ВМК МГУ

Задание 1. Алгоритм Loopy Belief Propagation для LDPC-кодов

Курс: Графические модели, весна 2016

Начало выполнения задания: 29 февраля
Срок сдачи: **13 марта (воскресенье), 23:59.**
Среда для выполнения задания – PYTHON 3.

Содержание

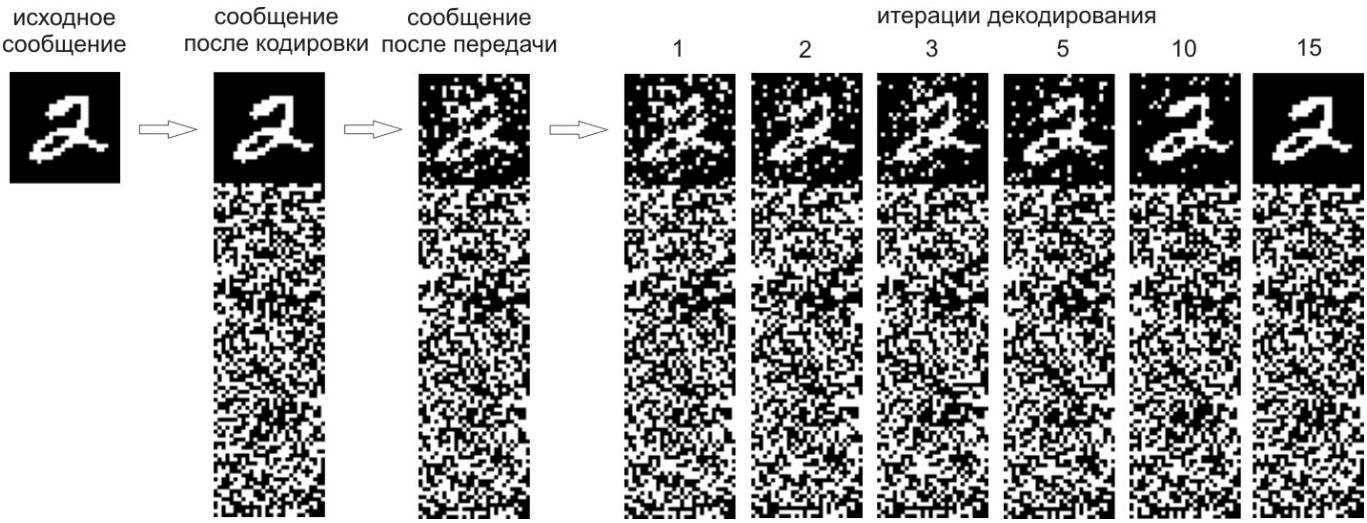

Рис. 1: Демонстрация процесса кодирования/декодирования с использованием низкоплотностного кода. Кодируемое сообщение (изображение цифры) содержит значительную избыточность для возможности визуальной оценки процесса декодирования. Алгоритм декодирования не использует свойство избыточности входного сигнала.

Низкоплотностные коды

Задача помехоустойчивого кодирования

Рассмотрим решение задачи безошибочной передачи потока битовой информации по каналу с шумом с помощью кодов, исправляющих ошибки. При *блоковом кодировании* входящий поток информации разбивается на блоки фиксированной длины k, и каждый блок кодируется/декодируется независимо. Обозначим один такой блок через $\boldsymbol{u} \in \{0,1\}^k$. Предположим, что во входном потоке данных, вообще говоря, нет избыточности. Поэтому для реализации схемы, способной исправлять ошибки, необходимо закодировать блок $\boldsymbol{u}$ в некоторое кодовое слово большей длины путем добавления избыточности в передаваемые данные. Обозначим кодовое слово через $\boldsymbol{v} \in \{0,1\}^n$, $n > k$. Для кодирования всевозможных блоков $\boldsymbol{u}$ необходимо использовать 2^k кодовых слов длины n. Назовём (n,k)-*блоковым кодом* C множество 2^k кодовых слов длины n, а величину $r = k/n$ – *скоростью*

кода. При передаче по каналу с шумом кодовое слово $\boldsymbol{v}$ переходит в принятое слово $\boldsymbol{w}$, которое, вообще говоря, отличается от $\boldsymbol{v}$. Предположим, что при передаче по каналу длина сообщения не изменяется, т.е. $\boldsymbol{w} \in \{0,1\}^n$, а каждый бит независимо инвертируется с некоторой вероятностью q. Задача алгоритма декодирования состоит в восстановлении по $\boldsymbol{w}$ переданного слова $\boldsymbol{v}$ путём поиска среди всевозможных кодовых слов ближайшего к $\boldsymbol{w}$. Обозначим результат работы алгоритма декодирования через $\hat{\boldsymbol{v}}$. На последнем этапе декодированное слово $\hat{\boldsymbol{v}}$ переводится в декодированное слово исходного сообщения $\hat{\boldsymbol{u}}$.

Кодирование с помощью (n,k)-линейного блокового кода

Множество $\{0,1\}^n$ с операциями суммы и произведения по модулю 2 образует линейное пространство над конечным полем из двух элементов $\{0,1\}$. (n,k)-блоковый код C называется *линейным*, если множество его кодовых слов образует линейное подпространство размерности k общего линейного пространства $\{0,1\}^n$. В этом случае C можно задать с помощью некоторого базиса из k элементов $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_k \in \{0,1\}^n$. Тогда произвольное кодовое слово $\boldsymbol{v} \in C$ представляется как $\boldsymbol{v} = \sum_{i=1}^{k} u_i \boldsymbol{g}_i$, где $u_i \in \{0,1\}$ – коэффициенты разложения по базису. Объединим базисные вектора по столбцам в общую матрицу $G = [\boldsymbol{g}_1, \ldots, \boldsymbol{g}_k] \in \{0,1\}^{n \times k}$ – *порождающую матрицу кода*. Тогда процедуру кодирования можно представить как $\boldsymbol{v} = G\boldsymbol{u}$.

Рассмотрим для кода C его ортогональное дополнение $C^\perp = \{z \in \{0,1\}^n : \boldsymbol{z}^T \boldsymbol{v} = 0 \ \forall \boldsymbol{v} \in C\}$. Множество $C^\perp$ также является линейным подпространством $\{0,1\}^n$, причём $\dim\{0,1\}^n = n = \dim C + \dim C^\perp = k + m$, где через m обозначена размерность $C^\perp$. Ортогональное подпространство $C^\perp$ можно задать с помощью базиса из m элементов $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_m \in \{0,1\}^n$. Тогда условие $\boldsymbol{v} \in C$ эквивалентно $\boldsymbol{v}^T \boldsymbol{h}_j = 0 \ \forall j = 1, \ldots, m$. Объединим базисные вектора $\boldsymbol{h}_j$ по строкам в общую матрицу

$$H = \begin{bmatrix} \boldsymbol{h}_1^T \\ \boldsymbol{h}_2^T \\ \ddots \\ \boldsymbol{h}_m^T \end{bmatrix} \in \{0,1\}^{m \times n}.$$

Назовём матрицу H *проверочной матрицей* кода. С её помощью можно проверить, является ли слово $\boldsymbol{v}$ кодовым словом путём проверки соотношения $H\boldsymbol{v} = \boldsymbol{0}$ (здесь и далее все операции с бинарной информацией проводятся по модулю 2).

Линейный блоковый код однозначно задаётся либо порождающей матрицей G, либо проверочной матрицей H. Рассмотрим задачу кодирования слов исходного сообщения $\boldsymbol{u}$ в кодовые слова $\boldsymbol{v}$ (n,k)-линейного блокового кода, заданного своей проверочной матрицей H. Над строками матрицы H можно проводить эквивалентные преобразования, соответствующие выбору другого базиса ортогонального подпространства $C^\perp$. С помощью подобных преобразований матрицу H можно привести к каноническому ступенчатому виду. С точностью до перестановки столбцов канонический ступенчатый вид матрицы H эквивалентен её представлению в виде $\begin{bmatrix} I_m & P \end{bmatrix}$, где I_m – единичная матрица размера $m \times m$. Тогда в качестве порождающей матрицы кода можно выбрать матрицу

$$G = \begin{bmatrix} P \\ I_k \end{bmatrix}. \tag{1}$$

Действительно, в этом случае $H\boldsymbol{v} = HG\boldsymbol{u} = (P + P)\boldsymbol{u} = \boldsymbol{0}$. Кодирование по правилу $\boldsymbol{v} = G\boldsymbol{u}$ с помощью матрицы (1) будет *систематическим*, т.к. здесь все биты слова $\boldsymbol{u}$ копируются в фиксированные (последние) биты кодового слова $\boldsymbol{v}$. При систематическом кодировании обратный процесс преобразования из декодированного кодового слова $\hat{\boldsymbol{v}}$ в декодированное сообщение $\hat{\boldsymbol{u}}$ становится тривиальным.

Декодирование низкоплотностного кода

Низкоплотностным кодом (или кодом с малой плотностью проверок на чётность) называется бинарный (n,k)-линейный блоковый код, в котором проверочная матрица $H \in \{0,1\}^{m \times n}$ является сильно разреженной.

Рассмотрим бинарный симметричный канал для передачи данных. Здесь при передаче каждый бит независимо инвертируется с некоторой вероятностью q. В результате бинарный симметричный канал задает распределение $p(\boldsymbol{w}|\boldsymbol{v})$ для передаваемого кодового слова $\boldsymbol{v} \in \{0,1\}^n$ и полученного на выходе слова $\boldsymbol{w} \in \{0,1\}^n$ как

$$p(\boldsymbol{w}|\boldsymbol{v}) = \prod_{i=1}^{n} p(w_i|v_i) = \prod_{i=1}^{n} q^{w_i + v_i}(1-q)^{w_i + v_i + 1}.$$

Пропускная способность данного канала определяется величиной $1 + q\log_2 q + (1-q)\log_2(1-q)$.

Объединяя низкоплотностный код с бинарным симметричным каналом, получаем следующую вероятностную модель для пары $\boldsymbol{w}, \boldsymbol{v}$:

$$p(\boldsymbol{v}|\boldsymbol{w}) \propto p(\boldsymbol{w}|\boldsymbol{v}) I[H\boldsymbol{v} = \boldsymbol{0}] \propto \prod_{i=1}^{n} p(w_i|v_i) \prod_{j=1}^{m} I[\boldsymbol{h}_j^T \boldsymbol{v} = \boldsymbol{0}].$$

Здесь $m = n-k$ – количество проверок на чётность, $\boldsymbol{h}_j^T$ – j-ая строка матрицы H, а $I[\cdot]$ – индикаторная функция. Фактор-граф введённой модели показан на рис. 2.

Назовём *синдромом* принятого слова $\boldsymbol{w}$ вектор $\boldsymbol{s} \in \{0,1\}^m$, определяемый как $\boldsymbol{s} = H\boldsymbol{w}$. Процесс передачи кодового слова по бинарному симметричному каналу можно представить как $\boldsymbol{w} = \boldsymbol{v} + \boldsymbol{e}$, где $\boldsymbol{e} \in \{0,1\}^n$ – вектор ошибок ($e_i = 1$, если в позиции i произошла ошибка). Тогда $\boldsymbol{s} = H\boldsymbol{w} = H(\boldsymbol{v} + \boldsymbol{e}) = H\boldsymbol{v} + H\boldsymbol{e} = H\boldsymbol{e}$. Далее можно перейти от вероятностной модели для переменных $\boldsymbol{v}, \boldsymbol{w}$ к аналогичной для переменных $\boldsymbol{e}, \boldsymbol{s}$:

$$p(\boldsymbol{e}|\boldsymbol{s}) \propto p(\boldsymbol{e})p(\boldsymbol{s}|\boldsymbol{e}) \propto \prod_{i=1}^{n} p(e_i) \prod_{j=1}^{m} I[\boldsymbol{h}_j^T \boldsymbol{e} = s_j].$$

Здесь $p(e_i) = q^{e_i}(1-q)^{1-e_i}$. Зная значение вектора ошибок $\hat{\boldsymbol{e}}$, результат декодирования вычисляется как $\hat{\boldsymbol{v}} = \boldsymbol{w} + \hat{\boldsymbol{e}}$. При использовании вероятностной модели для $\boldsymbol{e}, \boldsymbol{s}$ тестирование алгоритма декодирования можно проводить без предварительной реализации процедуры кодирования.

При использовании побитовой функции потерь $\lambda(\boldsymbol{e}, \tilde{\boldsymbol{e}}) = \sum_{i=1}^{n} I[e_i \neq \tilde{e}_i]$ оптимальная процедура декодирования связана с максимизацией маргиналов отдельных переменных, т.е. $\hat{e}_i = \arg\max p(e_i|\boldsymbol{s})$. Для поиска маргинальных распределений $p(e_i|\boldsymbol{s})$ воспользуемся циклическим алгоритмом передачи сообщений (sum-product loopy BP) на фактор-графе. Введём обозначения $N(i) = \{j : H_{ji} = 1\}$ – множество факторов, в которых участвует переменная e_i, и $N(j) = \{i : H_{ji} = 1\}$ – множество переменных, которые входят в фактор h_j. Тогда общая схема алгоритма декодирования выглядит следующим образом:

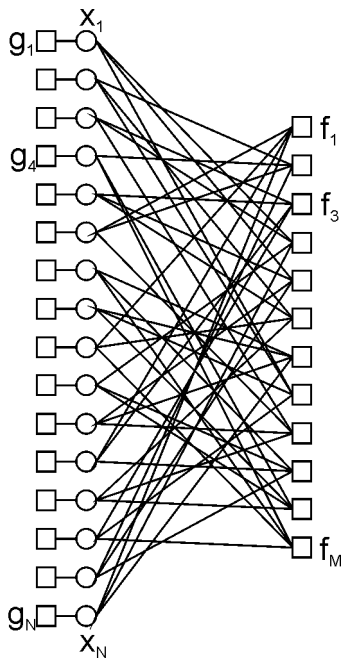

Рис. 2: Фактор-граф для $(16, 4)$-низкоплотностного кода, в проверочной матрице которого в каждом столбце по 3 единицы, а в каждой строке – по 4 единицы.

1. Инициализация:
$$\mu_{e_i \to h_j}(e_i) = p(e_i);$$

2. Пересчёт сообщений от факторов:
$$\mu_{h_j \to e_i}(e_i) = \sum_{e_k : k \in N(j) \setminus i} I[e_i + \sum_k e_k = s_j] \prod_{k \in N(j) \setminus i} \mu_{e_k \to h_j}(e_k);$$

3. Пересчёт сообщений от переменных и вычисление beliefs (оценок на маргинальные распределения):
$$\mu_{e_i \to h_j}(e_i) \propto p(e_i) \prod_{k \in N(i) \setminus j} \mu_{h_k \to e_i}(e_i);$$
$$b_i(e_i) \propto p(e_i) \prod_{k \in N(i)} \mu_{h_k \to e_i}(e_i);$$

4. Оценка вектора ошибок:
$$\hat{e}_i = \arg\max b_i(e_i);$$

5. Критерий остановки:
   - Если $H\hat{\boldsymbol{e}} = \boldsymbol{s}$, то выход алгоритма со статусом 0;
   - Если произошла стабилизация по всем b_i, то выход алгоритма со статусом 1;
   - Если достигнуто максимальное число итераций, то выход алгоритма со статусом 2;

6. Переход к шагу 2.

Оптимизации в процедуре декодирования

Пересчёт сообщений от факторов

При прямой реализации шага 2 общего алгоритма декодирования требуется рассмотрение для каждого фактора $2^{|N(j)|-1}$ различных конфигураций значений переменных e_k. Это может приводить как к низкой скорости пересчёта сообщений, так и к большим требованиям по памяти.

Рассмотрим более эффективную схему реализации шага 2. Пусть нам необходимо вычислить сообщение $\mu_{h_j \to e_i}(e_i)$. Перенумеруем все переменные, входящие в j-ый фактор (кроме переменной e_i), как $e_{(1)}, e_{(2)}, \ldots, e_{(l)}$, где $l = |N(j)| - 1$. Тогда вычисление сообщения $\mu_{h_j \to e_i}(e_i)$ можно записать как

$$\mu_{h_j \to e_i}(e_i) \propto \sum_{e_{(k)}} I[\sum_{k=1}^{l} e_{(k)} = s_j + e_i] \prod_{k=1}^{l} \mu_{e_{(k)} \to h_j}(e_{(k)}).$$

Данный результат можно интерпретировать как вычисление вероятности $p(\sum_{k=1}^{l} e_{(k)} = s_j + e_i)$ для набора независимых бинарных переменных $e_{(k)}$ с распределением $p_{(k)}(e_{(k)}) = \mu_{e_{(k)} \to h_j}(e_{(k)})$. Обозначим через e^k сумму первых k переменных $e_{(k)}$, т.е. $e^k = e_{(1)} + e_{(2)} + \cdots + e_{(k)}$. Тогда распределение на e^k можно итерационно пересчитывать по формуле

$$p_k(e^k) = \sum_{e^{k-1}} p_{k-1}(e^{k-1}) p_{(k)}(e^k - e^{k-1}).$$

В результате вычисление $p_l(e^l)$ имеет линейную по l сложность.

Дальнейшее повышение эффективности реализации шага 2 связано с рассмотрением разностей вероятностей $\delta p_{(k)} = p_{(k)}(0) - p_{(k)}(1)$ и $\delta p_k = p_k(0) - p_k(1)$. Нетрудно показать, что

$$\delta p_l = \prod_{k=1}^{l} \delta p_{(k)}. \tag{2}$$

Зная значение δp_l и учитывая условие нормировки $p_l(0) + p_l(1) = 1$, сами вероятности могут быть вычислены как $p_l(0) = \frac{1}{2}(1 + \delta p_l)$, $p_l(1) = \frac{1}{2}(1 - \delta p_l)$. Таким образом, $\mu_{h_j \to e_i}(e_i) = p_l(s_j + e_i)$. Основное преимущество условия (2) по сравнению с последовательным пересчётом распределений $p_k(e^k)$ связано с тем, что формула (2) может быть реализована с помощью векторных операций в PYTHON.

Расписание пересчёта сообщений и дэмпфирование

Общая схема циклического алгоритма передачи сообщений оставляет определённый произвол в выборе расписания пересчёта сообщений. Обычно здесь рассматриваются следующие подходы:

- *Параллельное расписание.* В данном случае сначала все вершины посылают сообщения во все факторы, а затем все факторы посылают сообщения во все вершины.
- *Последовательное расписание.* Здесь в цикле по вершинам фактор-графа для данной вершины сначала вычисляются все входящие сообщения от соседних факторов, а затем вычисляются все исходящие сообщения в соседние факторы. Аналогично цикл можно проводить по факторам.

При использовании дэмпфирования с параметром $\lambda \in (0, 1]$ сообщения на итерации t пересчитываются как

$$\mu^t_{h_j \to e_i}(e_i) = \lambda \mu_{h_j \to e_i}(e_i) + (1 - \lambda) \mu^{t-1}_{h_j \to e_i}(e_i);$$
$$\mu^t_{e_i \to h_j}(e_i) = \lambda \mu_{e_i \to h_j}(e_i) + (1 - \lambda) \mu^{t-1}_{e_i \to h_j}(e_i).$$

Здесь $\mu_{h_j \to e_i}(e_i)$ и $\mu_{e_i \to h_j}(e_i)$ – сообщения, вычисляемые на шагах 2 и 3 общего алгоритма декодирования.

Формулировка задания

1. Реализовать алгоритм построения по заданной проверочной матрице чётности H порождающей матрицы кода G для систематического кодирования;
2. Реализовать алгоритм декодирования низкоплотностного кода на основе loopy BP; при реализации шага 2 пересчета сообщений от факторов к переменным необходимо использовать эффективные схемы, обозначенные выше; реализовать последовательное и параллельное расписание пересчёта сообщений, а также дэмпфирование сообщений; сделать иллюстративную картинку процесса декодирования (например, как на рис. 1);
3. Провести эксперименты с различными расписаниями пересчёта сообщений и коэффициентами дэмпфирования; в частности, оценить долю стабилизировавшихся beliefs в зависимости от номера итерации алгоритма декодирования (усреднённую по различным запускам); оценить время работы алгоритма декодирования в зависимости от выбранного расписания и коэффициента дэмпфирования;

4. Рассмотрим две характеристики качества кода — вероятность совершить ошибку хотя бы в одном бите при декодировании блока ($p(\hat{e} \neq e)$) и среднюю вероятность совершить ошибку при декодировании в одном бите ($\frac{1}{n}\sum_{i=1}^{n} p(\hat{e}_i \neq e_i)$). Требуется реализовать алгоритм оценки вероятности битовой и блоковой ошибки кода с помощью метода стат. испытаний (многократная случайная генерация вектора ошибок $\boldsymbol{e}$ ($e_i = 1$ с вероятностью q), вычисление по нему синдрома $\boldsymbol{s} = H\boldsymbol{e}$, восстановление вектора ошибок $\hat{\boldsymbol{e}}$ с помощью алгоритма декодирования и подсчет необходимых характеристик);

5. Провести эксперименты по оцениванию битовой и блоковой ошибки низкоплотностного кода для различных значений длины кодового слова n, скорости кода r, вероятности инвертирования бита при передаче по каналу связи q и среднего количества единиц в столбце проверочной матрицы j. В частности, необходимо проанализировать следующие ситуации:

   - Теорема Шеннона определяет пропускную способность канала как максимально допустимую скорость кода, при которой возможно осуществление надежной коммуникации. Требуется проверить, как меняются характеристики кода при изменении скорости r от минимального значения до пропускной способности канала (при изменении скорости r рекомендуется фиксировать длину кода n).
   - Теорема Шеннона предполагает, что качество кода растет при увеличении длины кодового слова n (при фиксированной скорости r). Требуется проверить это предположение.
   - Одно из следствий теоремы Шеннона утверждает, что хорошими кодами являются коды со случайной проверочной матрицей H. В частности, здесь предполагается, что качество кода должно расти при увеличении среднего количества единиц в столбце проверочной матрицы j. Требуется проверить это утверждение для низкоплотностных кодов (путём рассмотрения нескольких значений j, начиная от 3).

6. Провести эксперименты по сравнению низкоплотностного кода с кодами БЧХ[1]; в частности, рассмотреть ситуацию длинных и коротких кодов;

7. Составить отчёт в формате PDF с описанием всех проведённых исследований.

Рекомендации по выполнению задания

1. Разреженную проверочную матрицу кода заданных размеров можно строить с помощью случайной генерации с соблюдением условия полноранговости. Для оценки ранга проверочной матрицы можно использовать процедуру построения порождающей матрицы кода. При проведении случайной генерации матрицы по столбцам гарантируется, что каждый фактор и каждая вершина будут иметь непустое множество соседей.

2. При тестировании алгоритма декодирования рекомендуется пробовать запускать итерационный процесс из различных начальных приближений для сообщений (не только из значений унарных потенциалов). При корректной реализации метода должна наблюдаться сходимость к одним и тем же финальным сообщениям независимо от начальных приближений.

Оформление задания

Выполненное задание следует отправить письмом по адресу *bayesml@gmail.com* с заголовком письма

[ГМ16] Задание 1 Фамилия Имя.

Убедительная просьба присылать выполненное задание **только один раз** с окончательным вариантом. Также убедительная просьба строго придерживаться заданных ниже прототипов реализуемых функций.

Выполненный вариант задания должен содержать PDF файл с отчётом, а также py-файл с реализацией функций *make_generator_matrix*, *decode* и *estimate_errors* по прототипам в таблицах 1, 2, 3.

Список литературы

[1] D. J. C. MacKay. Information Theory, Inference, and Learning Algorithms. Cambridge University Press, 2003. Раздел 47.

[1] Этот пункт задания является необязательным для слушателей не из 417-й группы

Таблица 1: Построение порождающей матрицы для систематического кодирования

G, ind = make_generator_matrix(H)
ВХОД H – проверочная матрица кода, numpy array, бинарная матрица размера $m \times n$;
ВЫХОД G – порождающая матрица кода, numpy array, бинарная матрица размера $n \times k$; ind – номера позиций кодового слова, в которые копируются биты исходного сообщения, т.е. $G[ind, :]$ является единичной матрицей.

Таблица 2: Алгоритм декодирования LDPC-кода в синдромном представлении

e, status = decode(s, H, q, schedule = 'parallel', damping = 1, max_iter = 300, tol_beliefs = 1e-4, display = False)
ВХОД s – наблюдаемый синдром, numpy array, бинарный вектор длины m; H – проверочная матрица кода, numpy array, бинарная матрица размера $m \times n$; q – вероятность инверсии бита при передаче по каналу связи, число от 0 до 1/2; $schedule$ – расписание пересчёта сообщений, возможные значения 'parallel' и 'sequential'; $damping$ – коэффициент дэмпфирования при пересчёте сообщений, число от 0 до 1; max_iter – максимальное число итераций алгоритма декодирования; $tol\_beliefs$ – порог стабилизации для beliefs, число; $display$ – режим отображения, True или False, если True, то отображается промежуточная информация на итерациях.
ВЫХОД e – восстановленный вектор ошибок, бинарный вектор-столбец длины n; status – результат декодирования: 0, если найден вектор $\boldsymbol{e}$, соответствующий входному синдрому $\boldsymbol{s}$, 1, если произошла стабилизация значений beliefs, 2, если произошёл выход по максимальному числу итераций.

Таблица 3: Оценка характеристик LDPC-кода с помощью метода Монте Карло

err_bit, err_block, diver = estimate_errors(H, q, num_points = 200)
ВХОД H – проверочная матрица кода, numpy array, бинарная матрица размера $m \times n$; q – вероятность инверсии бита при передаче по каналу связи, число от 0 до 1/2; num_points – общее количество стат. испытаний, число;
ВЫХОД err_bit – вероятность битовой ошибки декодирования (относительно n бит кодового слова), число от 0 до 1, вычисляется по тем ситуациям, когда алгоритм декодирования сошёлся (status < 2); err_block – вероятность блоковой ошибки декодирования, число от 0 до 1, вычисляется по тем ситуациям, когда алгоритм декодирования сошёлся (status < 2); $diver$ – доля ситуаций расходимости алгоритма декодирования, число от 0 до 1.