МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Кубанский государственный университет»**

**(ФГБОУ ВО «КубГУ»)**

Факультете математики и компьютерных наук

Кафедра вычислительной математики и информатики

**ОТЧЕТ О ПРОХОЖДЕНИИ УЧЕБНОЙ ПРАКТИКИ**

**Практика по получению первичных профессиональных   
умений и навыков**

по направлению подготовки 02.04.01 Математика и компьютерные науки

направленность (профиль): вычислительная математика

Работу выполнила \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Васильченко М.?

(подпись, дата)

Руководитель

учебной практики

кандидат педагогических наук, доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Царёва И.Н.

(подпись, дата)

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc199723863)

[1 Описание реализованных алгоритмов на сайте 4](#_Toc199723864)

[1.1 Алгоритм рендеринга математических формул MathJax.js 4](#_Toc199723865)

[1.2 Алгоритм интерактивного слайдера с решениями задач slider.js 5](#_Toc199723866)

[1.3 Алгоритм отображения всплывающих подсказок tips.js 6](#_Toc199723867)

[1.4 Алгоритм интерактивных вычислений по комбинаторным формулам interactive\_formulas.js 7](#_Toc199723868)

[1.5 Алгоритм решения интерактивных задач по комбинаторике interactive\_tasks.js 9](#_Toc199723869)

[1.6 Алгоритм работы мини-квизов mini\_quiz.js 10](#_Toc199723870)

[1.7 Алгоритм интерактивной диаграммы Венна interactive\_diagrams.js (с использованием D3.js) 11](#_Toc199723871)

[1.8 Алгоритм симуляции вероятностных экспериментов interactive\_experiment.js 12](#_Toc199723872)

[1.9 Алгоритм расчета вероятности в лотерее lotto.js 13](#_Toc199723873)

[1.10 Алгоритм расчета вероятности извлечения шаров balls-probability.js 15](#_Toc199723874)

[1.11 Алгоритм расчета полной вероятности для удобрений fertilizer-probability.js 16](#_Toc199723875)

[1.12 Алгоритм расчета по формуле Байеса (задача о диетах) interactive-bayes.js 17](#_Toc199723876)

[1.13 Алгоритм расчета по формуле Бернулли interactive-bernulli.js 18](#_Toc199723877)

[1.14 Алгоритм раскрытия/сворачивания спойлеров toggleQuiz 19](#_Toc199723878)

[Заключение 21](#_Toc199723879)

[Список использованных источников 23](#_Toc199723880)

Введение

Данная учебная практика была направлена на модернизацию и повышение интерактивности существующего веб-сайта, который изначально был разработан в рамках моей дипломной работы. Первоначальная версия сайта представляла собой платформу, ориентированную на проверку знаний студентов по ключевым темам теории вероятностей и математической статистики посредством онлайн-тестирования. Функциональность сайта позволяла пользователям проходить тесты, получать результаты и анализировать свои ошибки.

В ходе консультаций с научным руководителем была поставлена задача по существенному расширению возможностей сайта за счет внедрения интерактивных элементов. Основная цель такой модернизации заключалась в повышении вовлеченности пользователей в учебный процесс, улучшении визуального восприятия и понимания сложного теоретического материала, а также в предоставлении инструментов для непосредственного практического применения изучаемых концепций и формул. Предполагалось, что добавление интерактивности сделает процесс обучения более динамичным и эффективным.

В рамках учебной практики мною был проведен анализ структуры и содержания теоретических разделов сайта для выявления областей, наиболее подходящих для внедрения интерактивных компонентов. На основе этого анализа были определены типы интерактивных элементов (калькуляторы формул, симуляторы экспериментов, интерактивные диаграммы, всплывающие подсказки, мини-квизы) и разработаны соответствующие алгоритмы на языке JavaScript. Особое внимание уделялось тому, чтобы новые элементы органично вписывались в существующую структуру сайта и способствовали более глубокому осмыслению материала пользователями.

1 Описание реализованных алгоритмов на сайте

В данном разделе представлено описание алгоритмов, использованных при разработке интерактивных компонентов веб-сайта. Рассмотрим скрипты, подключенные к главной странице (index.html)

* 1. Алгоритм рендеринга математических формул MathJax.js

1. **Источник:** Внешняя библиотека, подключаемая через CDN (https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.1/MathJax.js?config=MML\_CHTML)
2. **Блок:** Глобальный, применяется ко всем математическим выражениям на сайте.

Для корректного отображения математических формул на страницах сайта используется внешняя JavaScript-библиотека MathJax. Данная библиотека обеспечивает преобразование текстовых представлений формул, записанных в форматах LaTeX, MathML (как указано в конфигурационном блоке <script type="text/x-mathjax-config;executed=true"> в index.html) или AsciiMath, в визуально корректные математические выражения. Алгоритм работы MathJax включает следующие этапы:

1. Сканирование DOM-структуры документа на предмет наличия стандартных тегов разметки математического контента (например, <math>) или пользовательских разделителей, указывающих на наличие формулы.
2. Парсинг содержимого этих тегов для построения внутреннего представления математического выражения.
3. Рендеринг выражения с использованием веб-шрифтов или векторной графики SVG, обеспечивая правильное расположение символов, индексов, дробей, интегралов и других математических конструкций.  
   Конфигурационный блок в index.html также активирует расширения для MathML и устанавливает автоматический перенос строк в длинных формулах для улучшения читаемости.

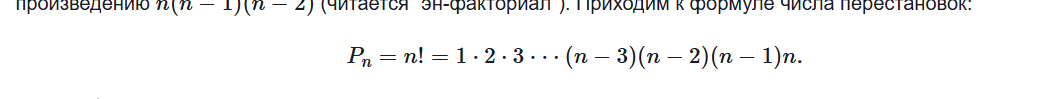


Рисунок 1 – Пример блока с визуализацией формул

1.2 Алгоритм интерактивного слайдера с решениями задач slider.js

1. **Источник:** Файл scripts/slider.js
2. **Блок:** Слайдер с примерами задач (например, в разделе "Классическое определение вероятности" и аналогичных блоках, содержащих элементы с классом .slide).

Мной был разработан алгоритм для управления интерактивным слайдером, который отображает примеры задач и позволяет просматривать их решения в модальном окне.

1. **Инициализация:** При полной загрузке DOM-структуры (DOMContentLoaded) скрипт получает ссылки на ключевые HTML-элементы: коллекцию слайдов (.slide), кнопки навигации "вперед" (.next-btn) и "назад" (.prev-btn), кнопку "показать решение" (.show-solution), а также элементы модального окна (.solution-modal, .solution-text, .close-modal). Инициализируется переменная currentSlide (индекс текущего слайда) и массив examples, содержащий объекты с текстами вопросов и HTML-разметкой решений.
2. **Навигация по слайдам:**
   1. Обработчики событий для кнопок .next-btn и .prev-btn инкрементируют или декрементируют значение currentSlide соответственно. Реализована циклическая прокрутка: currentSlide = (currentSlide + 1) % elements.slides.length и currentSlide = (currentSlide - 1 + elements.slides.length) % elements.slides.length.
   2. Функция updateSlide отвечает за визуальное переключение слайдов, добавляя класс active текущему слайду и удаляя его у остальных.
3. **Отображение решения:** При нажатии на кнопку .show-solution происходит следующее:
   1. Содержимое элемента .solution-text обновляется HTML-кодом решения для examples[currentSlide].solution.
   2. Модальному окну .solution-modal устанавливается style.display = 'block', делая его видимым.
   3. Вызывается MathJax.Hub.Queue(["Typeset", MathJax.Hub]) для корректного рендеринга математических формул в загруженном решении.
4. **Закрытие модального окна:** Реализовано два способа закрытия:
   1. По клику на элемент .close-modal.
   2. По клику на область вне контента модального окна (проверяется, если event.target === elements.modal).  
      В обоих случаях модальному окну устанавливается style.display = 'none'.

1.3 Алгоритм отображения всплывающих подсказок tips.js

1. **Источник:** Файл scripts/tips.js
2. **Блок:** Глобальный, применяется ко всем элементам с классом tooltip-keyword на сайте.

Для предоставления дополнительной информации по ключевым терминам мной был реализован алгоритм создания и управления модальными окнами-подсказками.

1. **Инициализация:** Скрипт выполняет поиск всех HTML-элементов, имеющих класс tooltip-keyword.
2. **Динамическое создание модальных окон:** Для каждого найденного элемента tooltip-keyword:
   1. Извлекается текст подсказки из атрибута data-tooltip.
   2. Генерируется уникальный идентификатор для модального окна с помощью функции generateUniqueId.
   3. Динамически создается HTML-структура модального окна (div с классом tooltip-modal, содержащий modal-content, кнопку закрытия .close-button и текстовый блок .modal-text).
   4. Созданное модальное окно добавляется в document.body.
3. **Управление видимостью модальных окон:**
   1. На каждый элемент tooltip-keyword назначается обработчик события click. При срабатывании, соответствующему модальному окну устанавливается style.display = 'flex', а элементу document.body добавляется класс modal-open для возможной блокировки прокрутки основной страницы.
   2. Кнопке закрытия .close-button внутри каждого модального окна также назначается обработчик click, который скрывает окно и удаляет класс modal-open с document.body.
   3. Реализован глобальный обработчик события click на window, который проверяет, был ли клик произведен вне активного модального окна. Если да, окно закрывается.

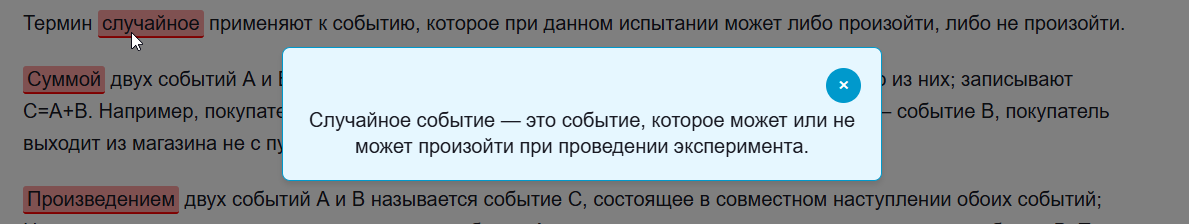


Рисунок 2 – Внешний вид типового модального окна с блоком подсказки

1.4 Алгоритм интерактивных вычислений по комбинаторным формулам interactive\_formulas.js

1. **Источник:** Файл scripts/interactive\_formulas.js
2. **Блок:** Интерактивные карточки с формулами (например, "Формула перестановок", "Формула сочетаний"), идентифицируемые классом formula-card.

Мной был разработан алгоритм, обеспечивающий интерактивный расчет значений по основным комбинаторным формулам на основе пользовательского ввода.

1. **Инициализация:** Скрипт находит все элементы с классом formula-card. Для каждого такого элемента определяются поля ввода для n (.n-input), k (.k-input, если применимо) и блок для вывода результата (.result).
2. **Вспомогательные функции:**
   1. factorial(n): Рекурсивная функция для вычисления факториала с применением кеширования для оптимизации повторных вычислений одних и тех же значений.
   2. debounce(func, delay): Функция для отложенного вызова переданной func через delay миллисекунд после последнего события, что предотвращает избыточные вычисления при быстром вводе данных.
3. **Обновление расчета (updateCalculation):** Данная функция, обернутая в debounce, выполняет следующие действия:
   1. Считывание и валидация значений n и k из полей ввода. Значение k автоматически ограничивается значением n.
   2. В зависимости от наличия классов combination (сочетания), permutation (размещения) или permutations (перестановки) у родительского элемента formula-card, производится расчет по соответствующей формуле:
      1. Сочетания: C(n,k) = n! / (k! \* (n-k)!)
      2. Размещения (в данном коде используется имя permutation): A(n,k) = n! / (n-k)!
      3. Перестановки (в данном коде используется имя permutations): P(n) = n!
   3. Результат расчета и формула с подставленными значениями отображаются в блоке .result.
   4. Вызывается MathJax.Hub.Queue(["Typeset", MathJax.Hub]) для корректного отображения формул.
4. **Обработчики событий:** На поля ввода nInput и kInput назначены обработчики события input, вызывающие updateCalculation. Для kInput также добавлен обработчик blur для коррекции значения, если оно превышает n.

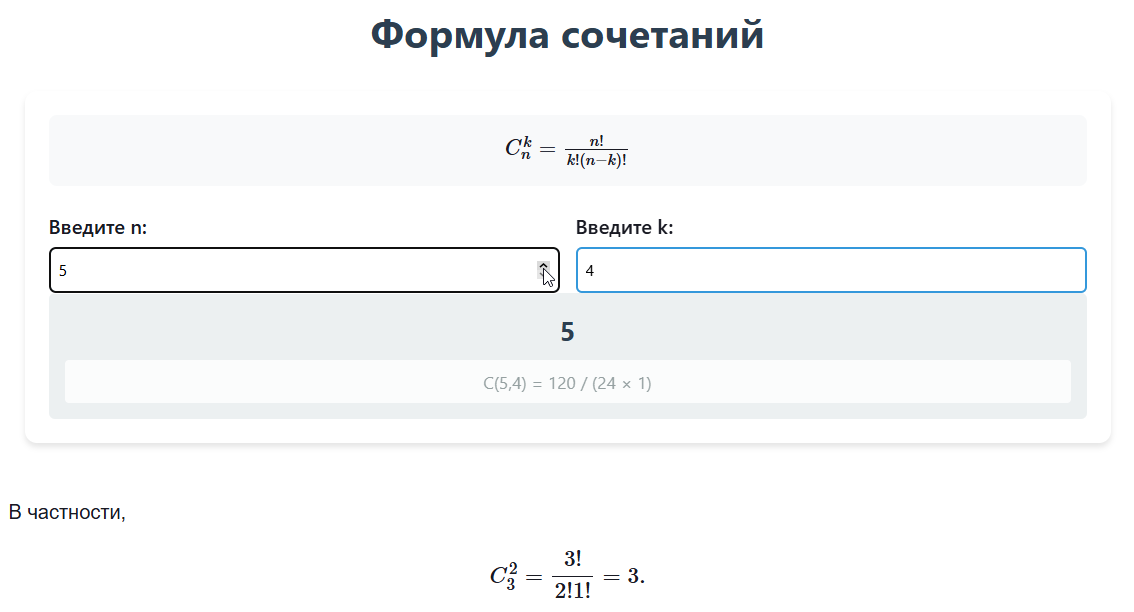


Рисунок 3 – Блок с интерактивной формулой

1.5 Алгоритм отображения интерактивных формул в задачах по комбинаторике interactive\_tasks.js

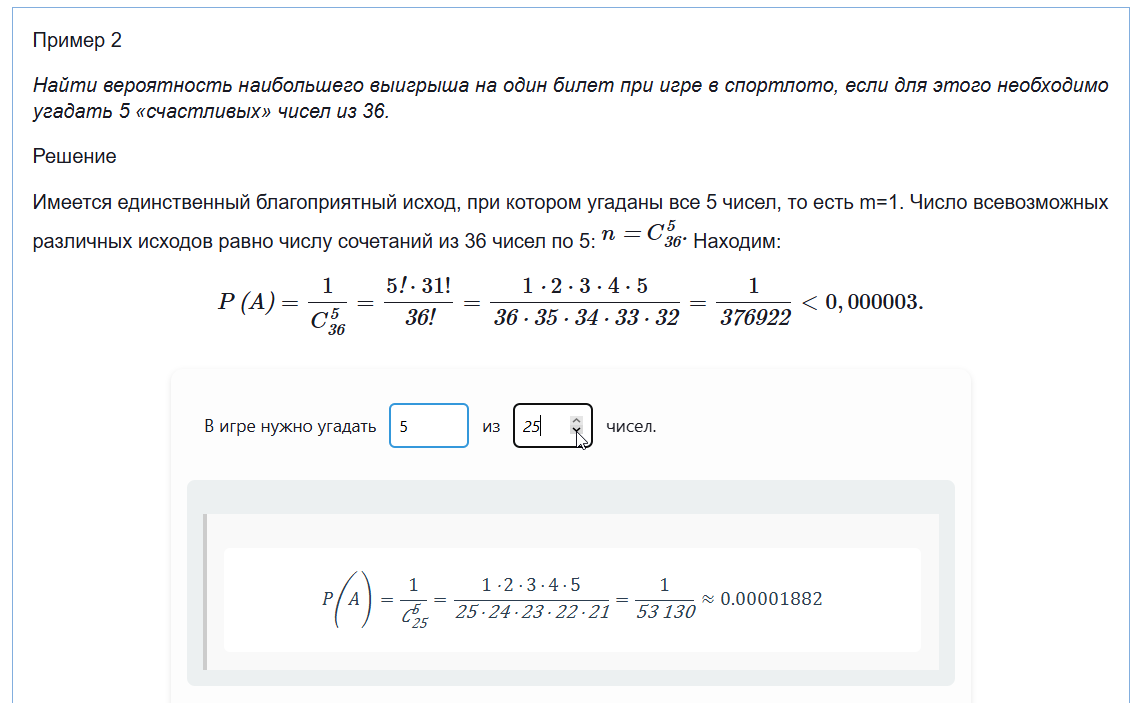


Рисунок 4 – Пример типового интерактивного блока

1. **Источник:** Файл scripts/interactive\_tasks.js
2. **Блок:** Интерактивный блок для решения задачи о выборе комитета (или аналогичной), где требуется расчет числа сочетаний и размещений.

Для демонстрации применения комбинаторных формул на конкретном примере (выбор k элементов из n с учетом и без учета порядка) мной был реализован следующий алгоритм:

1. **Инициализация:** Получение ссылок на DOM-элементы: поля ввода для n (#students) и k (#committee), а также блоки для вывода решений #combinationSolution и #permutationSolution.
2. **Функция факториала:** Используется локальная рекурсивная функция factorial(n).
3. **Валидация и ограничение ввода (clampValues):** Функция считывает значения n и k, приводит их к числовому типу, ограничивает в заданных диапазонах (n: 2-50, k: 1-n) и обновляет значения в полях ввода.
4. **Функция рендеринга формул (renderMathFormula):** Вставляет переданную MathML-строку в указанный контейнер и вызывает MathJax для ее отображения.
5. **Обновление решений (updateSolutions):**
   1. Вызывает clampValues для получения корректных n и k.
   2. Вычисляет число сочетаний C(n,k) = n! / (k! \* (n-k)!).
   3. Вычисляет число размещений A(n,k) = n! / (n-k)!.
   4. Формирует MathML-строки, отображающие формулы с подставленными значениями и результатами, и выводит их в соответствующие блоки с помощью renderMathFormula.
   5. Добавляет класс updated к блокам решений для кратковременной визуальной индикации обновления.
6. **Обработчики событий:** На поля ввода #students и #committee назначены обработчики события input, вызывающие updateSolutions.

1.6 Алгоритм работы мини-квизов mini\_quiz.js

1. **Источник:** Файл scripts/mini\_quiz.js
2. **Блок:** Блоки с мини-тестами (.quiz-question) внутри спойлеров (.quiz-spoiler).

Для интерактивной проверки усвоения материала в теоретических разделах мной была реализована функция checkAnswer, обеспечивающая немедленную обратную связь.

1. **Контекст вопроса:** Функция checkAnswer (вызываемая через onclick из HTML) определяет родительский элемент .quiz-question, чтобы работать с элементами только текущего вопроса.
2. **Получение ответа пользователя:** Внутри контекста вопроса производится поиск выбранного пользователем радио-переключателя (input[name^="q"]:checked).
3. **Проверка ответа:** Если ответ не выбран, выводится соответствующее сообщение. В противном случае, значение (value) выбранного варианта сравнивается с эталонным значением correctValue, переданным в функцию из HTML.
4. **Предоставление обратной связи:** В элемент .quiz-feedback выводится текстовое сообщение ("Правильно!" или "Неправильно...") и присваивается CSS-класс (correct или incorrect) для визуального отражения результата проверки.

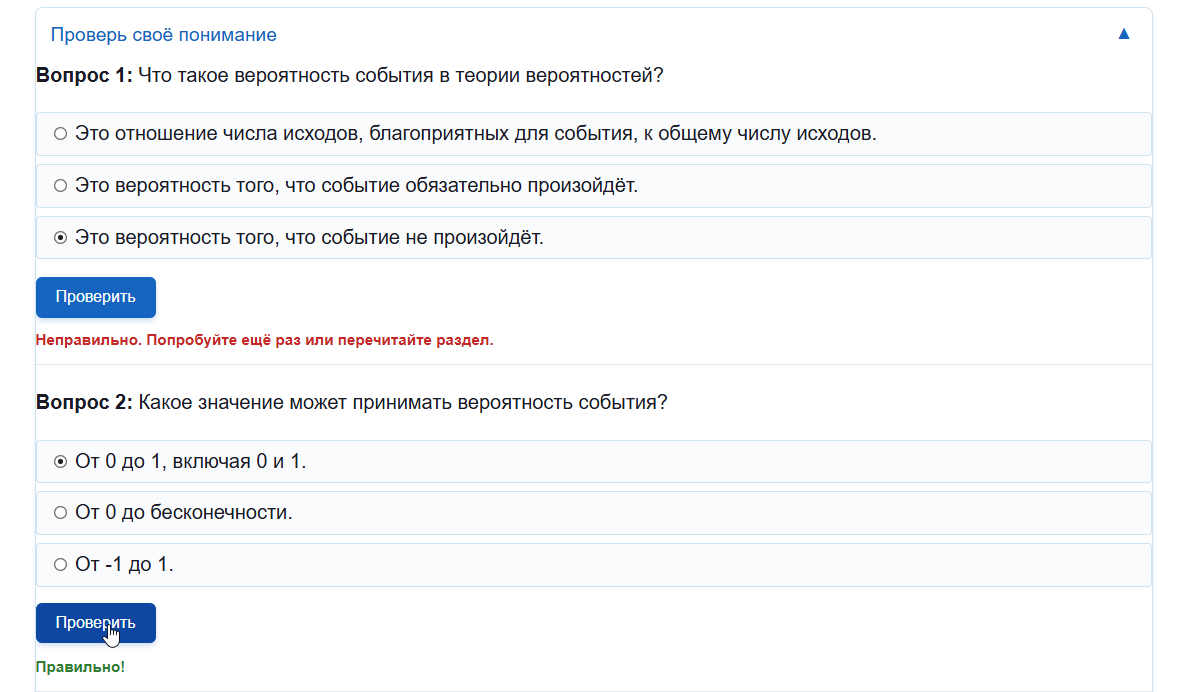


Рисунок 5 – Спойлер с мини-квизом

1.7 Алгоритм интерактивной диаграммы Венна interactive\_diagrams.js (с использованием D3.js)

1. **Источник:** Файл scripts/interactive\_diagrams.js и внешняя библиотека d3.v7.min.js.
2. **Блок:** Интерактивная диаграмма Венна (#unique-venn-diagram) в разделе "События и их классификация".

Для визуализации операций над событиями с помощью диаграммы Венна была использована библиотека D3.js. Мной был разработан следующий алгоритм для создания интерактивности:

1. **Инициализация SVG и параметров:** Задаются геометрические параметры (радиус кругов, координаты центров). С помощью D3.js выбирается SVG-элемент (#unique-venn-diagram) и создаются группы для кругов (.unique-venn-circles) и для выделенных областей (.unique-venn-overlays).
2. **Отрисовка базовых кругов:** Два базовых круга (A и B) рисуются с использованием D3.js, им задаются цвета и прозрачность.
3. **Функции генерации SVG-путей (paths):** Разработаны функции (uniqueCirclePath, uniqueIntersectionPath, uniqueAOnlyPath, uniqueBOnlyPath, uniqueUnionPath), которые возвращают строки, описывающие SVG-пути для соответствующих областей диаграммы (полный круг, пересечение, A\B, B\A, объединение). Эти функции учитывают координаты центров и радиусы.
4. **Управление выделением областей (uniqueUpdateHighlights):**
   1. Функция вызывается при изменении состояния чекбоксов (.unique-venn-checkbox).
   2. Перед отрисовкой нового выделения все предыдущие элементы из группы .unique-venn-overlays удаляются.
   3. В зависимости от того, какой чекбокс выбран (логика скрипта обеспечивает выбор только одного чекбокса единовременно), соответствующая функция генерации пути вызывается, и созданный path добавляется в группу .unique-venn-overlays с определенным цветом заливки. Атрибут fill-rule="evenodd" используется для корректной отрисовки сложных областей, таких как "A только" или "объединение".
5. **Обработчики событий:** На все чекбоксы управления диаграммой повешен обработчик события change, который сначала снимает отметки со всех чекбоксов, затем устанавливает отметку на текущем и вызывает uniqueUpdateHighlights.

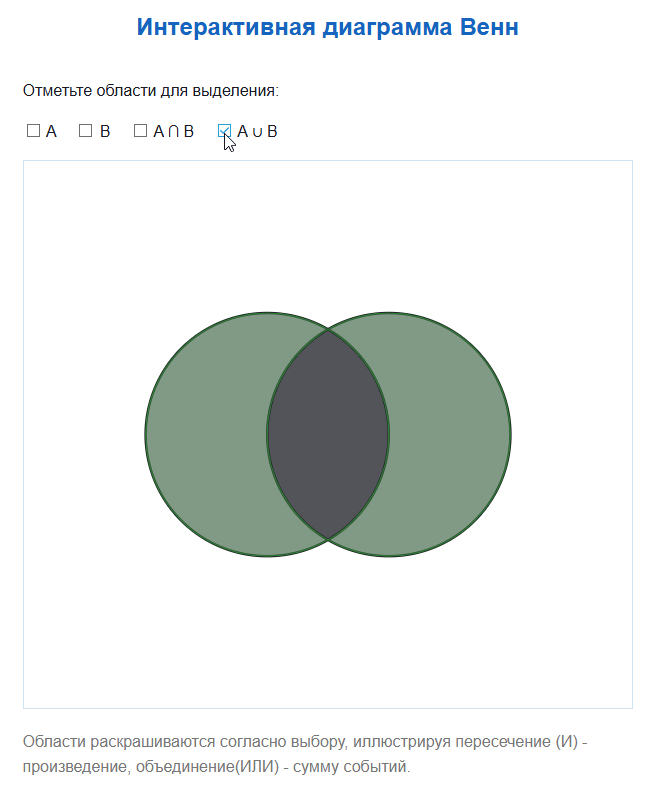


Рисунок 6 – Пример блока с визуализацией операций с пространством элементарных событий

1.8 Алгоритм симуляции вероятностных экспериментов interactive\_experiment.js

1. **Источник:** Файл scripts/interactive\_experiment.js
2. **Блок:** Интерактивный блок "Эксперимент: Бросок монеты или игральной кости".

Мной был реализован алгоритм для симуляции простых вероятностных экспериментов (бросок монеты, бросок игральной кости) с отображением результатов и накопленной статистики.

1. **Хранение статистических данных:** Используются переменные coinHeads, coinTails для подсчета исходов броска монеты и массив dieSides (6 элементов) для подсчета частоты выпадения каждой грани игральной кости. Заданы теоретические вероятности theoreticalCoinProbability и theoreticalDieProbability.
2. **Функции симуляции:**
   1. simulateCoinFlip(): Генерирует псевдослучайное число в диапазоне [0,1). Если оно меньше 0.5, результатом считается "Орел", иначе – "Решка". Соответствующий счетчик инкрементируется.
   2. simulateDieRoll(): Генерирует псевдослучайное целое число в диапазоне [1,6]. Инкрементируется соответствующий элемент массива dieSides.
3. **Обновление пользовательского интерфейса (updateResult):**
   1. Отображает результат последнего испытания (#result-value).
   2. Рассчитывает и отображает относительную частоту каждого исхода на основе накопленной статистики (#event-frequency).
   3. Отображает теоретическую вероятность для выбранного типа эксперимента (#theoretical-probability).
4. **Визуализация эксперимента (animateExperiment):** Кратковременно отображает анимированный элемент (монета или кость), добавляя и удаляя его из #experiment-animation, для имитации процесса броска.
5. **Обработчик основного действия:** Кнопка #start-experiment имеет обработчик события click, который:
   1. Определяет выбранный тип эксперимента (#experiment-type).
   2. Вызывает соответствующую функцию симуляции.
   3. Запускает анимацию.
   4. Обновляет отображаемые результаты через updateResult.

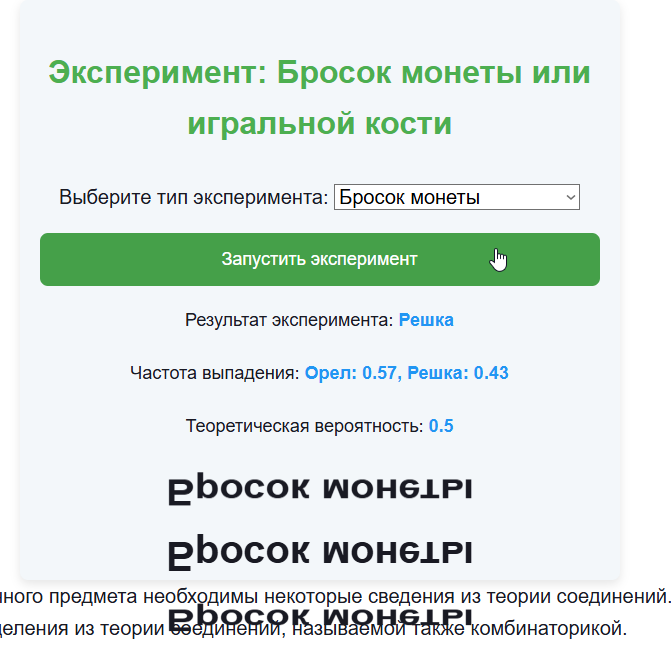


Рисунок 6 – Блок с демонстрацией сходимости экспериментальных и теоретических значений в эксперименте по броску монеты

1.9 Алгоритм расчета вероятности в лотерее lotto.js

1. **Источник:** Файл scripts/lotto.js
2. **Блок:** Интерактивный калькулятор для задачи о лото в разделе "Классическое определение вероятности".

Для интерактивного расчета вероятности выигрыша в лотерее типа "угадать k из n чисел" мной был разработан следующий алгоритм:

1. **Получение входных данных:** Считываются значения k (число угадываемых чисел, #luckyCount) и n (общее число чисел, #totalCount) из соответствующих полей ввода.
2. **Вспомогательные функции:** Для расчета используются функции вычисления факториала и числа сочетаний C(n,k). Эти функции либо реализованы в самом скрипте, либо предполагается их доступность. (Примечание: в предоставленном коде lotto.js эти функции не видны, они могут быть глобальными или импортированы, но для расчета C(n,k) они необходимы).
3. **Валидация входных данных:** Проверяется условие k <= n и другие возможные ограничения (например, положительность чисел). При некорректном вводе выводится сообщение об ошибке.
4. **Расчет вероятности:**
   1. Число благоприятных исходов принимается равным 1 (существует только одна выигрышная комбинация из k чисел).
   2. Общее число возможных исходов вычисляется как число сочетаний C(n,k).
   3. Искомая вероятность рассчитывается по классической формуле P = 1 / C(n,k).
5. **Отображение результата:** Рассчитанная вероятность выводится в блок #lottoProbabilityResult в виде дроби и/или десятичного числа. Для форматирования математического выражения используется MathJax.

1.10 Алгоритм расчета вероятности извлечения шаров balls-probability.js

1. **Источник:** Файл scripts/balls-probability.js
2. **Блок:** Интерактивный пример 4 (извлечение шаров из урны) в разделе "Теоремы сложения и умножения вероятностей".

Мной был реализован алгоритм для интерактивного расчета условной вероятности последовательного извлечения заданного количества белых шаров из урны без возвращения.

1. **Считывание параметров:** Из HTML-элементов (#whiteBalls, #redBalls, #drawCount) считываются: количество белых шаров (white), красных шаров (red) и количество извлекаемых белых шаров (draw).
2. **Валидация:** Проверяется, что draw не превышает white и общее количество шаров (total = white + red), а также что draw >= 1. При невыполнении условий в #ballsProbabilityResult выводится сообщение об ошибке.
3. **Пошаговый расчет условной вероятности:**
   1. Инициализируется итоговая вероятность probability = 1.
   2. Создаются массивы formulaNumerators и formulaDenominators для хранения числителей и знаменателей дробей на каждом шаге.
   3. В цикле от i = 0 до draw - 1:
      1. Числитель текущей дроби: num = white - i.
      2. Знаменатель текущей дроби: den = total - i.
      3. Элементы num и den добавляются в соответствующие массивы.
      4. probability умножается на num / den.
4. **Формирование и отображение результата:**
   1. Создается строка в формате MathML, представляющая произведение дробей (white / total) \* ((white-1) / (total-1)) \* ... и итоговое значение probability.
   2. Функция renderMathFormula обновляет содержимое #ballsProbabilityResult этой MathML-строкой и вызывает MathJax.typesetPromise для ее рендеринга.
5. **Обработчики событий:** На поля ввода (#whiteBalls, #redBalls, #drawCount) добавляются слушатели события input, вызывающие функцию updateBallsProbability. Также updateBallsProbability вызывается при DOMContentLoaded.

1.11 Алгоритм расчета полной вероятности для удобрений fertilizer-probability.js

1. **Источник:** Файл scripts/fertilizer-probability.js
2. **Блок:** Интерактивный пример 5 (качество удобрений) в разделе "Формула полной вероятности события".

Для интерактивного расчета полной вероятности соответствия удобрения стандарту, мной был реализован следующий алгоритм:

1. **Считывание условных вероятностей:** Из полей ввода (#p1Numerator, #p1Denominator, #p2Numerator, #p2Denominator) считываются числители и знаменатели для определения условных вероятностей P(A|B1) = p1Num / p1Den и P(A|B2) = p2Num / p2Den, где A - событие "удобрение стандартное", B1 - "удобрение из первого пункта", B2 - "удобрение из второго пункта".
2. **Задание априорных вероятностей гипотез:** Вероятности P(B1) и P(B2) жестко заданы в коде как 2/3 и 1/3 соответственно, согласно условию задачи (из первого пункта удобрений поступает в два раза больше, чем из второго).
3. **Расчет полной вероятности:** Вычисление производится по формуле полной вероятности: P(A) = P(B1) \* P(A|B1) + P(B2) \* P(A|B2).
4. **Отображение результата:**
   1. Формируется строка в формате MathML, представляющая формулу полной вероятности с подставленными значениями и итоговым результатом.
   2. Дополнительно выводится текстовое сообщение с вероятностью в процентах.
   3. Функция renderFertilizerMath обновляет содержимое #fertilizerResult этой MathML-строкой и вызывает MathJax.typesetPromise.
5. **Обработчики событий:** На поля ввода добавляются слушатели события input, вызывающие функцию updateFertilizerProbability. Функция также вызывается при DOMContentLoaded.

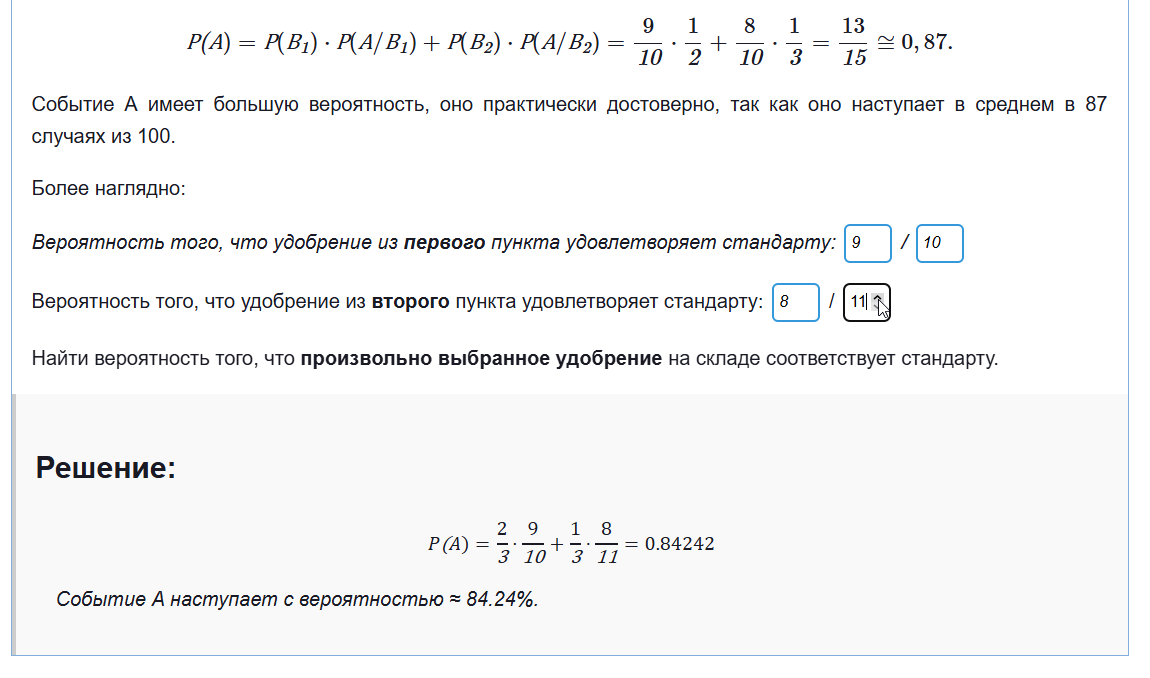


Рисунок 7 – Блок с интерактивной формулой для полной вероятности

1.12 Алгоритм расчета по формуле Байеса (задача о диетах) interactive-bayes.js

1. **Источник:** Файл scripts/interactive-bayes.js
2. **Блок:** Интерактивный пример 6 (диеты и заболевания) в разделе "Формула Байеса".

Мной был разработан алгоритм для интерактивного вычисления апостериорной вероятности с использованием формулы Байеса.

1. **Считывание условных вероятностей:** Из полей ввода (#pA\_B1, #pA\_B2) считываются значения P(A|B1) (вероятность заболевания при специальной диете) и P(A|B2) (вероятность заболевания в контрольной группе), которые затем делятся на 100 для перевода из процентов в доли.
2. **Задание априорных вероятностей гипотез:** Априорные вероятности P(B1) (принадлежность к группе со спец. диетой) и P(B2) (принадлежность к контрольной группе) заданы в коде как 0.5 каждая (группы одинаковой численности).
3. **Расчет полной вероятности события A (P(A)):** Вычисляется P(A) = P(B1) \* P(A|B1) + P(B2) \* P(A|B2).
4. **Расчет апостериорной вероятности P(B2|A):** Применяется формула Байеса: P(B2|A) = (P(B2) \* P(A|B2)) / P(A).
5. **Отображение результата:** В элемент #result выводится HTML-разметка, содержащая:
   1. Расчет полной вероятности P(A) с подставленными значениями.
   2. Расчет P(B2|A) по формуле Байеса с подставленными значениями.
   3. Итоговую вероятность P(B2|A) в процентах.
6. **Обработчики событий:** На поля ввода #pA\_B1 и #pA\_B2 добавляются слушатели события input, вызывающие функцию updateResults. Функция также вызывается при DOMContentLoaded.

1.13 Алгоритм расчета по формуле Бернулли interactive-bernulli.js

1. **Источник:** Файл scripts/interactive-bernulli.js
2. **Блок:** Интерактивный пример 7 (бракованные изделия) в разделе "Формула Бернулли".

Для интерактивного расчета вероятности по формуле Бернулли мной был реализован следующий алгоритм:

1. **Считывание параметров:** Из полей ввода (#n-input, #p-input, #k-input) считываются: n (общее число испытаний), p (вероятность "успеха" в одном испытании), k (требуемое число "успехов").
2. **Расчет вероятности "неудачи" q:** Автоматически вычисляется как q = 1 - p. Значение q отображается в поле #q-input, которое, вероятно, является readonly.
3. **Валидация k:** Проверяется, что k не превышает n. (Примечание: код для явной валидации k в предоставленном JS-файле отсутствует, но k-input имеет атрибут max="10" в HTML, который должен бы динамически обновляться).
4. **Вспомогательные функции:** Для расчета числа сочетаний C(n,k) и возведения в степень используются стандартные математические операции или встроенные функции JavaScript (Math.pow). Необходима функция факториала для C(n,k). (Примечание: явной реализации факториала или C(n,k) в interactive-bernulli.js нет, они могут быть глобальными или ожидается их наличие).
5. **Расчет по формуле Бернулли:** Pn(k) = C(n,k) \* p^k \* q^(n-k).
6. **Отображение результата:** Рассчитанная вероятность Pn(k) выводится в элемент #bernulli-result. (Примечание: в коде JS не показана сама логика расчета и вывода, а только инициализация и привязка слушателей. Предполагается, что соответствующая функция calculateBernoulli и рендеринг результата существуют).
7. **Обработчики событий:** На поля ввода добавляются слушатели события input, которые должны вызывать функцию расчета и обновления результата.

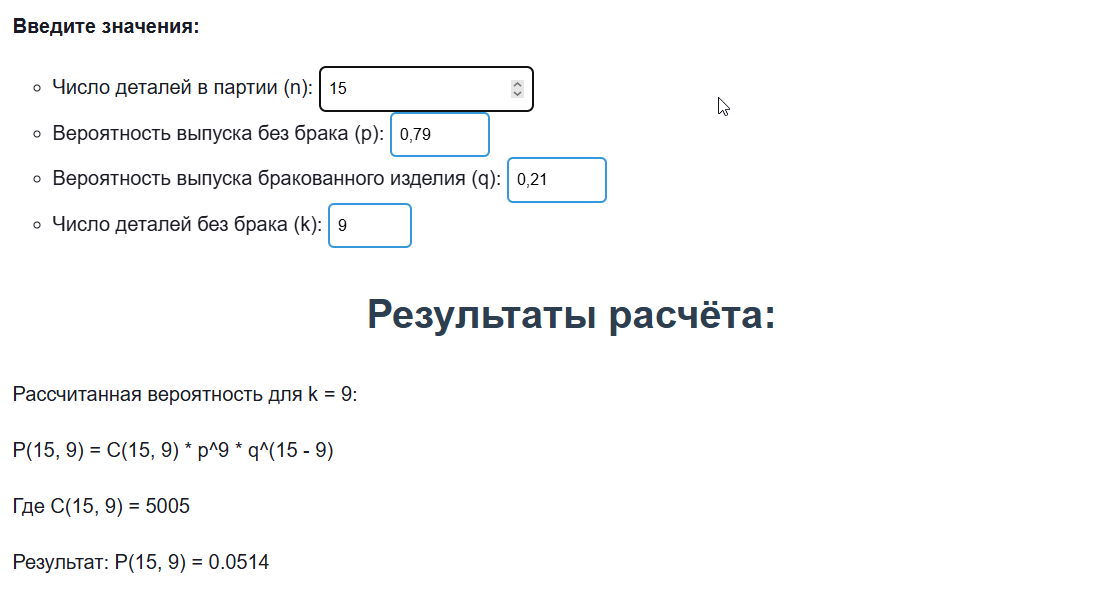


Рисунок 8 – Блок с интерактивным решением задачи по формуле Бернулли

1.14 Алгоритм раскрытия/сворачивания спойлеров toggleQuiz

1. **Источник:** Встроенный <script> в файле index.html
2. **Блок:** Элементы-спойлеры с классом quiz-spoiler.

Для обеспечения интерактивности спойлеров (плавное раскрытие/сворачивание контента) мной была реализована функция toggleQuiz.

**Идентификация элементов:** Функция toggleQuiz принимает в качестве аргумента элемент-заголовок спойлера (summaryElement), по которому был произведен клик. Затем она находит родительский элемент-спойлер (summaryElement.parentNode) и обертку для его контента (spoiler.querySelector('.quiz-content-wrapper')).

**Механизм переключения:**

**Если спойлер открыт (имеет класс open):**

1. Контентной обертке (contentWrapper) устанавливается фиксированная высота, равная ее текущей полной высоте (contentWrapper.scrollHeight + 'px').
2. Выполняется принудительная перерисовка (void contentWrapper.offsetHeight) для корректной работы CSS-перехода.
3. Высота contentWrapper изменяется на '0px', инициируя анимацию сворачивания.
4. У элемента спойлера удаляется класс open.

**Если спойлер закрыт:**

1. Элементу спойлера добавляется класс open.
2. contentWrapper.style.height устанавливается равной contentWrapper.scrollHeight + 'px', инициируя анимацию раскрытия.
3. На contentWrapper добавляется одноразовый слушатель события transitionend, который по завершении анимации устанавливает contentWrapper.style.height = 'auto', позволяя контенту занимать естественную высоту.

Заключение

В ходе прохождения учебной практики была успешно решена задача по интеграции интерактивных элементов в существующий веб-сайт, посвященный теории вероятностей и математической статистике. Основной целью работы являлось повышение образовательной эффективности ресурса и улучшение пользовательского опыта за счет внедрения динамических компонентов, способствующих лучшему усвоению теоретического материала.

Мною был разработан и реализован комплекс алгоритмов на языке JavaScript, который включает:

Интерактивные калькуляторы для основных комбинаторных формул (перестановок, сочетаний, размещений), позволяющие пользователям экспериментировать с входными данными и наблюдать результаты вычислений.

Симуляторы вероятностных экспериментов (бросок монеты, игральной кости), демонстрирующие статистические закономерности и связь теоретических вероятностей с практическими частотами.

Интерактивные визуализации, такие как диаграмма Венна для наглядного представления операций над событиями.

Механизмы для интерактивного решения задач с пошаговым отображением формул и расчетов (например, задачи на полную вероятность, формулу Байеса, формулу Бернулли).

Вспомогательные интерактивные элементы, такие как всплывающие подсказки для терминов и мини-квизы с немедленной проверкой ответов, интегрированные в теоретические разделы.

Внедрение данных интерактивных элементов позволило трансформировать сайт из преимущественно информационно-тестовой платформы в более комплексный обучающий инструмент. Пользователи получили возможность не только знакомиться с теорией и проверять свои знания, но и активно взаимодействовать с изучаемыми концепциями, что, несомненно, способствует более глубокому пониманию предмета.

В процессе работы я углубила свои практические навыки в области веб-разработки, включая манипуляцию DOM, обработку событий, применение асинхронных операций, а также интеграцию и использование сторонних JavaScript-библиотек (D3.js, MathJax). Особое внимание было уделено структурированию кода и обеспечению его работоспособности в контексте существующего проекта.

Таким образом, поставленные в рамках учебной практики цели были достигнуты. Реализованные интерактивные компоненты повысили наглядность и практическую ценность образовательного ресурса. Полученный опыт будет полезен в моей дальнейшей профессиональной деятельности и при возможном дальнейшем развитии данного проекта, которое может включать расширение спектра интерактивных задач и симуляций.

Список использованных источников

1. Флэнаган, Д. JavaScript. Подробное руководство. 7-е изд. – СПб.: Питер, 2021. – 720 с.
2. Макфарланд, Д. Новая большая книга CSS. 5-е изд. – СПб.: Питер, 2021. – 720 с.
3. Фримен, Э., Робсон, Э. Изучаем программирование на JavaScript. – СПб.: Питер, 2019. – 656 с.
4. Мюррей, С. Интерактивная визуализация данных для веба. Создание графики на JavaScript с помощью HTML, SVG и CSS. 2-е изд. – М.: Диалектика, 2018. – 496 с.
5. D3.js - Data-Driven Documents. [Электронный ресурс]. URL: [https://d3js.org/](https://www.google.com/url?sa=E&q=https%3A%2F%2Fd3js.org%2F) (дата обращения: ЧЧ.ММ.ГГГГ).
6. MathJax Documentation. [Электронный ресурс]. URL: [http://docs.mathjax.org/en/latest/index.html](https://www.google.com/url?sa=E&q=http%3A%2F%2Fdocs.mathjax.org%2Fen%2Flatest%2Findex.html) (дата обращения: ЧЧ.ММ.ГГГГ).
7. MDN Web Docs: JavaScript. [Электронный ресурс]. URL: [https://developer.mozilla.org/ru/docs/Web/JavaScript](https://www.google.com/url?sa=E&q=https%3A%2F%2Fdeveloper.mozilla.org%2Fru%2Fdocs%2FWeb%2FJavaScript) (дата обращения: 25.05.2025).
8. MDN Web Docs: HTML (HyperText Markup Language). [Электронный ресурс]. URL: [https://developer.mozilla.org/ru/docs/Web/HTML](https://www.google.com/url?sa=E&q=https%3A%2F%2Fdeveloper.mozilla.org%2Fru%2Fdocs%2FWeb%2FHTML) (дата обращения: 21.04.2025).
9. MDN Web Docs: Document Object Model (DOM). [Электронный ресурс]. URL: [https://developer.mozilla.org/ru/docs/Web/API/Document\_Object\_Model](https://www.google.com/url?sa=E&q=https%3A%2F%2Fdeveloper.mozilla.org%2Fru%2Fdocs%2FWeb%2FAPI%2FDocument_Object_Model) (дата обращения: 18.04.2025).
10. Кречман Д.Л., Пушница А.И. Разработка веб-приложений с помощью JavaScript. – М.: ДМК Пресс, 2017. – 368 с.