = \pi(dx)$, то условие баланса будет выполнено. Тогда если $X_t \sim \pi$, то и X_{t+1} (вообще говоря, зависимая от X_t величина) будет иметь распределение π.

Определение 3. *Распределение $\pi(dx)$ называется стационарным для марковской цепи, если соотношение*

$$\lim_{t\to\infty} P(X_t \in A | X_0 = x) = \pi(A)$$

для π-почти всех x сохраняется для любого измеримого множества A.

Определение 4. *Будем называть марковскую цепь π-неприводимой, если для любого начального состояния и для любого $A \in \mathcal{B}_{\mathcal{X}}$ такого, что $\pi(A) > 0$, выполнено $P(X_t \in A) > 0$ для некоторого t.*

Определение 5. *π-Неприводимая марковская цепь называется π-периодической, если существует целое число $d \geq 2$ и последовательность $\{A_0, A_1, \ldots, A_{d-1}\}$ непустых множеств из $\mathcal{B}_{\mathcal{X}}$, таких что для всех $i = 0, \ldots, d-1$ и всех $x \in A_i$ выполнено*

$$P(x, A_j) = 1$$

для $j = i + 1 \mod d$. В противном случае, она называется апериодической.

Будем говорить, что события $\{A_i\}$ случаются бесконечно часто и обозначать это как $\{A_n$ б.ч.$\}$, если $P\left(\sum_i \mathbb{I}_{A_i} = \infty\right) = 1$.

Определение 6. *(a) Марковская цепь с инвариантным распределением π является рекуррентной, если для любого B, такого что $\pi(B) > 0$, выполняется условие*

$$P(X_n \in \ B \text{б.ч.} | X_0 = x) > 0$$

для всех x и

$$P(X_n \in B \ \text{б.ч.} | X_0 = x) = 1$$

для π-почти всех x.
(b) Марковская цепь с инвариантным распределением π является Харрис-рекуррентной, если для любого B, такого что $\pi(B) > 0$,

$$P(X_n \in B \ \text{б.ч.} | X_0 = x) = 1$$

для всех x.

Определение 7. *(a) Марковская цепь называется эргодической, если она является Харрис-рекуррентной и π-апериодической.*
(b) Марковская цепь с инвариантным распределением π обладает свойством геометрической эргодичности, если существует неотрицательная веществennозначная функция M, для которой выполняется $E[|M(X)|] < \infty$, и положительная константа $r < 1$ такая что

$$||P^n(x, \cdot) - \pi|| \leq M(x) r^n$$

для всех x.

(c) Марковская цепь (b) равномерно эргодическая, если существует константа M и $0 < r < 1$ такие что

$$||P^n(x, \cdot) - \pi|| \leq M r^n.$$

Здесь $||\cdot||$ *— расстояние по вариации:*

$$||\lambda|| = \sup_{A\in\mathcal{X}} \lambda(A) - \inf_{A\in\mathcal{X}} \lambda(A)$$

Определение 8. *Марковская цепь с переходным ядром* $P(x, dy)$ *и инвариантным распределением* $\pi(dx)$ *удовлетворяет уравнению детального баланса, если*

$$\int_B \int_A \pi(dx)P(x, dy) = \int_A \int_B \pi(dy)P(y, dx) \tag{1.6}$$

Для любых $A, B \in \mathcal{B}_{\mathcal{X}}$*. Такая цепь также называется реверсивной.*

Можно показать, что из условия детального баланса вытекает условие баланса (1.5).

Теорема 1 ([1]) *Пусть марковская цепь с переходным ядром является* π*-неприводимой и* π*-инвариантной. Тогда она является рекуррентной и* π *— её единственное инвариантное распределение. Если она при этом апериодическая, то для* π*-почти всех выполнено*

$$||P^n(x, \cdot) - \pi|| \xrightarrow{n\to+\infty} 0.$$

Другими словами, этот результат означает, что существует единственное стационарное распределение и оно совпадает с единственным инвариантным распределением. Также для марковских цепей есть аналог усиленного закона больших чисел. Справедлива следующая теорема.

Теорема 2 ([1]) *Пусть* X_n *эргодическая со стационарным распределением* π*, а* $h(x)$ *вещественнозначная и* $\mathbb{E}_\pi [|h(X)|] < +\infty$*. Тогда для любого начального распределения* $\bar{h}_n \to \mathbb{E}_\pi [h(X)]$ *почти наверное.*

Основная идея построения марковских цепей для приближения (1.1) состоит в том, чтобы построить $P(x, dy)$ с инвариантным распределением $\nu(dx)$. Как показано выше, при определённых условиях оно будет совпадать со стационарным распределением. При этом по Теореме 2 будет иметь место состоятельность интересующей нас оценки. Далее рассмотрим алгоритм, позволяющий построить такую марковскую цепь.

1.3.2. Алгоритм Метрополиса-Гастингса

Предположим, что нужно промоделировать выборку из распределения $\pi(dx)$. Идея состоит в построении марковской цепи с инвариантным распределением $\pi(dx)$. Остановимся на условии детального баланса (1.6). Пусть $f(x)$ — плотность $\pi(dx)$, а $p(x, y)$ — плотность $P(x, dy)$. Условие детального баланса тогда эквивалентно соотношению

$$f(x)p(x, y) = f(y)p(y, x).$$

Как мы упоминали ранее, из условия детального баланса вытекает условие баланса. Пусть X_t — состояние цепи на шаге t. Алгоритм Метрополиса-Гастингса позволяет построить марковскую цепь с инвариантным распределением $\pi(dx)$, удовлетворяющую условию детального баланса. Зададим плотность перехода $\gamma(y|x)$. Чтобы получить марковскую цепь, удовлетворяющую уравнению детального баланса, введём также вероятность принятия состояния, которое получается из $\gamma(\cdot|x)$, при имеющемся состоянии x. обозначим её $\alpha(x, y)$. Тогда получим равенство

$$p(x, y) = \gamma(y|x) \cdot \alpha(x, y).$$

При этом уравнение детального баланса имеет вид

$$f(x) \cdot \gamma(y|x) \cdot \alpha(x, y) = f(y) \cdot \gamma(x|y) \cdot \alpha(y, x). \tag{1.7}$$

Нетрудно видеть, что

$$\alpha(x, y) = \min\left\{1, \frac{f(y)\gamma(x|y)}{f(x)\gamma(y|x)}\right\}$$

является одним из решений данного соотношения. Теперь мы можем записать алгоритм моделирования марковской цепи с инвариантным распределением, имеющим плот-

ность $f(x)$.

Input: Состояние x_t, плотность f, переходная плотность γ

Output: Состояние x_{t+1}

Промоделировать $y \sim \gamma(y|x_t)$;

Вычислить $\alpha = \min\left\{1, \dfrac{f(y)\gamma(x_t|y)}{f(x_t)\gamma(y|x_t)}\right\}$;

Промоделировать $\beta \sim Unif(0,1)$;

if $\beta < \alpha$ **then**

  $X_{t+1} = Y$;

else

  $X_{t+1} = X_t$;

end

Алгоритм 1: Алгоритм Метрополиса-Гастингса

1.4. Алгоритм Ванга-Ландау

Пусть для каждого значения ξ задана некоторая функция $\mathrm{Score}(\xi)$ с дискретным множеством значений. Рассмотрим некоторое распределение Q, имеющее плотность вида

$$q(x) \propto w(\mathrm{Score}(x)) \cdot f(x),$$

где w — некоторая весовая функция. Заметим, что если $w(\mathrm{Score}(x)) \approx \dfrac{1}{P(\mathrm{Score}(\xi) = x)}$, то распределение $\mathrm{Score}(\xi)$ будет близко к равномерному на своём множестве значений. Как мы увидим далее, такое распределение является весьма полезным для решаемой задачи и позволяет эффективно моделировать редкие события.

В качестве способа оценить весовую функцию рассмотрим алгоритм Ванга-Ландау [2]. Он является адаптивным и позволяет построить плотность, для которое распределение Score близко к равномерному. В начале работы алгоритма веса для всех значений Score одинаковые. Потом при помощи Алгоритма Метрополиса-Гастингса моделируется следующее состояние, вычисляется его Score, и вес данного состояния уменьшается, после чего дальнейшие состояния моделируются уже с учётом изменений. Процесс продолжается, пока гистограмма повстречавшихся значений не будет близка к плоской (то есть распределение значений Score близко к равномерному). Далее LC — логарифм константы уменьшения веса, LW — логарифмические значения весов, $H(i)$ количество встретившихся состояний цепи, для которых $\mathrm{Score} = i$. Запишем данную схему в виде

Алгоритма 2. Заметим что при такой схеме моделирования, вообще говоря, свойство эргодичности не гарантируется, поэтому использовать траекторию, полученную в процессе прохода алгоритма Ванга-Ландау для оценки Монте-Карло по схеме марковской цепи нельзя. Поэтому в дальнейшем мы будем сначала оценивать весовую функцию данным алгоритмом, а потом моделировать цепь с нужным нам инвариантным распределением по Алгоритму 1.

Input: Множество состояний Score(), число итераций K_{max}, начальное состояние X

Output: Логарифмированные значения весовой функции LW

$LC = 1$;
while $K < K_{max}$ **do**
  $H(i) = 0$ для всех состояний Score ;
  while H *не «плоская»* **do**
    Получить следующее состояние x^* марковской цепи для $q(x) = f(x)\exp(LW(\text{Score}(x)))$;
    $LW(\text{Score}(x^*)) = LW(\text{Score}(x^*)) - LC$;
    $H(\text{Score}(x^*)) = H(\text{Score}(x^*)) + 1$;
  end
  $LC = LC/2$;
  $K = K + 1$;
end

Алгоритм 2: Алгоритм Ванга-Ландау

1.5. Оценка дисперсии

Кроме самих оценок, нас также интересует их дисперсия. В этом разделе мы приведём теоретические факты, касающиеся дисперсии оценок, а также способам её оценить. Дисперсия оценки (1.3) имеет вид

$$\sigma^2_{MC} = p(1-p). \tag{1.8}$$

Далее речь пойдет об оценке вероятности некоторого события методом Монте-Карло по схеме марковской цепи. Справедлива следующая теорема.

Теорема 3 ([1]) *Пусть X_n — равномерно эргодическая марковская цепь со стационарным распределением $\pi(dx)$ с плотностью $f(x)$, $h(x)$ — веществennозначная функция*

и $E_f[h^2(\xi)] < \infty$. *Тогда существует* $\sigma_h \in \mathbb{R}$ *такое, что*

$$\sqrt{n}(\bar{h}_n - E_f[h(\xi)]) \Rightarrow N(0, \sigma_h^2),$$

где $\bar{h}_n = \dfrac{1}{n}\sum_{i=1}^{n} h(X_i)$,

$$\sigma_h^2 = \mathbb{D}_f(h(X_1)) + 2\sum_{n=2}^{\infty} \mathrm{cov}_f\left[h(X_1), h(X_n)\right].$$

В статье [3] авторы предлагают следующий способ оценки σ_h^2. Пусть у нас есть траектория длины n и скользящее окно размера b_n. Вычислим оценку (1.2) для каждого положения окна: $\bar{h}_j(b_n) = b_n^{-1}\sum_{i=1}^{b_n} X_{i+j}$, где $j = 0, \ldots, n - b_n$. Тогда оценкой по методу перекрывающихся средних (overlapping batch means) называется величина

$$\hat{\sigma}^2_{OBM} = \frac{nb_n}{(n-b_n)(n-b_n+1)} \sum_{j=0}^{n-b_n} (\bar{h}_j(b_n) - \bar{h}_n)^2 \tag{1.9}$$

Для оценки (1.9) справедлива следующая теорема.

Теорема 4 ([3]) *Пусть марковская цепь* X *обладает свойством геометрической эргодичности и имеет инвариантное распределение* $\pi(dx)$ *с плотностью* $f(x)$, *а* h *является борелевской веществennозначной функцией. При этом* $E_\pi[|h|^{2+\delta+\epsilon}] < \infty$ *для некоторых* $\epsilon > 0$ *и* $\delta > 0$. *Если* $b_n = [n^\nu]$, *где* $3/4 > \nu > (1+\delta/2)^{-1}$. *Тогда оценка* (1.9) *является сильно состоятельной.*

Глава 2

Оценка вероятности максимального совпадения строк

Теперь рассмотрим конкретную задачу вычисления малой вероятности. Стоит отметить, что для данного примера истинное значение известно заранее.

Имеется конечный алфавит Σ размера n. Рассмотрим множество строк длины k над этим алфавитом. Зафиксируем строку $S^* \in \Sigma^k$. Пусть на нашем множестве задано распределение $P(dx)$. В рамках данной задачи распределение P положим равномерным. Будем оценивать вероятность максимального совпадения строк:

$$p = P(\text{Score}(S^*, S) = d_{max}),$$

где $\text{Score}(S^*, S) = k - d(S^*, S)$, а d — некоторая метрика, S — случайная строка. В качестве метрики возьмём расстояние Хэмминга.

Нетрудно видеть, что $p = P(d(S^*, S) = 0) = \dfrac{1}{n^k}$. Мы оценим данную вероятность двумя способами: методом Монте-Карло по независимой выборке и при помощи существенной выборки, промоделировав марковскую цепь с инвариантным распределением, имеющим плотность, оценённую при помощи алгоритма Ванга-Ландау.

2.1. Построение оценок

Пусть $n = 4$, а $k = 10$. Тогда $p \approx 9.5 \cdot 10^{-7}$. Оценка по независимой выборке имеет вид

$$\hat{p}_{MC} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}_A(S_i), \tag{2.1}$$

где A — множество строк, на которых достигается максимальное совпадение с S^*.

2.1.1. Оценка при помощи существенной выборки

Рассмотрим распределение Q с плотностью

$$q(S) = c \cdot w(\text{Score}(S)), \tag{2.2}$$

где c — некоторая нормирующая константа. Для начала мы получим оценку весовой функции (2.2) при помощи Алгоритма 2, и, соответственно, $\hat{Q}$. Затем промоделируем

при помощи Алгоритма 1 марковскую цепь с инвариантным распределением $\hat{Q}$. Заметим, что для моделирования состояний цепи нет необходимости знать константу c. Вероятность принять состояние Y выглядит так:

$$\alpha(X_t, Y) = \min\left\{1, \frac{q(Y)}{q(X_t)}\right\}.$$

По имеющейся траектории оценка будет выглядеть следующим образом:

$$\hat{p}_{IS} = \frac{1}{cn}\sum_{i=1}^{n} \mathbb{I}_A(S_i)/w(\text{Score}(S_i)).$$

Заменим теперь c на состоятельную оценку и получим

$$\hat{p}_{IS} = \frac{\sum_{i=1}^{n} \mathbb{I}_A(S_i)/w(\text{Score}(S_i))}{\sum_{i=1}^{n} 1/w(\text{Score}(S_i))}. \tag{2.3}$$

Для работы этого метода нужно выбрать переходную плотность γ. При имеющейся строке Y выберем в ней случайную позицию (вероятности для всех позиций равны), и на место этой позиции запишем случайный равновероятно выбранный символ из алфавита. По полученной траектории вычислим оценку (2.3). Схема моделирования траектории приведена в виде Алгоритма 3.

Input: Начальное состояние S_0, длина траектории N, число итераций для алгоритма Ванга-Ландау K, множество состояний Score, переходная плотность γ

Output: траектория $\{S_n\}$

```
1 q̂ = exp(Wang-Landau(K, Score, γ)) ;
2 i = 1 ;
3 while i < N do
4 |   S_{i+1} = MH-Step(S_i, q̂, γ) ;
5 |   i = i + 1 ;
6 end
```

Алгоритм 3: Алгоритм моделирования траектории для оценки по методу существенной выборки

Нетрудно проверить, что для такой переходной плотности будет выполняться уравнение детального баланса. Действительно, в нашем случае $\gamma(x|y) = \gamma(y|x)$ и уравнение (1.7) можно сократить, после чего останется верное равенство. Также вычислим дисперсии оценок. Дисперсия оценки (2.1) имеет вид (1.8). Обратим внимание на то, что оценка (2.3) представляет из себя частное двух случайных величин. Сформулируем утверждение, известное как дельта-метод.

Теорема 5 (Дельта-метод, [4]) *Пусть выполнено*

$$\sqrt{n}(\bar{\xi}_n - \bar{a}) \implies N(\boldsymbol{0}, \Sigma)$$

для некоторой последовательности случайных величин. Если f — гладкое отображение, то тогда будет верно соотношение

$$\sqrt{n}(f(\bar{\xi}_n) - f(\bar{a})) \implies N(\boldsymbol{0}, \nabla_f^{\mathrm{T}}(\bar{a})\Sigma\nabla_f(\bar{a})).$$

Применительно к оценке (2.3), функция $f : \mathbb{R}^2 \to \mathbb{R}$ и имеет вид $f(x, y) = x/y$. Также величины в числителе и знаменателе являются асимптотически нормальными. Для того чтобы оценить дисперсию, нужно вычислить оценку ковариационной матрицы. Для этого применим оценку (1.9) отдельно для оценки дисперсии числителя $x = \sum_{i=1}^{n} \mathbb{I}_A(S_i)/w(\mathrm{Score}(S_i))$ и знаменателя $y = \sum_{i=1}^{n} 1/w(\mathrm{Score}(S_i))$ по имеющейся траектории. Для оценки $\mathrm{cov}(x, y)$ воспользуемся видоизмененной оценкой (1.9): вместо суммы квадратов по одному batch'у будем вычислять сумму произведений для числителя и для знаменателя. Дисперсию оценки (2.3) также можно оценить, запустив несколько независимых траекторий и вычислив выборочную дисперсию.

2.1.2. Параллельное темперирование

Рассмотрим ещё один способ вычислить оценку при помощи существенной выборки. Пусть интересующее нас распределение Q задано следующим образом: $q(x) \propto \exp(H(x))$. Также пусть есть n марковских цепей $X^{(1)}, \ldots, X^{(n)}$, не зависящих друг от друга. При этом цепь $X^{(i)}$ имеет инвариантное распределение $f_i(x) \propto \exp(\beta_i H(x))$. Из вида плотности видно, что цепи, соответствующие большим значениям β_i, больше «застревают» в больших значениях функции $H(x)$. Поэтому возникает идея иногда обменивать состояния разных цепей, например, чтобы уменьшать дисперсию оценки по марковской цепи, соответствующей β_1, но так, чтобы совместное распределение сохранялось, как если бы обмена не было. Метод параллельного темперирования состоит в том, что цепи моделируются методом Метрополиса-Гастингса, при этом на каждом шаге происходит попытка обменять состояния у двух случайно выбранных цепей. Подробнее

метод изложен в [1].

Input: Состояния $(X_t^{(1)}, \ldots, X_t^{(n)})$, значения $\beta_1, \ldots, \beta_n$

Output: Состояния $X_{t+1}^{(1)}, \ldots, X_{t+1}^{(n)}$

1 Для каждого $X_t^{(i)}$ выполнить шаг Метрополиса-Гастингса ;

2 Выбрать случайную пару индексов (i, j) ;

3 вычислить $\alpha = \min\left\{1, \exp(\left[H(X_{t+1}^{(i)}) - H(X_{t+1}^{(j)})\right][\beta_j - \beta_i])\right\}$

4 Промоделировать $\gamma \sim Unif(0, 1)$;

5 **if** $\gamma < \alpha$ **then**

6 $\quad Swap(X_{t+1}^{(j)}, X_{t+1}^{(i)})$;

7 **else**

8 **end**

Алгоритм 4: Параллельное темперирование

2.1.3. Численные результаты

Все используемые алгоритмы были реализованы на языке R. Зафиксируем объем выборки (а также длину траектории марковской цепи) $N = 10^7$. Далее приведём значения оценок. Здесь $\hat{p}_{MC}$ — оценка по методу Монте-Карло (1.3) по независимой выборке из равномерного распределения, $\hat{p}_{IS}$ — оценка (2.3) по траектории, полученной при помощи Алгоритма 3, $\hat{p}_{PT}$ — оценка при помощи параллельного темперирования (Алгоритм 4), σ^2_{MC} — значение (1.8), $\hat{\sigma}^2_{IS-MS}$ — оценка дисперсии по нескольким независимым траекториям, $\hat{\sigma}^2_{PT-OBM}$ — оценка дисперсии при помощи OBM для параллельного темперирования, $\hat{\sigma}^2_{IS-OBM}$ — оценка дисперсии при помощи OBM для $\hat{p}_{IS}$.

- $\hat{p}_{MC} \approx 10^{-6}$
- $\hat{p}_{IS} \approx 9.5 \cdot 10^{-7}$
- $\hat{p}_{PT} \approx 9.57 \cdot 10^{-7}$
- $\sigma^2_{MC} \approx 9.5 \cdot 10^{-7}$
- $\hat{\sigma}^2_{IS-MS} \approx 2.6 \cdot 10^{-9}$
- $\hat{\sigma}^2_{IS-OBM} \approx 1.85 \cdot 10^{-9}$
- $\hat{\sigma}^2_{PT-OBM} \approx 2.97 \cdot 10^{-9}$

Видно, что дисперсия оценки по методу существенной выборки меньше, чем дисперсия оценки по независимой выборки напрямую из равномерного распределения.

Теперь рассмотрим асимптотическое поведение оценки (2.3) с плотностью, оценённой при помощи Алгоритма 2. Для начала построим такие оценки для траекторий различной длины. Зафиксируем длину строки $k = 10$. на рис. 2.1 изображено поведение среднеквадратического отклонения в зависимости от длины траектории. На рисунке 2.2 изображены оценки дисперсии по методу OBM $\hat{\sigma}^2_{OBM-IS}$ и по нескольким независимым траекториям $\hat{\sigma}^2_{MS}$. Видно, что при большой длине траектории оценки ведут себя одинаково.

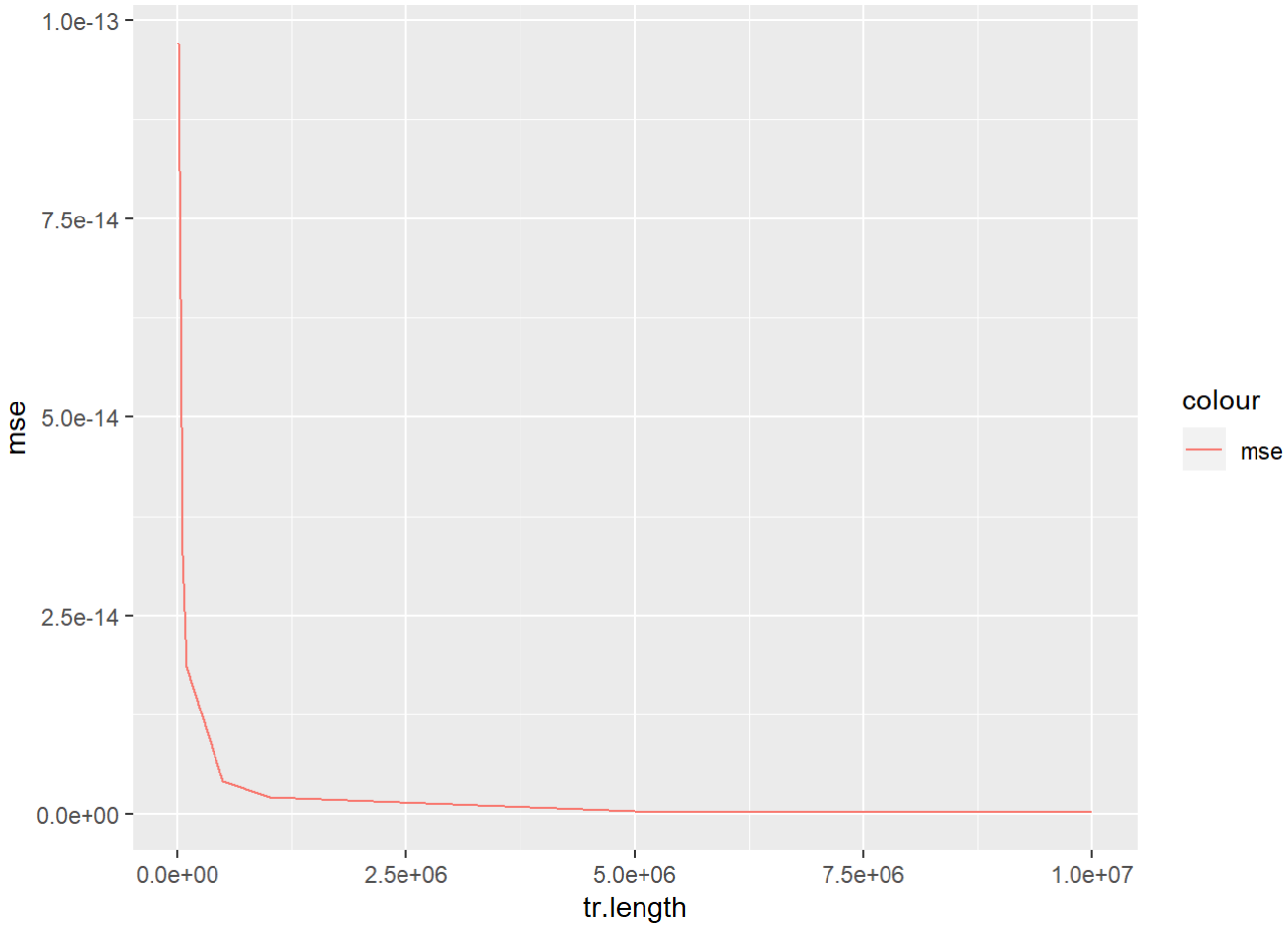

Рис. 2.1. Среднеквадратическое отклонение при различных длинах траектории

Теперь пусть $N = 10^6$, а $k = 4, \ldots, 40$, где k— длина строки. Для каждого значения k промоделируем 30 траекторий и вычислим относительные (поделённые на истинное значение вероятности) смещение, среднеквадратическое отклонение и дисперсию. Результаты изображены на Рис. 2.3. Видно, что среднеквадратическое отклонение растёт

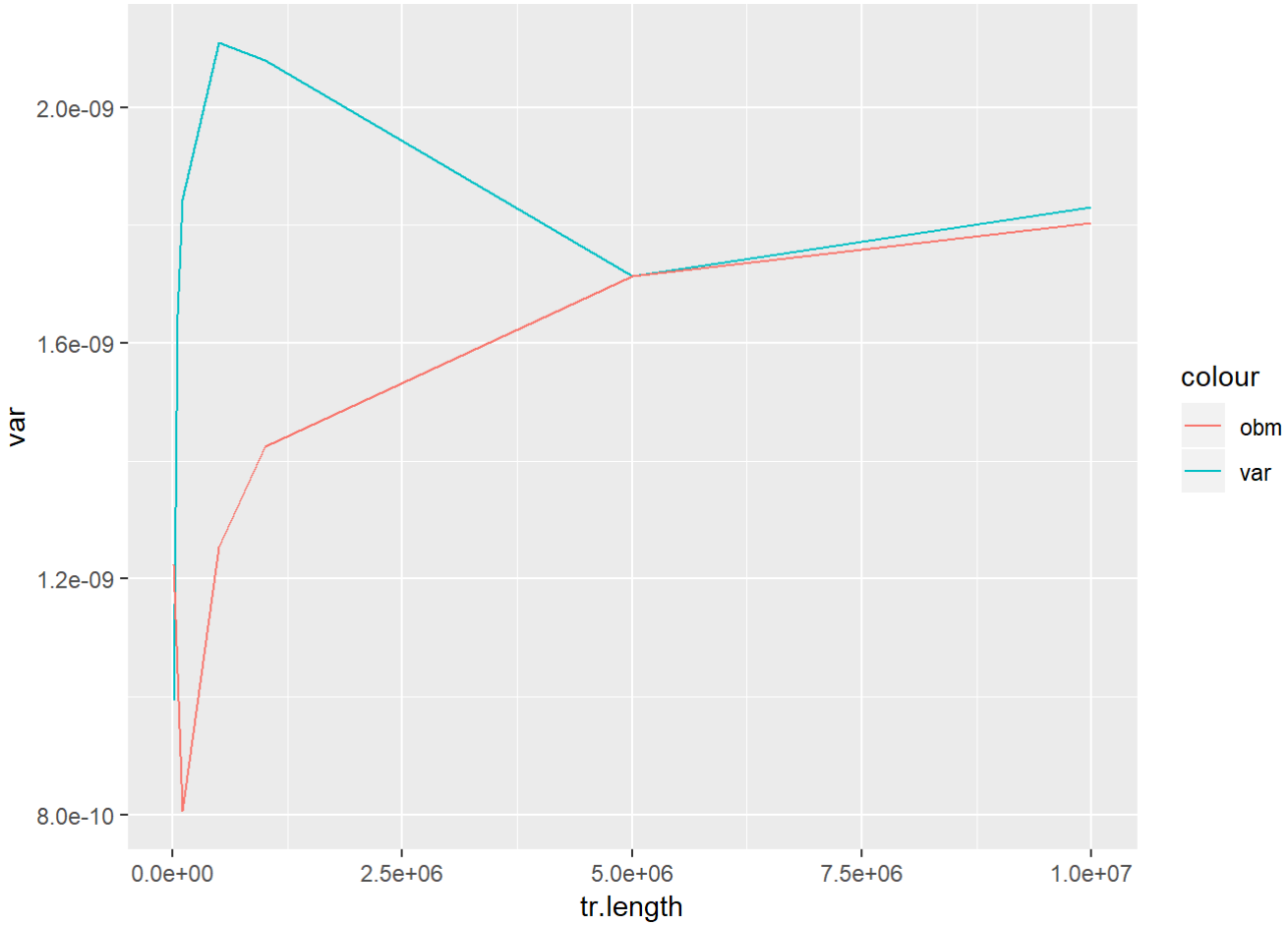

Рис. 2.2. Среднеквадратическое отклонение при различных длинах траектории

при уменьшении значения вероятности, которую мы хотим оценивать.

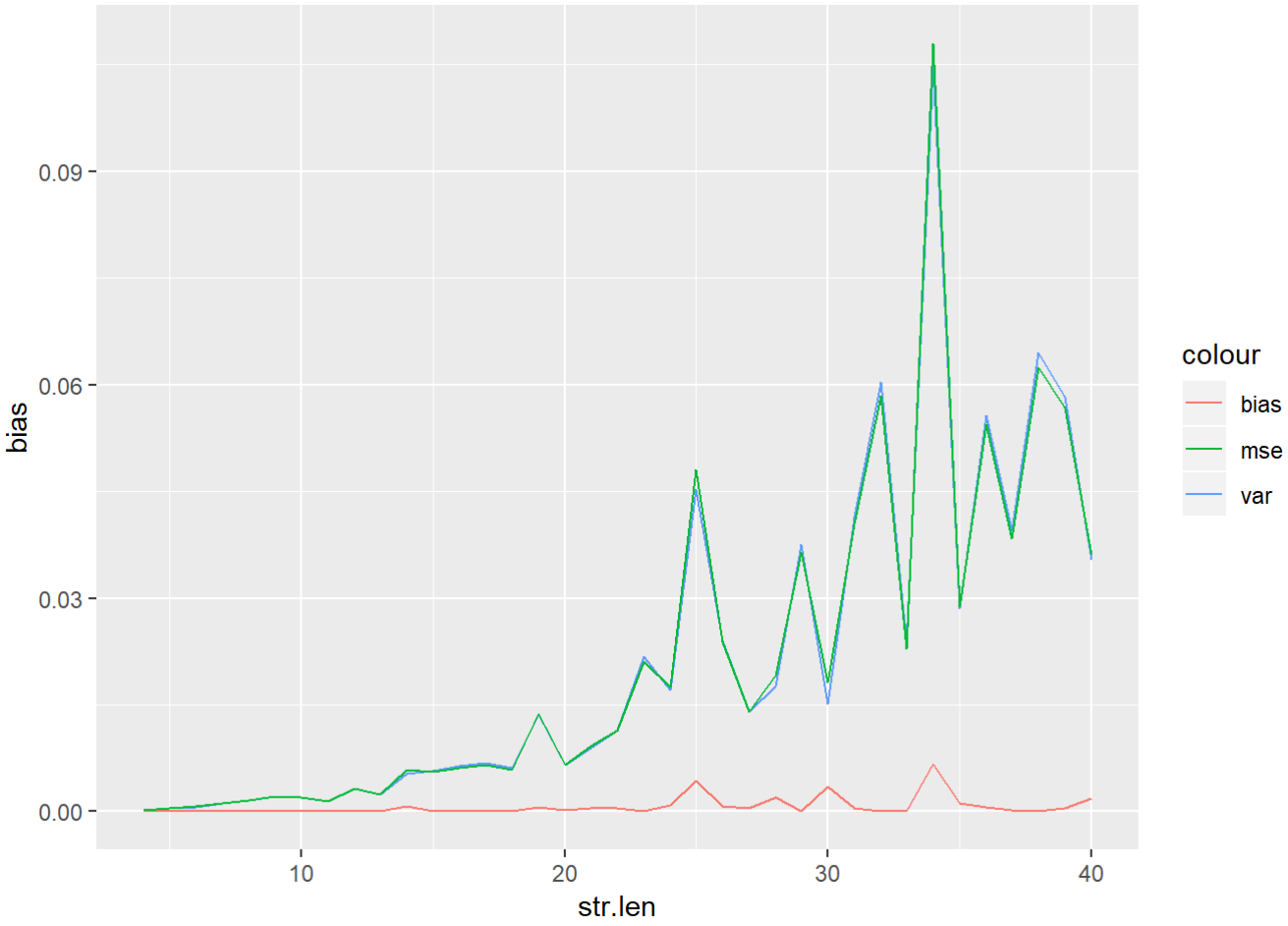

Рис. 2.3. Среднеквадратическое отклонение при различных длинах траектории

Глава 3

Разработка библиотеки для оценивания малых вероятностей

Напомним, что основной целью данной работы является разработка библиотеки для вычисления вероятностей редких событий. Данная глава посвящена разработке данного фреймворка. Главная проблема состоит в том, что мы хотим избежать хранения пространства состояний в каком-либо виде. Например, как мы видели в рассмотренной в Главе 2 задаче, пространство состояний растёт экспоненциально с ростом длины строки. На самом деле нам и не нужно его хранить полностью. Нужно только понимать, как из одного состояния моделировать следующее состояние, то есть нужны распределение γ и функция Score. Это позволит в полной мере воспользоваться алгоритмом Ванга-Ландау и оценкой по существенной выборке.

3.1. Обзор существующих библиотек

Важным инструментом оценивания малых вероятностей является моделирование марковской цепи с заданным стационарным распределением. Здесь мы рассмотрим существующие решения с открытым исходным кодом в области моделирования марковских цепей, чтобы понять, какие из них можно использовать для разработки.

3.1.1. Stan

Stan — язык, разработанный для вероятностного программирования (см. [5]). В основе языка лежит Stan Core Library — библиотека на языке C++, предоставляющая следующий интерфейс:

- полный байесовский вывод с использованием No-U-Turn sampler (NUTS), варианта Гамильтонова Монте-Карло (см. [6]);
- приближенный байесовский вывод с использованием вариационного вывода;
- оценки максимального правдоподобия.

Stan Core Library реализована на основе Stan Math Library, в которой реализованы полностью шаблонизированная матричная алгебра, линейная алгебра и библиотека вероятностных функций.

Нас в первую очередь интересует моделирование марковских цепей, которое в Stan реализовано с помощью NUTS. Обычно такой способ моделирования требует непрерывность пространства состояний. Стоит отметить, что есть способ использовать Гамильтоново Монте-Карло для моделирования дискретных случайных величин (см. [7]). Но для такого метода нужно хранить все пространство состояний целиком, что для нас неприемлемо. Поэтому Stan не очень подходит в качестве вспомогательного инструмента. Также интерфейс библиотеки доступен в R и Python в виде соответствующих пакетов RStan и PyStan.

3.1.2. PyMC3

PyMC3 — библиотека на языке Python, предоставляющая интерфейс байесовских методов, преимущественно с использованием марковских цепей (см. [8]). Основные элементы API:

- распределения. Есть возможность работать с дискретными, непрерывными, многомерными распределениями, а также задавать пользовательские распределения;
- моделирование. В библиотеке реализованы различные варианты моделирования марковских цепей: NUTS, алгоритм Метрополиса-Гастингса, метод Гиббса и другие шаговые методы;
- диагностика сходимости марковской цепи на основе траектории;
- построение различных графиков.

Низкоуровневая часть PyMC3 реализована на основе Theano. PyMC3 позволяет задавать полные вероятностные модели, содержащие различные случайные величины и их взаимодействие. В целом, PyMC3 является гибким инструментом для моделирования марковских цепей и их диагностики. За счёт возможности расширять имеющиеся распределения данная библиотека является наиболее подходящей к рассмотренной в Главе 2 задаче. При этом отметим, что в PyMC3 нет готовых инструментов для работы с марковскими цепями (определенных на дискретном пространстве состояний) в

терминах функции Score. Как мы упоминали выше, это существенный момент. Сейчас в PyMC3 нет полностью готовых функций, из которых можно скомпоновать наше решение. Однако данная библиотека пригодна для расширения.

3.2. Детали реализации

Библиотеку в целом можно разделить на три основных части:

- пользовательский интерфейс;
- инструменты для моделирования траектории марковской цепи;
- инструменты для вычисления оценок

Здесь мы рассмотрим эти компоненты и опишем их реализацию.

3.2.1. Пользовательская часть

Сначала опишем компоненты, которые должен задать пользователь:

- распределение $\gamma(y|x)$ в виде некоторой функции `proposal(`x`)`, которая моделирует потенциально следующее состояние марковской цепи при текущем состоянии x.
- целевое распределение P случайного объекта, для которого мы хотим вычислить вероятность попасть в некоторое множество, например, в виде функции плотности;
- функция, вычисляющая $\mathrm{Score}(x)$ для состояния x, так как в основе метода Ванга-Ландау для оценки распределения Q существенной выборки лежит возможность вычислять значение этой функции. Более того, для марковской цепи, в которой на множестве состояний определён Score, сама задача оценивания малой вероятности формулируется в терминах Score.
- начальное состояние марковской цепи x_0.

Кроме того, состояние должно иметь вид NumPy-массива. Это необходимо для корректного использования средств PyMC3 для моделирования.

Далее опишем подробности реализации: расширение PyMC3 и также реализацию остальных компонентов для решения задачи оценивания вероятностей редких событий.

Расширенная версия PyMC3 представляет собой некоторое ядро, на основе которого реализованы методы Монте-Карло по схеме марковской цепи.

3.2.2. Расширение возможностей PyMC3

Для начала заметим, что PyMC3 работает с распределениями, сосредоточенными на числовых носителях (даже дискретные распределения работают только с множеством целых чисел). В нашем же случае это неприемлемо: задать отображение множества состояний произвольного случайного объекта сводится либо к заданию некой функции, определённой на множестве случайных объектов, либо к кодированию целыми числами каждого объекта. Первый вариант возможен, вообще говоря, не всегда, а второй означает, что мы должны хранить пространство состояний целиком. Поэтому мы решили расширить возможности PyMC3, чтобы получить инструменты, подходящие для решения поставленной задачи.

Центральным звеном PyMC3 является класс `Model`, который представляет интерфейс для определения вероятностной модели посредством добавления в неё случайных величин. Причем работа с этим классом со стороны пользователя предполагает использование контекстного менеджера в Python, с использованием синтаксической конструкции `with` (см [9]). В контексте вероятностной модели, определяемой таким образом, происходят все действия со случайными величинами: моделирование выборки, байесовский вывод и так далее.

Другим важным классом объектом для PyMC3 является распределение, реализованное в виде класса `Distribution` и его наследников. Каждая случайная величина добавляется в модель в виде вызова конструктора класса её распределения. Интерфейс `Distribution` достаточно просто устроен и требует лишь задать функцию, возвращающую логарифм плотности в точке.

Theano в PyMC3

Теперь рассмотрим подробнее детали вероятностной модели в PyMC3. Случайная величина в оригинальной реализации представляет собой `TensorVariable` из Theano. Theano — это библиотека для символьных вычислений ([10]). В рамках Theano из переменных и действий между ними составляется граф вычислений. Когда пользователю нужен результат, граф оптимизируется, компилируется в вызываемую функцию, в ко-

торую можно подставить значения входов и получить результат. Примерно то же самое происходит со случайными величинами в PyMC3: по мере добавления переменных в модель составляется граф для вычисления логарифма совместной плотности, и в момент, когда нужно получать конкретные значения плотности (например, при моделировании марковской цепи алгоритмом Метрополиса-Гастингса), компилируется функция, которая используется до тех пор, пока сама модель не меняется.

Реализованные расширения PyMC3

В нашей версии реализован класс `FreeCatRV`, представляющий собой произвольный случайный объект, а также возможность добавлять такие объекты в вероятностную модель. Данный класс наследуется от `Variable` из Theano, что позволяет добавлять произвольные Generic-объекты и операции над ними в граф вычислений. В Theano уже реализована возможность задавать произвольную операцию над переменными при помощи функции `FromFunctionOp(fn)`, где `fn` — Python-функция. Таким образом реализовано вычисление Score() от случайного объекта. Далее, плотность распределения, нужного для оценки по методу существенной выборки, имеет вид $P(\text{Score}(X))$, а после вычисления Score() мы получаем число, поэтому дальнейшее вычисление $\log(P)$ сводится к уже тензорным операциям и легко встраивается в уже имеющиеся в PyMC3 операции. На листинге 3.1 показано, как при вычислении логарифма плотности используется определение операции Python-функцией.

```
from theano.compile.ops import FromFunctionOp
import theano.tensor as tt
...
def logp(self, value):
  ...
  #value is theano Variable
  #this is custom operation and can be applied to theano Variable
  scoreOp = FromFunctionOp(fn=self.get_score,
              itypes=[theano.gof.generic],
              otypes=[tt.lscalar],
              infer_shape=None)
  scoret = scoreOp(value) #this is theano tensor

  return ...
```

```
15
```

Листинг 3.1. Определение вычисления Score() с помощью `FromFunctionOp`

Также реализован класс `WeightedScoreDistribution`, представляющий распределение, имеющее плотность вида (2.2). Он наследуется от `Distribution` из PyMC3 и при помощи конструктора такого распределения мы можем добавить в вероятностную модель наш произвольный случайный объект. Чтобы задать такое распределение, нужно задать веса и w(Score) и Python-реализацию вычисления Score состояния. В листинге 3.2 продемонстрированы основные элементы этого класса.

```
1  class WeightedScoreDistribution(Distribution):
2    def __init__(self, scorer, weighting, pfun=None, ...):
3      ...
4      self.scorer = scorer #this is a function of Generic object
5      self.weights = theano.shared(weighting)
6      self.p = pfun
7      ...
8    def pfun(self, value):
9      if self.p is None:
10       return np.array(1).astype('float64')
11     else:
12       return np.array(self.p(value)).astype('float64')
13   def get_score(self, value):
14     return np.array(int(self.scorer(value)))
15
16   def logp(self, value):
17     ...
```

Листинг 3.2. Класс `WeightedScoreDistribution`

Далее, для возможности моделирования реализован класс `GenericCatMetropolis`, принимающий на вход произвольную функцию `proposal()`, которая моделирует следующее значение и представляет собой переходную плотность. Класс `GenericCatMetropolis` позволяет использовать алгоритм Метрополиса-Гастингса для моделирования марковской цепи со стационарным распределением (2.2) при помощи функции `sample()`, уже реализованной в PyMC3. Процесс получения траектории продемонстрирован в листинге 3.3.

```
1  ...
```

```
with pm.Model() as model:
  s = pm.WeightedScoreDistribution('S', #variable name
                  scorer=..., #custom score
                  weighting=...), #list of weights
                  cat=True, #distribution is non-numeric
                  default_val=...)
  trace = pm.sample(draws=100000, cores=1,
            start={'S':...}, #initial state
            step=pm.GenericCatMetropolis(vars=[s], #step-method
                   proposal=...), #custom proposal
            compute_convergence_checks=False,
            chains=1, wl_weights=True) #wang-landau is on
...
```

Листинг 3.3. Моделирование траектории марковской цепи

Также реализована функция `wang_landau()`, которая позволяет оценить весовую функцию $w(\text{Score})$ алгоритмом 2, и соответственно плотность, из которой нужно моделировать, непосредственно перед процессом моделирования. В функции `wang_landau()` используется реализованный ранее класс `GenericCatMetropolis`.

3.2.3. Модуль для оценки вероятностей

Теперь у нас есть инструмент для моделирования траектории марковской цепи. Здесь мы опишем реализацию модуля для оценок вероятностей $P(\text{Score}(X) \in A)$. Центральным объектом является класс `ProbabilityEstimator`. Конструктор класса принимает следующие параметры:

- `p`: функция плотности, определяющая изначальное распределение случайной величины;
- `scorefun`: функция, возвращающая Score состояния;
- `proposal`: плотность распределения $\gamma(\cdot|x)$ — функция, возвращающая потенциально следующее состояние;
- `default_val`: состояние по умолчанию, используется в качестве стартового состояния марковской цепи. Ожидается, что начальное значение будет в виде NumPy-массива.

- `save_trace`: флаг, означающий, сохранять траекторию из Score состояний или нет;
- `initial_weights`: начальные веса для целевой плотности распределения;
- `n_samples`: длина моделируемой траектории.

Данный класс предоставляет следующий интерфейс:

- `estimate_between(left, right)`: вычислить вероятность $P(\text{Score}(\xi) \in [left, right))$
- `estimate_boolean(bool_func)`: вычислить вероятность $P(\text{Score}(\xi) in A)$, где принадлежность множеству A задается логической функцией `bool_func`. В результате выполнения этой и предыдущей функций также вычисляется оценка дисперсии при помощи Overlapping Batch Means;
- `confint(gamma)`: возвращает доверительный интервал уровня `gamma`.

Также в классе есть внутренняя функция `__get_score_trace()`, при помощи которой происходит моделирование марковской цепи. Данная функция использует результаты, описанные в предыдущем разделе. Объект класса `ProbabilityEstimator` при создании сохраняет параметры задачи, которые задает пользователь, далее, при вызове, например, метода `estimate_between(left, right)` моделируется марковская цепь, оценивается нужная вероятность, а также оценивается дисперсия этой оценки.

Таким образом, получена реализация всего процесса решения задачи, который представлен в алгоритме 3. Исходный код данного модуля доступен на Zenodo ([11]). В приложении А рассмотрен пример из главы 2 вычисления вероятности с использованием всех реализованных ранее компонент.

Заключение

В работе выполнена декомпозиция исходной задачи на подзадачи, приведена необходимая теория для их решения, а также на примере конкретной задачи продемонстрированы численные результаты. Показано, что при использовании оценки по методу существенной выборки можно добиться уменьшения дисперсии оценки. Также схожий результат показывает параллельное темперирование. Также численно исследовано и продемонстрировано асимптотическое поведение оценок, как в зависимости от длины траектории, так и от оцениваемой вероятности.

Также проведён небольшой обзор существующих и пользующихся популярностью фреймворков в области моделирования марковских цепей — Stan и PyMC3 и приведены общие идеи реализации библиотеки для оценивания малых вероятностей путём адаптации инструментария PyMC3 под возможность использовать метод Ванга-Ландау.

После этого были реализованы основные компоненты, из которых состоит библиотека: инструмены моделирования марковской цепи на произвольном дискретном пространстве состояний, а также средства оценивания вероятности по такой траектории. Кроме того, разработан пользовательский интерфейс, предоставляющий доступ к этим частям. После этого был рассмотрен пример использования данной библиотеки.

Приложение А

Пример вычисления малой вероятности

Здесь продемонстрируем работу библиотеки применительно к примеру, рассмотренному в главе 2, а именно, вычисление вероятности полного совпадения строк по расстоянию Хэмминга.

Для начала импортируем нужные модули:

```
from small_probs.probability import ProbabilityEstimator
from scipy.spatial.distance import hamming
import numpy as np
from scipy.stats import norm
```

Зададим параметры задачи в качестве пользовательских компонент:

```
class string2:
    def __init__(self, length):
    self.n_letters = length
    self.state_fixed = np.array(["A"] * self.n_letters)
    alphabet = frozenset("ATGC")
    self.letters_list = list(alphabet)
    self.proposed = 0
    self.p_l = np.random.choice(a=self.letters_list,
                                size=100000)
    self.propose_positions = np.random.choice(a=self.n_letters,
                                size=100000)

  def score(self, state):
    return self.n_letters - np.sum(state != self.state_fixed)

  def proposal(self, state):
    if (self.proposed == 100000):
      self.p_l = np.random.choice(a=self.letters_list,
                                size=100000)
      self.propose_positions = np.random.choice(a=self.n_letters,
                                size=100000)
      self.proposed = 0
    state[self.propose_positions[self.proposed]] = self.p_l[self.proposed]
    self.proposed += 1
```

```
25      return state
```

Промоделируем марковскую цепь и вычислим оценку:

```
s2 = string2(10)
ps = ProbabilityEstimator(p=None, scorefun=s2.score, proposal=s2.proposal,
                    default_val=s2.state_fixed,
                    initial_weights=np.array([2] * 11), save_trace=True)
ps.estimate_between(10, 11)
```

Выведем получившуюся оценку и доверительный интервал уровня 0.95

```
print("True value: ", 1/4**10)
print("Probability estimation: ", ps.prob)
print("OBM variance estimation: ", ps.var)
print(gamma, " confidence interval for true probability ",
    ps.confint(0.95))
```

Вывод предыдущих строк:

```
True value:  9.5367431640625e-07
Probability estimation:  9.735235035741608e-07
OBM variance estimation:  1.8743340848133425e-09
0.95  confidence interval for true probability
(8.886696456593089e-07, 1.0583773614890127e-06)
```

Также можно сохранить траекторию, и, например вывести график Score состояний.

```
  trace = ps.score_trace
  plt.figure(figsize=(35,10))
  plt.plot(trace[-15000:])
```

Траектория изображена на Рисунке А.1

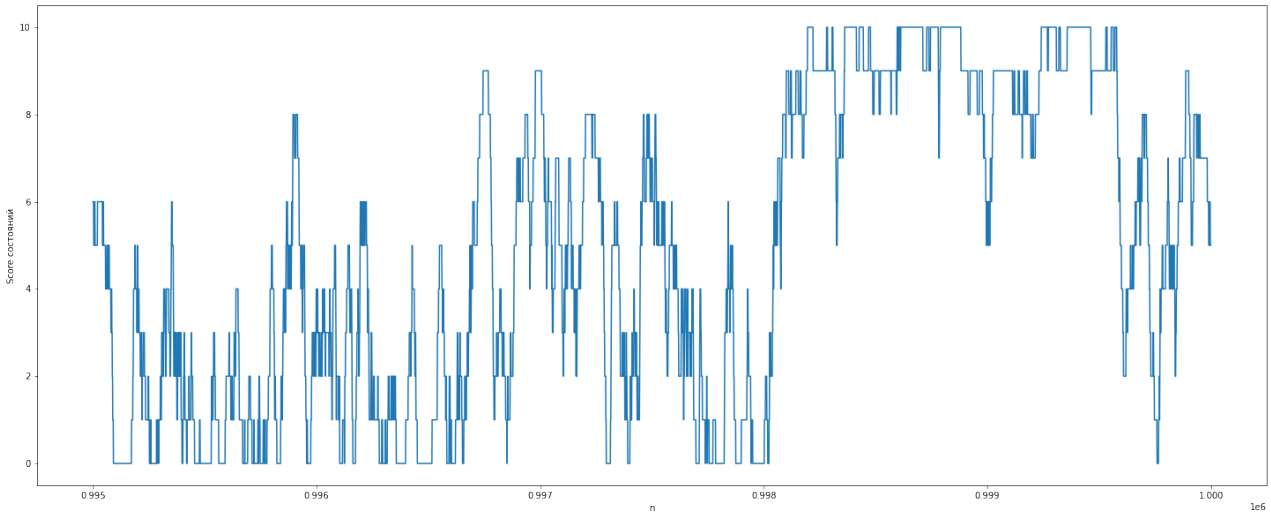

Рис. А.1. Траектория из Score состояний

Список литературы

1. Liang Faming, Liu Chuanhai, Carroll Raymond. Advanced Markov chain Monte Carlo methods: learning from past samples. — John Wiley & Sons, 2011. — Vol. 714.
2. Iba Yukito, Saito Nen, Kitajima Akimasa. Multicanonical MCMC for sampling rare events: an illustrative review // Annals of the Institute of Statistical Mathematics. — 2014. — Vol. 66, no. 3. — P. 611–645.
3. Flegal James M, Jones Galin L et al. Batch means and spectral variance estimators in Markov chain Monte Carlo // The Annals of Statistics. — 2010. — Vol. 38, no. 2. — P. 1034–1070.
4. Wolter K. Introduction to Variance Estimation. Springer Series in Statistics. — Springer New York, 2007.
5. Stan: A Probabilistic Programming Language / Bob Carpenter, Andrew Gelman, Matthew Hoffman et al. // Journal of Statistical Software, Articles. — 2017. — Vol. 76, no. 1. — P. 1–32.
6. Homan Matthew D., Gelman Andrew. The No-U-turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo // J. Mach. Learn. Res. — 2014. — Vol. 15, no. 1. — P. 1593–1623.
7. Nishimura Akihiko, Dunson David, Lu Jianfeng. Discontinuous Hamiltonian Monte Carlo for sampling discrete parameters // arXiv preprint arXiv:1705.08510. — 2017.
8. Salvatier John, Wiecki Thomas V., Fonnesbeck Christopher. Probabilistic programming in Python using PyMC3 // PeerJ Computer Science. — 2016. — Vol. 2. — P. e55.
9. The "with" Statement : PEP : 343 ; Executor: Guido van Rossum, Nick Coghlan : 2005. — Access mode: `python.org/dev/peps/pep-0343/`.
10. Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions // arXiv e-prints. — 2016. — May. — Vol. abs/1605.02688. — Access mode: `http://arxiv.org/abs/1605.02688`.
11. Strashko Vladislav. wlad111/small_probs: Basic functionality. — 2020. — May. — Access mode: `https://doi.org/10.5281/zenodo.3867194`.