

ГРАФИЧЕСКИЕ ВЕРОЯТНОСТНЫЕ МОДЕЛИ

Сергей Николенко

НИУ ВШЭ — Санкт-Петербург

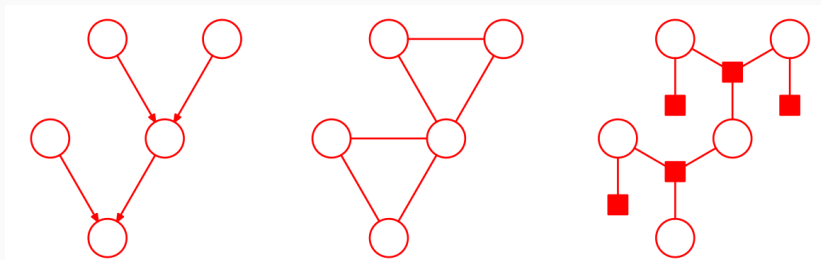
6 октября 2018 г.

Random facts:

- 6 октября — празднование зачатия честного, славного Пророка, Предтечи и Крестителя Господня Иоанна, а также День российского страховщика
- 6 октября 1582 г. в Италии, Польше, Португалии и Испании так и не наступило
- 6 октября 1790 г. ювелир из Женевы Якоб Швейпп представил процесс изготовления искусственной минеральной воды
- 6 октября 1889 г. Томас Эдисон показал свой первый фильм, а 6 октября 1927 г. в Нью-Йорке показали первый звуковой фильм «Певец джаза»
- 6 октября 2010 г. начало работу приложение по обмену фотографиями Instagram

АЛГОРИТМ ПЕРЕДАЧИ СООБЩЕНИЙ

ТРИ ПРЕДСТАВЛЕНИЯ



- Чтобы поставить задачу в общем виде, рассмотрим функцию

$$p^*(X) = \prod_{j=1}^m f_j(X_j),$$

где $X = \{x_i\}_{i=1}^n$, $X_j \subseteq X$.

- Т.е. мы рассматриваем функцию, которая раскладывается в произведение нескольких других функций.

- Задача нормализации: найти $Z = \sum_X \prod_{j=1}^m f_j(X_j)$.
- Задача маргинализации: найти

$$p_i^*(x_i) = \sum_{k \neq i} p^*(X).$$

Также может понадобиться, например, $p_{i_1 i_2}$, но реже.

- Поиск гипотезы максимального правдоподобия:

$$x^* = \arg \max_X p(X).$$

- Все эти задачи NP-трудные.
- То есть, если мир не рухнет, сложность их решения в худшем случае возрастает экспоненциально.
- Но можно решить некоторые частные случаи.

- Давайте начнём с графа в виде (ненаправленной) цепи:

$$p(x_1, \dots, x_n) = \frac{1}{Z} \psi_{1,2}(x_1, x_2) \dots \psi_{n-1,n}(x_{n-1}, x_n).$$

- Мы хотим найти

$$p(x_k) = \sum_{x_1} \dots \sum_{x_{k-1}} \sum_{x_{k+1}} \dots \sum_{x_n} p(x_1, \dots, x_n).$$

- Очевидно, тут можно много чего упростить; например, справа налево:

$$\begin{aligned} \sum_{x_n} p(x_1, \dots, x_n) &= \\ &= \frac{1}{Z} \psi_{1,2}(x_1, x_2) \dots \psi_{n-2,n-1}(x_{n-2}, x_{n-1}) \sum_{x_n} \psi_{n-1,n}(x_{n-1}, x_n). \end{aligned}$$

- Эту сумму можно вычислить отдельно и продолжать в том же духе справа налево, потом аналогично слева направо.

- В итоге процесс сойдётся на узле x_k , куда придут два «сообщения»: слева

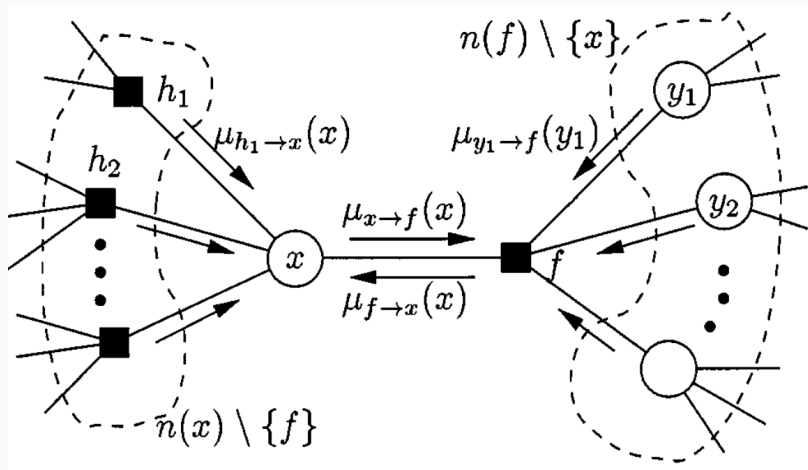
$$\mu_\alpha(x_k) = \sum_{x_{k-1}} \psi_{k-1,k}(x_{k-1}, x_k) \left[\dots \sum_{x_2} \psi_{2,3}(x_2, x_3) \left[\sum_{x_1} \psi_{1,2}(x_1, x_2) \right] \dots \right],$$

справа

$$\mu_\beta(x_k) = \sum_{x_{k+1}} \psi_{k,k+1}(x_k, x_{k+1}) \left[\dots \left[\sum_{x_n} \psi_{n-1,n}(x_{n-1}, x_n) \right] \dots \right].$$

- Каждую частичную сумму можно рассматривать как «сообщение» от узла к своему соседу, причём это сообщение – функция от соседа.

- Чтобы обобщить, удобно рассмотреть опять фактор-граф.
- Предположим, что фактор-граф – дерево (если не дерево, так просто не работает).
- Алгоритм передачи сообщений решает задачу маргинализации для функции вида $p(x_1, \dots, x_n) = \prod_s f_s(X_s)$, заданной в виде фактор-графа.
- Передаём сообщения по направлению к нужному узлу от переменных к функциям и наоборот.



- Чтобы найти $p(x_k)$, запишем

$p(x_1, \dots, x_n) = \prod_{s \in \neq(x_k)} F_s(x_k, X_s)$, где X_s – переменные из поддерева с корнем в f_s . Тогда

$$\begin{aligned} p(x_k) &= \sum_{x_{i \neq k}} p(x_1, \dots, x_n) = \prod_{s \in \neq(x_k)} \left[\sum_{X_s} F_s(x_k, X_s) \right] = \\ &= \prod_{s \in \neq(x_k)} \mu_{f_s \rightarrow x_k}(x_k), \end{aligned}$$

где $\mu_{f_s \rightarrow x_k}(x_k)$ – сообщения от соседних функций к переменной x_k .

- Чтобы найти $\mu_{f_s \rightarrow x_k}(x_k)$, заметим, что $F_s(x_k, X_s)$ тоже можно разложить по соответствующему подграфу:

$$F_s(x_k, X_s) = f_s(x_k, Y_s) \prod_{y \in Y_s} G_y(y, X_{s,y}),$$

где Y_s – переменные, непосредственно связанные с f_s (кроме x_k), $X_{s,y}$ – соответствующие поддеревья.

- Итого получаем

$$\begin{aligned} \mu_{f_s \rightarrow x_k}(x_k) &= \sum_{Y_s} f_s(x_k, Y_s) \prod_{y \in Y_s} \left(\sum_{X_{s,y}} G_y(y, X_{s,y}) \right) = \\ &= \sum_{Y_s} f_s(x_k, Y_s) \prod_{y \in Y_s} \mu_{y \rightarrow f_s}(y). \end{aligned}$$

- Можно аналогично подсчитать, что

$$\mu_{y \rightarrow f_s}(y) = \prod_{f \in \neq(y) f_s} \mu_{f \rightarrow y}(y).$$

- Итак, получился простой и понятный алгоритм:
 - как только узел получил сообщения от всех соседей, кроме одного, он сам начинает передавать сообщение в этого соседа;
 - сообщение по ребру между функцией и переменной является функцией от этой переменной;
 - узел-переменная x передаёт сообщение

$$\mu_{x \rightarrow f}(x) = \prod_{g \in \text{neigh}(x) \setminus \{f\}} \mu_{g \rightarrow x}(x);$$

- узел-функция $f(x, Y)$ передаёт сообщение

$$\mu_{f \rightarrow x}(x) = \sum_{y \in Y} f(x, Y) \prod_{y \in Y} \mu_{y \rightarrow f}(y);$$

- начальные сообщения в листьях $\mu_{x \rightarrow f}(x) = 1, \mu_{f \rightarrow x}(x) = f(x)$.

- Когда сообщения придут из всех соседей в какую-то переменную x_k , можно будет подсчитать

$$p(x_k) = \prod_{f \in \bar{x}_k} \mu_{f \rightarrow x_k}(x_k).$$

- Когда сообщения придут из всех соседей в какой-то фактор $f_s(X_s)$, можно будет подсчитать совместное распределение

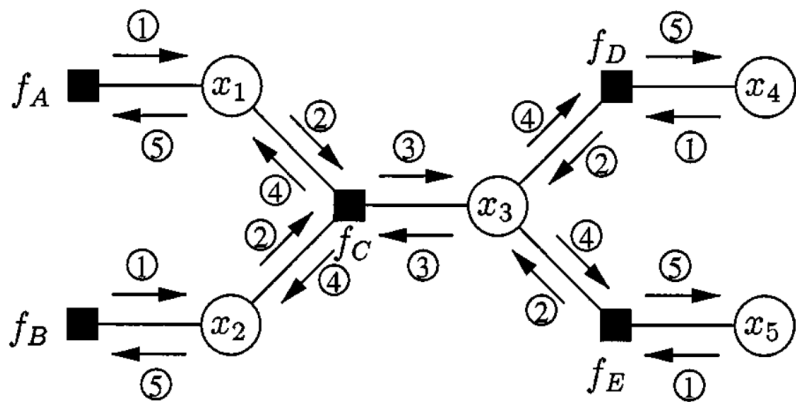
$$p(X_s) = f_s(X_s) \prod_{y \in \bar{f}_s} \mu_{y \rightarrow f_s}(y).$$

- За два прохода (по каждому ребру туда и обратно) можно будет подсчитать маргиналы во всех узлах.

- Это называется алгоритм sum-product, потому что сообщение вычисляется как

$$\mu_{f \rightarrow x}(x) = \sum_{y \in Y} f(x, Y) \prod_{y \in Y} \mu_{y \rightarrow f}(y).$$

- Задача максимизации $\arg \max_x p(x_1, \dots, x_n)$ решается так же, но алгоритмом max-sum: сумма заменяется на максимум, а произведение на сумму.



ТАК ЧТО ЖЕ ДЕЛАТЬ С БАЙЕСОВСКОЙ СЕТЬЮ?

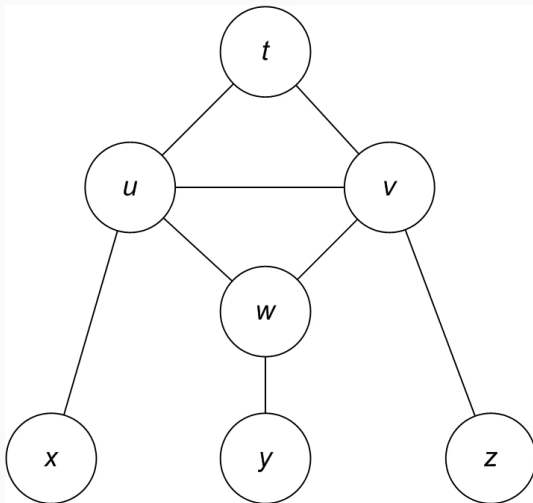
Для модели не в виде фактор-графа надо просто представить её в виде фактор-графа тем или иным способом.

Для байесовской сети это может означать, что надо сначала сделать морализацию, а потом добавить факторы в явном виде.

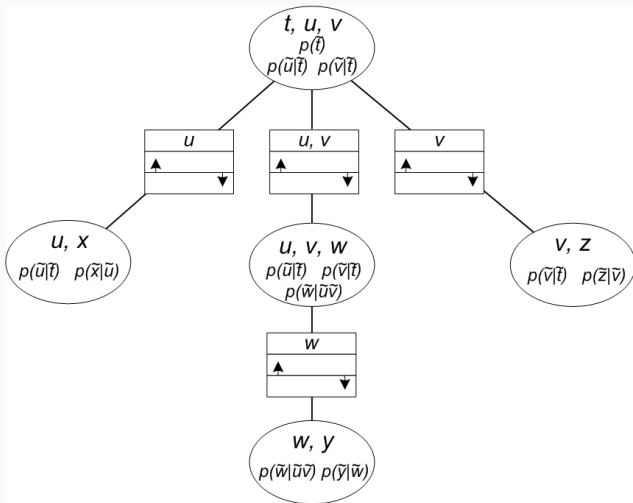
ТАК ЧТО ЖЕ ДЕЛАТЬ С БАЙЕСОВСКОЙ СЕТЬЮ?



ТАК ЧТО ЖЕ ДЕЛАТЬ С БАЙЕСОВСКОЙ СЕТЬЮ?



ТАК ЧТО ЖЕ ДЕЛАТЬ С БАЙЕСОВСКОЙ СЕТЬЮ?



ПРИБЛИЖЁННЫЙ ВЫВОД

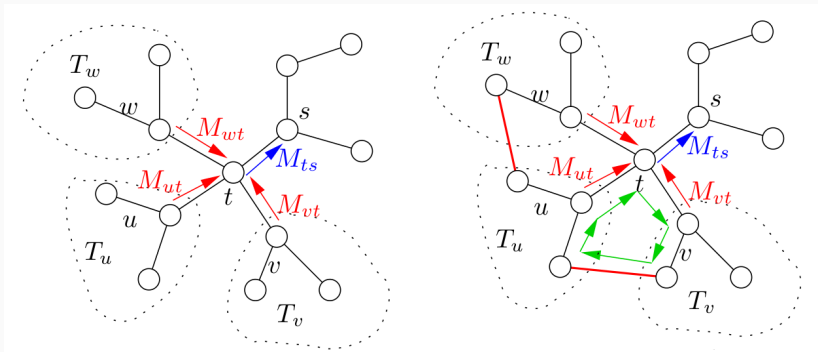
- Когда граф – дерево, и в каждом узле всё считается явно и аналитически, можно посчитать быстро и точно.
- Но что делать, когда зубная щётка недоступна?
- Могут быть две проблемы:
 1. сложная структура графа, с циклами;
 2. сложные факторы – результат маргинализации в отдельном узле неудобный.

- Sum-product работает корректно, только если граф — дерево (ну, разве что скрестить пальцы и помолиться...).
- Что делать, когда граф содержит циклы?
- Нужно использовать деревья сочленений.

- Если цикл не сдаётся, его уничтожают, то есть заменяют весь цикл на одну вершину.
- Получается дерево, в котором уже можно работать обычным sum-product'ом; но при этом, конечно, замена нескольких вершин одной приводит к экспоненциальному раздуванию соответствующего фактора (множество значений соответствующей переменной должно содержать все комбинации значений исходных переменных).

- Если цикл всё-таки большой, то есть хороший общий метод, который применяют, когда нельзя применять sum-product.
- Метод заключается в том, чтобы применять sum-product. :)
- Он работает довольно часто даже тогда, когда в принципе работать не обязан (когда есть циклы).

ПЕРЕДАЧА СООБЩЕНИЙ С ЦИКЛАМИ



- Если факторы простые, а структура сложная, можно приближать сложное распределение более простой формой, разрывая связи в графе: *вариационные приближения* (из матфизики).
- Т.е. надо будет выбрать распределение из некоторого более простого семейства, которое лучше всего приближает сложное распределение.
- «Похожесть» можно определить по расстоянию Кульбака–Лейблера

$$d(p, q) = \int p(x) \ln \frac{p(x)}{q(x)} dx.$$

- Например: давайте искусственно разорвём связи, оставив только рёбра внутри подмножеств вершин X_i .
- Иначе говоря, будем предполагать, что любой фактор $q(Z)$ представляется в виде

$$q(Z) = \prod q_i(Z_i), \text{ где } Z_i = Z \cup X_i.$$

- Затем оптимизируем параметры, минимизируя расстояние между исходным распределением и таким факторизованным; это соответствует методу самосогласованного поля (mean field theory) в физике.
- Более подробно мы рассмотрим вариационные методы позже.

- Если структура простая, но сложные факторы (результат не представляется в виде распределения нужной формы), можно его приближать простыми распределениями. Если в нашем факторизованном распределении

$$p(\theta | D) = \frac{1}{p(D)} \prod_i f_i(\theta)$$

слишком сложные факторы f_i , мы хотим их заменить на какие-нибудь простые (из экспоненциального семейства, например, гауссианы):

$$q(\theta | D) = \frac{1}{Z} \prod_i \hat{f}_i(\theta).$$

- И тоже минимизировать расстояние Кульбака–Лейблера между p и q .

- Для одного фактора всё это очень просто было бы – посчитать среднее и дисперсию (moment matching).
- Для многих факторов надо приближать все \hat{f}_i одновременно. Можно доказать (мы не будем), что это можно делать последовательно, приближая фактор за фактором и итерируя, пока не сойдётся.
- Таким образом, алгоритм Expectation Propagation на самом деле очень простой:
 1. запустить алгоритм передачи сообщений, но на каждом шаге вместо сообщения $\mu_{f_s \rightarrow x_k}(x_k)$ считать его приближение $\hat{\mu}_{f_s \rightarrow x_k}(x_k)$ из какого-нибудь простого семейства;
 2. повторять передачу сообщений, пока не сойдётся.

ВЫБОРКИ С ОТКЛОНЕНИЕМ И ВЕСАМИ

- Пусть у нас есть некоторое вероятностное распределение.
- Как с ним работать? Как, например, его симулировать?
- Мы не всегда можем приблизить (как по методу Лапласа) распределение каким-нибудь известным так, чтобы всё посчитать в явном виде.
- Например, в кластеризации: мультимодальное распределение с кучей параметров, что с ним делать?

- Пусть имеется некое распределение $p(x)$.
- Задача 1: научиться генерировать сэмплы $\{x^{(r)}\}_{r=1}^R$ по $p(x)$.
- Задача 2: научиться оценивать ожидания функций по распределению $p(x)$, т.е. научиться оценивать интегралы вида

$$E_p[f] = \int p(x)f(x)dx.$$

ПОСТАНОВКА ЗАДАЧИ

- Мы будем обычно предполагать, что x — это вектор из \mathbb{R}^n с компонентами x_n , но иногда будем рассматривать дискретные множества значений.
- Функции f — это, например, моменты случайных величин, зависящих от x .
- Например, если $t(x)$ — случайная величина, то её среднее — это $E_p[t(x)] (\int p(x)t(x)dx)$, а её вариация равна $E_p[t^2] - (E_p[t])^2$.
- И мы предполагаем, что явно вычислить не получается — слишком сложная функция p .

- Мы будем заниматься только сэмплингом, потому что задача оценки ожиданий функций легко решится, если мы научимся делать сэмплинг.
- Как она решится?

Ожидания и сэмплинг

- Мы будем заниматься только сэмплингом, потому что задача оценки ожиданий функций легко решится, если мы научимся делать сэмплинг.
- Как она решится?
- Нужно взять сэмплы $\{x^{(r)}\}_{r=1}^R$ и подсчитать

$$\hat{f} = \frac{1}{R} \sum_r f(x^{(r)}).$$

- Ожидание \hat{f} равно $E_p[f]$, а вариация убывает обратно пропорционально R .

- Пример применения: вспомним, где мы часто вычисляем ожидания – в алгоритме EM, на E-шаге:

$$Q(\theta, \theta^{\text{old}}) = \int p(Z | X, \theta^{\text{old}}) \ln p(Z, X | \theta) dZ.$$

- Давайте приблизим:

$$Q(\theta, \theta^{\text{old}}) \approx \frac{1}{R} \sum_{r=1}^R \ln p(Z^{(r)}, X | \theta) dZ.$$

- А потом будем это приближение оптимизировать; получится *Monte Carlo EM*.
- Пример ещё проще: байесовские предсказания – это ожидания известных функций по сложному апостериорному распределению, и посчитать их руками обычно сложно.

Что же сложного в сэмплинге?

- Мы предполагаем, что дана функция $p^*(x)$, которая отличается от $p(x)$ только нормировочной константой $Z = \int p^*(x)dx$: $p(x) = p^*(x)/Z$.
- Почему трудно делать сэмплинг?
- Во-первых, мы обычно не знаем Z ; но это не главное.
- Главное — обычно правильные сэмплы p^* часто попадают туда, где p^* велика. А как определить, где она велика, не вычисляя её *везде*?

- Простейшая идея: давайте дискретизируем пространство, вычислим p^* на каждом участке (пусть она гладкая), потом будем брать дискретные сэмплы, зная все вероятности (это нетрудно).
- Сколько же будет дискретных участков?
- Главная проблема — обычно велика размерность x . Например, если разделить каждую ось на 20 участков, то участков будет 20^n ; а n в реальных задачах может достигать нескольких тысяч...
- Иными словами, такой подход никак не работает.

ПРИМЕР: СКОЛЬКО В ОЗЕРЕ НЕФТИ?

- Перед вами — участок, под которым залежи нефти (да хоть подземное озеро нефти).
- Вам нужно определить, сколько её тут.
- Вы можете проводить замер в каждой конкретной точке, чтобы определить глубину слоя в этой точке.
- Проблема в том, что значительная часть общего объёма нефти может быть сосредоточена в глубоких, но узких каньонах.
- И это только размерность два. :)

- Может быть, всё-таки получится решить хотя бы вторую задачу?
- Давайте брать сэмплы $\{x^{(r)}\}_{r=1}^R$ *равномерно* из всего пространства, затем вычислять там p^* и нормализовать посредством $Z_R = \sum_{r=1}^R p^*(x^{(r)})$.
- Тогда \hat{f} можно будет оценить как

$$\hat{f} = \frac{1}{Z_R} \sum_{r=1}^R f(x^{(r)}) p^*(x^{(r)}).$$

- В чём проблема?

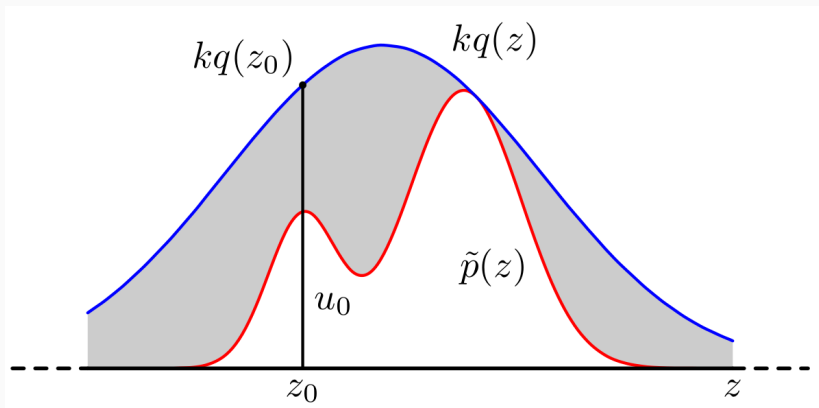
- Да в том же самом.
- Обычно значительная часть p^* сосредоточена в очень небольшой части пространства.
- Вероятность попасть в неё за R равномерно выбранных сэмплов тоже экспоненциально мала (например, если по каждой оси вероятность попасть $1/2$, и всё независимо, то получится вероятность 2^{-n}).
- Так что даже вторую задачу решить не получится.

- Но что-то всё-таки делать надо.
- Выборка с отклонением — rejection sampling.
- Наше предположение теперь в том, что у нас есть q^* , которое мы можем сэмплировать и про которое мы знаем константу c , такую, что

$$\forall x \quad cq^*(x) > p^*(x).$$

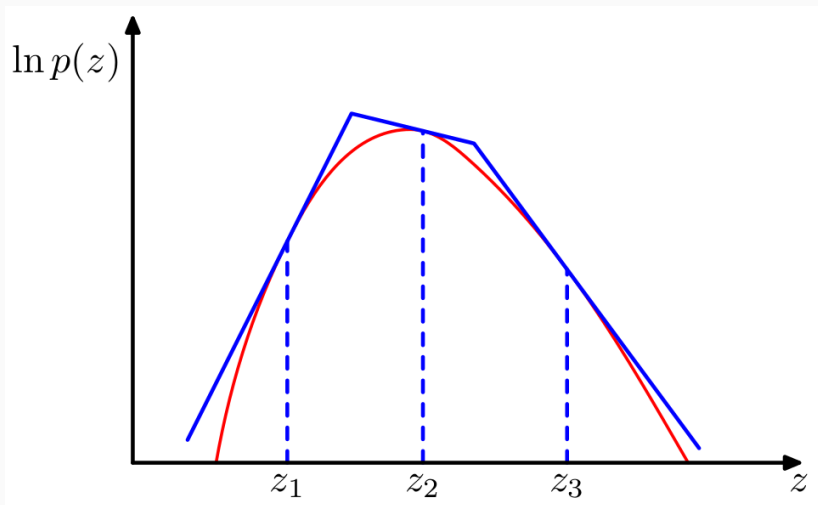
- Тогда мы сумеем сэмплировать p .

- Взять сэмпл x по распределению $q^*(x)$.
- Выбрать случайное число u равномерно из интервала $[0, cq^*(x)]$.
- Вычислить $p^*(x)$. Если $u > p^*(x)$, x отклоняется (отсюда и название), иначе добавляется в сэмплы.



- Алгоритм работает, потому что выбирает точки $[x, u]$ равномерно из области под графиком $p^*(x)$, а это и значит, что получатся сэмплы p^* .
- Вариант – *адаптивная выборка*: если мы можем точнее определить $q(x)$, например построить её как многогранник, касающийся выпуклой (как правило, лог-выпуклой – и многогранник в логарифмическом пространстве) плотности распределения.

АДАПТИВНАЯ ВЫБОРКА С ОТКЛОНЕНИЕМ



СЭМПЛИРОВАНИЕ

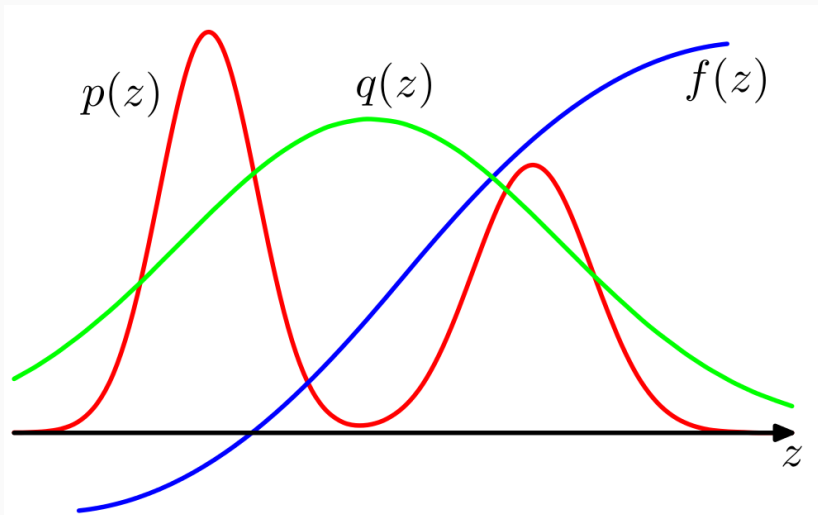
- Вариант выборки с отклонением можно применить к направленным графическим моделям.
- Сэмплировать без evidence – тривиально.
- Сэмплировать с evidence можно так: сделаем сэмпл, если наблюдаемые переменные не сошлись, выкинем.
- Для ненаправленных не так просто, да и для направленных не сработает, если наблюдаемых много.

- Как и у предыдущего алгоритма, у выборки с отклонением начинаются проблемы в больших размерностях.
- Суть проблемы та же, что в предыдущем случае, а выражается она в том, что c будет очень большим (экспоненциальным от n), и почти все сэмплы будут отвергаться.

- Выборка по значимости — importance sampling.
- Мы решаем только вторую задачу, а не первую.
- То есть нам нужно брать сэмплы, при этом желательно попадая в зоны, где функция p^* имеет большие значения.

- Предположим, что у нас есть какое-то другое распределение вероятностей q (точнее, q^*), попроще, и мы умеем брать его сэмплы.
- Тогда алгоритм такой: сначала взять выборку по q^* , а затем перевзвесить её так, чтобы получилась всё-таки выборка по p^* .

ВЫБОРКА ПО ЗНАЧИМОСТИ



- Мы хотим

$$\begin{aligned} E[f] &= \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = \\ &\approx \frac{1}{L} \sum_r \frac{p(x^{(r)})}{q(x^{(r)})} f(x^{(r)}). \end{aligned}$$

- $w_r = p(x^{(r)})/q(x^{(r)})$ – веса, с которыми входят сэмплы, но все сэмплы остаются в множестве.

- Если у нас не p и q , а p^* и q^* , и $p = \frac{1}{Z_p}p^*$, $q = \frac{1}{Z_q}q^*$, то

$$\begin{aligned} \mathbb{E}[f] &= \int f(x)p(x)dx = \frac{Z_q}{Z_p} \int f(x) \frac{p^*(x)}{q^*(x)} q(x)dx \approx \\ &\approx \frac{Z_q}{Z_p} \frac{1}{R} \sum_{r=1}^R \frac{p^*(x^{(r)})}{q^*(x^{(r)})} f(x^{(r)}), \end{aligned}$$

и Z_q/Z_p можно оценить из тех же сэмплов:

$$\frac{Z_p}{Z_q} = \frac{1}{Z_q} \int p^*(x)dx = \int \frac{p^*(x)}{q^*(x)} q(x)dx \approx \frac{1}{R} \sum_{r=1}^R \frac{p^*(x^{(r)})}{q^*(x^{(r)})}.$$

- Получаем такой алгоритм:

1. Взять сэмплы $\{x^{(r)}\}_{r=1}^R$ по распределению q^* .
2. Рассчитать веса

$$w_r = \frac{p^*(x^{(r)})/q^*(x^{(r)})}{\sum_m p^*(x^{(m)})/q^*(x^{(m)})}.$$

3. Оценить функцию по формуле

$$\mathbb{E}[f] \approx \frac{1}{R} \sum_{r=1}^R w_r f(x^{(r)}).$$

- Зачем нужно q ? Чем это лучше равномерного распределения?

- Зачем нужно q ? Чем это лучше равномерного распределения?
- Проще говоря, распределение q должно помочь выбрать те участки, на которых имеет смысл сэмплить r .
- Если q хорошее, то может помочь, а если плохое, может только навредить.
- Но есть и более фундаментальные проблемы.

- Во-первых, сэмплер q не должен быть слишком узким.
- Например, если сэмплер гауссиановский с небольшой вариацией, то пики r далеко от центра q вообще никто не заметит.

- Во-вторых, может случиться, что все сэмплы будут напрочь убиты небольшим количеством сэмплов с огромными весами. Это плохо.
- Чтобы показать, как это бывает, давайте перейдём в многомерный случай.

- Пусть есть равномерное распределение r на единичном шаре и сэмплер q — произведение гауссианов с центром в нуле:

$$p(x) = \frac{1}{(2\pi\sigma^2)^{N/2}} e^{-\frac{1}{2\sigma^2} \sum_i x_i^2}.$$

Упражнение. Найдите среднее и дисперсию расстояния $r^2 = \sum_i x_i^2$ точки, взятой по этому распределению.

- Ответ на упражнение: расстояние будет $N\sigma^2 \pm \sqrt{2N}\sigma^2$ (распределение будет похоже на гауссовское).
- Значит, почти все сэмплы лежат в «типичном множестве», кольце расстоянием около $\sigma\sqrt{N}$ от нуля.

- Тогда большинство сэмплов q будут лежать в интервале

$$\frac{1}{(2\pi\sigma^2)^{n/2}} 2^{-\frac{N}{2} \pm \frac{\sqrt{2N}}{2}},$$

и ненулевые веса будут иметь значения порядка

$$(2\pi\sigma^2)^{n/2} 2^{\frac{N}{2} \pm \frac{\sqrt{2N}}{2}}.$$

- Это значит, что максимальный вес будет относиться к среднему примерно как $2^{\sqrt{2N}}$, а это очень много.

- Варианты выборки по значимости для направленных графических моделей:
 - uniform sampling – фиксируем evidence, выбираем остальные равномерно, вес у сэмпла получается просто $p(x)$, потому что он автоматически сходится с evidence;
 - likelihood weighted sampling – фиксируем evidence, выбираем остальные от родителей к детям из условного распределения $p(x_i \mid \text{pa}(x_i))$, где $\text{pa}(x_i)$ уже зафиксированы; вес тогда будет

$$r(x) = \prod_{x_i \notin E} \frac{p(x_i \mid \text{pa}(x_i))}{p(x_i \mid \text{pa}(x_i))} \prod_{x_i \in E} \frac{p(x_i \mid \text{pa}(x_i))}{1} = \prod_{x_i \in E} p(x_i \mid \text{pa}(x_i)).$$

- Если размерность большая, то у выборки по значимости есть две большие проблемы.
- Во-первых, чтобы получить разумные сэмплы, нужно уже заранее выбрать q так, чтобы оно хорошо аппроксимировало p .
- Во-вторых, даже если их получить, часто может так случиться, что веса у некоторых сэмплов будут слишком велики.
- В общем, для случая многих размерностей это не очень хороший метод.

МАРКОВСКИЕ МЕТОДЫ МОНТЕ-КАРЛО

- Суть алгоритма похожа на выборку с отклонением, но есть важное отличие.
- Распределение q теперь будет меняться со временем, зависеть от текущего состояния алгоритма.
- Как и прежде, нужно распределение q , точнее, семейство $q(x'; x^{(t)})$, где $x^{(t)}$ — текущее состояние.
- Но теперь q не должно быть приближением p , а должно просто быть каким-нибудь сэмплируемым распределением (например, сферический гауссиан).
- Кандидат в новое состояние x' сэмплируется из $q(x'; x^{(t)})$.

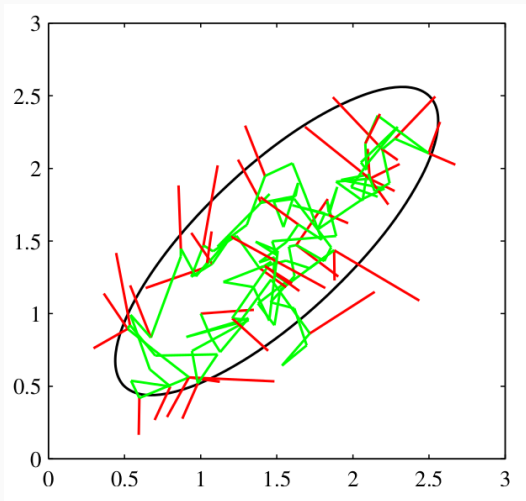
- Очередная итерация начинается с состояния $x^{(i)}$.
- Выбрать x' по распределению $q(x'; x^{(i)})$.
- Вычислить

$$a = \frac{p^*(x')}{p^*(x^{(i)})} \frac{q(x^{(i)}; x')}{q(x'; x^{(i)})}.$$

- С вероятностью a (1, если $a \geq 1$) $x^{(i+1)} := x'$, иначе $x^{(i+1)} := x^{(i)}$.

- Суть в том, что мы переходим в новый центр распределения, если примем очередной шаг.
- Получается этакий random walk, зависящий от распределения p^* .
- $\frac{q(x^{(i)}; x')}{q(x'; x^{(i)})}$ для симметричных распределений (гауссиана) равно 1, это просто поправка на асимметрию.
- Отличие от rejection sampling: если не примем, то не просто отбрасываем шаг, а записываем $x^{(i)}$ ещё раз.

ПРИМЕР БЛУЖДЕНИЯ [ВИНОР]



- Очевидно, что $x^{(i)}$ — отнюдь не независимы.
- Независимые сэмплы получаются только с большими интервалами.
- Поскольку это random walk, то если большая часть q сосредоточена в радиусе ϵ , а общий радиус p^* равен D , то для получения независимого сэмпла нужно будет минимум... сколько?

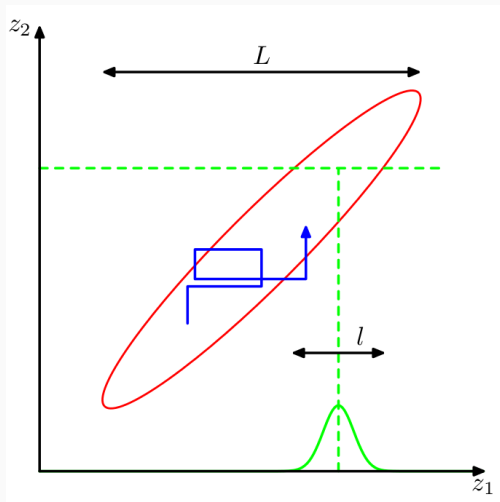
Упражнение. Рассмотрим одномерное случайное блуждание, где на каждом шаге с вероятностью $1/2$ точка движется влево или вправо на единицу длины. Какое ожидаемое расстояние точки от нуля после T шагов?

- Ответ на упражнение: ожидаемое расстояние будет \sqrt{T} .
- Значит, нам потребуется где-то $(\frac{D}{\epsilon})^2$ шагов (и это оценка снизу).
- Хорошие новости: это верно для любой размерности. То есть времени надо много, но нет катастрофы при переходе к размерности 1000.

- Когда размерность большая, можно не сразу все переменные изменять по $q(x'; x)$, а выбрать несколько распределений q_j , каждое из которых касается части переменных, и принимать или отвергать изменения по очереди.
- Тогда процесс пойдёт быстрее, чаще принимать изменения будем.

- Пусть размерность большая. Что делать?
- Давайте попробуем выбирать сэмпл не весь сразу, а покомпонентно.
- Тогда наверняка эти одномерные распределения окажутся проще, и сэмпл мы выберем.

- Пусть есть две координаты: x и y . Начинаем с (x^0, y^0) .
- Выбираем x^1 по распределению $p(x|y = y^0)$.
- Выбираем y^1 по распределению $p(y|x = x^1)$.
- Повторяем.



- В общем виде всё то же самое: x_i^{t+1} выбираем по распределению

$$p(x_i | x_1^{t+1}, \dots, x_{i-1}^{t+1}, x_{i+1}^t, \dots, x_n^t)$$

и повторяем.

- Это частный случай алгоритма Метрополиса (для распределений $q(x'; x) = p(x'_i | x_{-i})$, и вероятность принятия получится 1 – упражнение).
- Поэтому сэмплирование по Гиббсу сходится, и, так как это тот же random walk по сути, верна та же квадратичная оценка.

- Нужно знать $p(x_i | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$. Это, например, особенно легко знать в байесовских сетях.
- Как будет работать сэмплирование по Гиббсу в байесовской сети?
- Для сэмплирования по Гиббсу не нужны никаких особенных предположений или знаний. Можно быстро сделать работающую модель, поэтому это очень популярный алгоритм.
- В больших размерностях может оказаться эффективнее сэмплить по несколько переменных сразу, а не по одной.

- Марковская цепь задаётся начальным распределением вероятностей $p^0(x)$ и вероятностями перехода $T(x'; x)$.
- $T(x'; x)$ — это распределение следующего элемента цепи в зависимости от следующего; распределение на $(t + 1)$ -м шаге равно

$$p^{t+1}(x') = \int T(x'; x)p^t(x)dx.$$

- В дискретном случае $T(x'; x)$ — это матрица вероятностей $p(x' = i | x = j)$.

- Не всякая марковская цепь нам подойдёт.
- Во-первых, цепь должна сходиться к распределению, которое нас интересует.
- Это называется *инвариантным распределением*; инвариантное распределение π удовлетворяет

$$\pi(x') = \int T(x'; x)\pi(x)dx.$$

- Нам нужно, чтобы инвариантным распределением нашей цепи было $p(x)$, которое мы хотим сэмплировать.

- Ну, и нужно, чтобы собственно сходилось:

$$\forall p^0(x) \quad p^t(x) \longrightarrow \pi(x) \text{ при } t \rightarrow \infty.$$

- Какие могут быть примеры неэргодичных цепей?

- Ну, и нужно, чтобы собственно сходилось:

$$\forall p^0(x) \quad p^t(x) \longrightarrow \pi(x) \text{ при } t \rightarrow \infty.$$

- Какие могут быть примеры неэргодичных цепей?
- В цепи могут быть недостижимые состояния (тогда предел зависит от p^0).
- У цепи может быть период, т.е. предельное распределение может меняться с некоторым периодом (например, по соображениям чётности).

- Есть несколько удобных конструкций, с помощью которых можно построить достаточно сложную функцию T , сохраняя её свойства.
- Давайте их рассмотрим.

- Можно конкатенировать распределения, запуская их друг за другом:

$$T(x', x) = \int T_2(x', x'')T_1(x'', x)dx''.$$

- При этом сохраняется инвариантное распределение (докажите).

- Можно смешивать распределения. Если были функции $T_i(x', x)$, то можно ввести новую

$$T(x', x) = \sum_i p_i T_i(x', x), \text{ где } \sum_i p_i = 1.$$

- Как убедиться, что марковская цепь сходится именно к тому распределению, которое нам нужно?
- Свойство баланса в марковских цепях: для p и T

$$\forall x, x' \quad T(x, x')p(x') = T(x', x)p(x).$$

- Т.е. вероятность того, что мы выберем x и дойдём до x' , равна вероятности выбрать x' и дойти до x .
- Такие цепи называются *обратимыми* (reversible).
- Если выполняется условие баланса, то $p(x)$ — инвариантное распределение (докажите!).

- Очередная итерация начинается с состояния $x^{(i)}$.
- Выбрать x' по распределению $q(x'; x^{(i)})$.
- Вычислить

$$a(x', x) = \frac{p^*(x')}{p^*(x^{(i)})} \frac{q(x^{(i)}; x')}{q(x'; x^{(i)})}.$$

- С вероятностью $a(x', x)$ (1, если $a \geq 1$) $x^{(i+1)} := x'$, иначе $x^{(i+1)} := x^{(i)}$.

- Условие баланса:

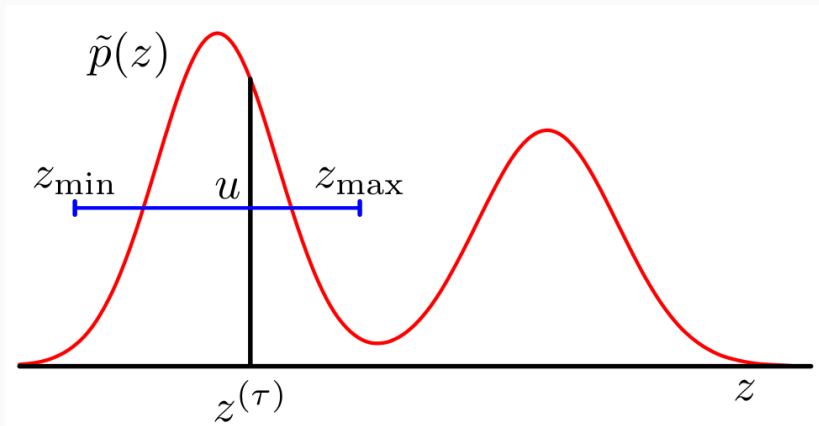
$$\begin{aligned} p(x)q(x; x')a(x', x) &= \min(p(x)q(x; x'), p(x')q(x'; x)) = \\ &= \min(p(x')q(x'; x), p(x)q(x; x')) = p(x')q(x'; x)a(x, x'). \end{aligned}$$

- Важный параметр – дисперсия распределения q ; она задаёт баланс между частым принятием и быстрым перемещением по пространству состояний.

- Slice sampling — ещё один алгоритм, похожий на алгоритм Метрополиса.
- Это аналог алгоритма Метрополиса, но в нём мы хотим настраивать длину шага («дисперсию») автоматически.

- Мы хотим сделать random walk из одной точки под графиком p^* в другую точку под графиком p^* , да так, чтобы в пределе получилось равномерное распределение.
- Вот как будем делать переход $(x, u) \rightarrow (x', u')$:
 - Вычислим $p^*(x)$ и выберем u' равномерно из $[0, p^*(x)]$.
 - Сделаем горизонтальный интервал (x_l, x_r) вокруг x .
 - Затем будем выбирать x' равномерно из (x_l, x_r) , пока не попадём под график.
 - Если не попадаем, модифицируем (x_l, x_r) .
- Осталось понять, как сделать (x_l, x_r) и как его потом модифицировать.

SLICE SAMPLING



- Исходный выбор (x_l, x_r) :
 - Выбрать r равномерно из $[0, \epsilon]$.
 - $x_l := x - r, x_r := x + (\epsilon - r)$.
 - Раздвигать границы на ϵ , пока $p^*(x_l) > u'$ и $p^*(x_r) > u'$.
- Модификация (x_l, x_r) : Если x' лежит выше p^* , сокращаем интервал до x' .

- В алгоритме Метрополиса нужно было выбирать размер шага. И от него всё зависело квадратично.
- А тут размер шага подправляется сам собой, и эта поправка происходит за линейное время (а то и логарифм).
- В задачах с большой размерностью нужно сначала выбрать (случайно или совпадающими с осями) направление изменения y , а потом проводить алгоритм относительно параметра α в распределении $p^*(x + \alpha y)$.

- Рассмотрим ситуацию, когда вероятность можно записать как $p(x) = \frac{1}{Z}e^{-E(x)}$.
- Во многих таких случаях можно вычислить не только $E(x)$, но и градиент $\nabla E(x)$.
- Такую информацию хотелось бы использовать.

- Займёмся матфизикой: рассмотрим механическую систему.
- Состояние системы описывается обобщёнными координатами q и обобщёнными моментами p (векторные переменные).
- Её общая энергия $H(q, p, t) = V(q, t) + K(p, t)$, где V — потенциальная, K — кинетическая.

- Тогда система будет описываться гамильтоновыми уравнениями

$$\dot{p} = -\frac{\partial H}{\partial q}, \quad \dot{q} = \frac{\partial H}{\partial p}.$$

- Гамильтонова механика — это, конечно, то же самое, что лагранжева, но вместо уравнений второго порядка на n переменных получаются уравнения первого порядка на $2n$ переменных.
- Важные для нас свойства: в течение эволюции системы
 1. значение гамильтониана H остаётся постоянным;
 2. объём любой области в пространстве переменных (p, q) сохраняется.

- Гамильтонов метод Монте-Карло — это вариация метода Метрополиса.
- Пространство поиска x расширяется *моментами* p .
- Благодаря законам сохранения гамильтонова динамика оставляет постоянным совместное распределение $p(x, p)$; применяя эволюцию вдоль гамильтониана, можно ходить далеко по пространству состояний, не меняя распределение; а потом делать несколько «обычных» (гиббсовских, например) шагов, которые уже будут менять H .

- Введём гамильтониан $H(x, p) = E(x) + K(p)$, где K — кинетическая энергия, например $K(p) = \frac{p^T p}{2}$.
- Теперь блуждание осуществляется двумя способами: первый случайно блуждает по пространству моментов (по Гиббсу, например).
- А второй шаг пытается сэмплировать совместную вероятность

$$p_H(x, p) = \frac{1}{Z_H} e^{-H(x, p)} = \frac{1}{Z_H} e^{-E(x)} \frac{1}{Z_H} e^{-K(p)}.$$

- Потом можно будет просто отбросить K и получить сэмплы для $e^{-E(x)}$, потому что тут всё так хорошо разделяется.

- Мы хотим построить траекторию в пространстве (x, p) , на которой H остаётся постоянным, а затем по методу Метрополиса либо принять, либо отклонить этот сэмпл.
- Понятно, что $\dot{x} = p$, а гамильтоновы уравнения нам говорят, что

$$\dot{p} = -\frac{\partial E(x)}{\partial x}.$$

- Осталось это проинтегрировать. Для этого можно использовать leapfrog technique приближённого интегрирования:

$$\begin{aligned} p_i(t + \frac{\tau}{2}) &= p_i(t) - \frac{\tau}{2} \left. \frac{\partial E}{\partial x_i} \right|_{x(t)}, \\ x_i(t + \tau) &= x_i(t) + \frac{\tau}{m_i} p_i(t + \frac{\tau}{2}), \\ p_i(t + \tau) &= p_i(t + \frac{\tau}{2}) - \frac{\tau}{2} \left. \frac{\partial E}{\partial x_i} \right|_{x(t+\tau)}. \end{aligned}$$

- Дополнительные «половинные» шаги позволяют добиться погрешности второго порядка по τ .

- Алгоритм делает m leapfrog шагов, потом по методу Метрополиса принимает или отвергает получившуюся точку (проекцию на x).
- То есть если мы можем подсчитывать ∇E , а не только E , мы можем включить эту информацию в наш random walk.
- В результате он будет двигаться более-менее в правильном направлении, и пройденное расстояние \sqrt{n} превратится в n (доказывать уж не будем).

Спасибо за внимание!