Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования
«Национальный исследовательский университет
«Высшая школа экономики»

Факультет Компьютерных Наук
Отделение Прикладной математики и информатики

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему

Разработка программной системы для конструирования и демонстрации модульных лекций

Выполнил студент группы 402
Карпов Алексей Дмитриевич

Научный руководитель:
к.ф.-м.н., доцент кафедры Высшей математики,
Никитин Алексей Антонович

Москва 2015

Содержание

Ключевые слова

Математическая визуализация, образование, двумерная графика, информационные технологии

1. Введение

1.1. Идея проекта. Объект, предмет работы, методы.

В рамках этой работы разрабатывается часть большого проекта VisualMath.ru. Идея этого проекта заключается во внедрении современных компьютерных и информационных технологий в классическое аудиторное образование. Проект состоит как из технической составляющей (о которой здесь идет речь), так и из методической. На текущем этапе он разрабатывается с прицелом на классический университетский курс математического анализа, не исключая возможности интеграции с другими дисциплинами школьной и университетской программ. Проект является реализацией концепции смешанного обучения (blended learning), когда образовательный курс подразумевает частичное получение материалов через интернет. Еще одно важное условие успешной реализации конкретно этого проекта заключается в возможности работы с системой с мобильных устройств: планшетов, смартфонов.

Здесь стоит оговориться, что смешанное обучение имеет два вырожденных случая. В первом случае образование полностью классическое, аудиторное и без использования интернета. Во втором случае имеет место полная зависимость от интернета (coursera и текстовые онлайн-курсы наподобие intuit.ru - близкие примеры). Разные воплощения идеи смешанного обучения могут находиться в любой точке на отрезке между классическим образованием и интернет образованием. Наш проект на этом отрезке находится в левой половине, дополняя (в то же время, не замещая собой) классический университетский курс.

Сегодня, когда интернет проникает во все области жизни (хорошие примеры: военное дело, финансы, автоматическое производство, управление, многие другие сферы), университеты по-прежнему в незначительной степени используют информационные технологи во время учебного процесса. Он широко применяется лишь в задачах менеджмента (управления). Во время непосредственно обучения интернет обычно используется для рассылки учебных пособий и приема домашних

заданий в электронном виде. В аудитории широкие возможности современных технологий зачастую не применяются. Наш проект нацелен на заполнение этой ниши, внедряя современные возможности в классический учебный процесс. Такой подход, возможно, улучшит понимание изучаемого предмета среди студентов и откроет дополнительные возможности контроля знаний.

Объектом работы является программная система для составления лекций из готовых небольших компонентов, модулей. Сама система состоит из нескольких частей, на создание и интеграцию которых были направлены усилия (предмет работы). К ним относятся:

1. Визуальные модули
2. Сервер для хранения и обмена данными

Визуальные модули можно разделить на две большие категории: двумерные и трехмерные. Предполагается разработка коллекции визуальных модулей, которая покроет классический курс математического анализа (стоит еще раз отметить, что проект не привязан ни к какому конкретному курсу, и математический анализ выбран в качестве начальной цели). Эти модули будут служить интерактивными и динамическими иллюстрациями теоремам, утверждениям и просто показывать интересные аспекты курса математического анализа. В этой работе речь идет только о двумерных модулях.

Сервер выполняет несколько функций. Помимо основной цели хранения данных, он выполняет интеграционную задачу, реализуя обмен данными между активными программами в реальном времени.

В рамках этой работы разработана библиотека, на базе которой в настоящее строится коллекция двумерных визуализаций, а также проведена интеграция этой библиотеки с сервером. В процесс интеграции входит создание на сервере подходящего для визуальных модулей программного окружения, а также реализация синхронного обновления состояния модулей у преподавателя и студентов. Когда преподаватель демонстрирует программу (визуализацию), то он меняет значения

переменных, построенные графики, масштаб, смещение графика и другие состояния. Все то же самое в режиме реального времени должно происходить и у студента, повторяя действия преподавателя.

В качестве методов создания системы были выбраны разные технологии для разных частей проекта. Сервер реализован на языке Java (фреймворк Tapestry 5), клиентская часть реализована на стандартном для этой области наборе технологии (html, css, js) с использованием open source решений для манипуляций с графикой, обмена данными, рендеринга формул в формате TeX (d3.js, socket.io, mathjax, katex).

1.2. Постановка задачи

Исходя из вышесказанного, постановка задачи была следующей: найти способ строить плоские интерактивные графики в браузере. Другой задачей была разработка сервера, на котором бы разместились визуальные текстовые модули. Задача выходит за рамки стандартной разработки статического сайта, на страницах которого были бы скриптовые программы. Сам сервер должен играть роль платформы для создания лекций и их демонстрации, а также включать в себя компоненты для мгновенного обмена информацией о своем состоянии между запущенными на нескольких устройствах визуализациями.

В ходе решения задачи построения интерактивных графиков были отброшены такие варианты, как Java Applets, чистый JavaScript, библиотека JSXGraph.

Чистый JavaScript также не подходил нашему проекту, потому что в таком случае было бы очень сложно выработать единый стиль построения графиков, а также возникли бы сложности в реализации некоторых моментов ввиду разной подготовки участников проекта.

Чистый JavaScript не может удовлетворить потребность в полном объеме, потому что в таком случае было бы сложно выработать единый стиль построения графиков, а также возникли бы сложности в реализации некоторых моментов ввиду разной предполагаемой подготовки программистов, которые создают сами графики. Все это привело бы к тому, что неизбежно пришлось бы составлять инструкции по

использованию стилей, а также поставлять базовый шаблон интерактивного графика, где уже были бы реализованы основные операции вроде перемещения графика мышкой и масштабирования колесиком мышки.

Другая технология, Java Applets, не подошла, в частности, из-за ограничений безопасности апплетов. Если программа-апплет не обладает сертифицированной цифровой подписью, то для ее просмотра необходимо вручную изменить настройки безопасности Java machine. Даже после такого действия пользователю нужно каждый раз перед просмотром программы подтверждать в браузере ее запуск. Такой подход весьма неудобен. Еще одна причина отказа от Java Applets заключается в том, что платформа сильно устарела. У нее нет современных инструментов для рисования графики, адаптированных для браузеров, и нет сообщества разработчиков, с которыми можно проконсультироваться в случае трудностей разработки.

1.3. Основной результат

Выполнена задача создания инструмента для двумерной графики. Библиотека сопровождена документацией и множеством примеров работы с ней. Программы, написанные на этой библиотеке, работают в современных браузерах для компьютеров и ноутбуков, планшетов и смартфонах. Задача интеграции с сервером также выполнена: программы, загруженные на сервер, корректно отображаются и работают синхронно в режиме демонстрации лекций. Выработан повторяемый процесс интегрирования программ с сервером.

2. Программная часть

2.1. Существующие решения

Если говорить о сервере, то готовых решений не существует. Есть отдельные компоненты, которые могут быть полезны в разработке. Например, реализация библиотеки socket.io, выполненная на языке Java. Этот компонент используется для синхронизации информации между программами и также может быть использован в

будущем для одновременного переключения страниц лекции между активными устройствами. Такой подход существенно улучшит скорость обмена по сравнению с периодическими Ajax запросами о текущем состоянии открытой лекции, которые сейчас выполняются раз в несколько миллисекунд. Кроме того, это может положительно повлиять на расходование ресурсов сервера, из-за того, что клиентские программы не будут генерировать новый запрос каждый несколько секунд, а будут использовать открытый туннель, выбранный socket.io (в современных условиях речь идет преимущественно об API WebSockets).

По причинам довольно нестандартных требований к серверу, нет возможности воспользоваться каким-то готовым решением. Напротив, на стороне клиента, в браузере, есть потенциальные альтернативы, о которых речь идет ниже.

Было просмотрено много разных решений. Все найденные библиотеки были ориентированы на отображение бизнес-данных. Специфика этой задачи в том, что не обязательно иметь полный контроль над процессом рисования, достаточно выбрать одну из стандартных форм: line-chart, bar-chart, pie-chart, donut-chart и так далее. Потом нужно передать библиотеке данные, которые могут быть просто одномерным массивом чисел. Все остальное сделает библиотека. Такой подход сильно ограничен и с трудом подходит для всего, что не входит в его основные задачи. Подобные решения блестяще работают с массивами данных, но попытка расширить возможности была бы сопряжена с большими трудностями.

Среди готовых решений, которое выполняет поставленные задачи, была найдена только библиотека JSXGraph, которая разрабатывается в университете города Байройт, Германия. Библиотека JSXGraph была наиболее подходящим кандидатом, но в течение нескольких месяцев ее использования стало ясно, что она не отвечает всем поставленным задачам. Когда этот инструмент нами использовался в 2013 году, в нем были нерешенные проблемы с масштабированием и перемещением графиков, а также проблемы со скоростью из-за большого числа внутренних зависимостей. Ниже будут продемонстрированы указанные проблемы, а также проведено сравнение с инструментом skeleton, который описывается в этой работе.

Было принято решение создать инструмент, похожий на JSXGraph по способу применения, но лишенный указанных недостатков. Идея похожести заключается в простоте API и реализации основных операций построения графиков: функции, точки, линии, закрашенные области.

2.2. Сравнение с JSXGraph

Из-за большого числа внутренних зависимостей и большого количества полей у каждого из объектов, JSXGraph демонстрировал низкую скорость работы. Например, такой простой объект как точка, требует 12 других модулей для своей работы. Как результат, добавление точек в DOM-дерево требует работы и с этими зависимостями тоже. Ниже приведена таблица, в которой сравниваются JSXGraph и skeleton.

Таблица 1

JSXGraph	2275	3787	2824	2699	2799	2683	2802	2682	2378	2792	**2772**
skeleton	275	294	287	289	286	290	283	345	286	290	**293**

В таблице приведены результаты тестирования. Проведен тест, когда на график добавлялось 1000 точек. В первой строке результаты в миллисекундах для JSXGraph, во второй строке результаты в миллисекундах для библиотеки, о которой идет речь в этой работе. В последнем столбце среднее время работы. Видно, что JSXGraph работает более чем в 9 раз медленнее.

Не наблюдается также ошибок в прорисовке некоторых графиков. Например, ниже показаны два рисунка одного и того же, выполненных в JSXGraph. Рисунки отличаются небольшим смещением области на графике, однако такое смещение приводит к ошибке в рисовании функции.

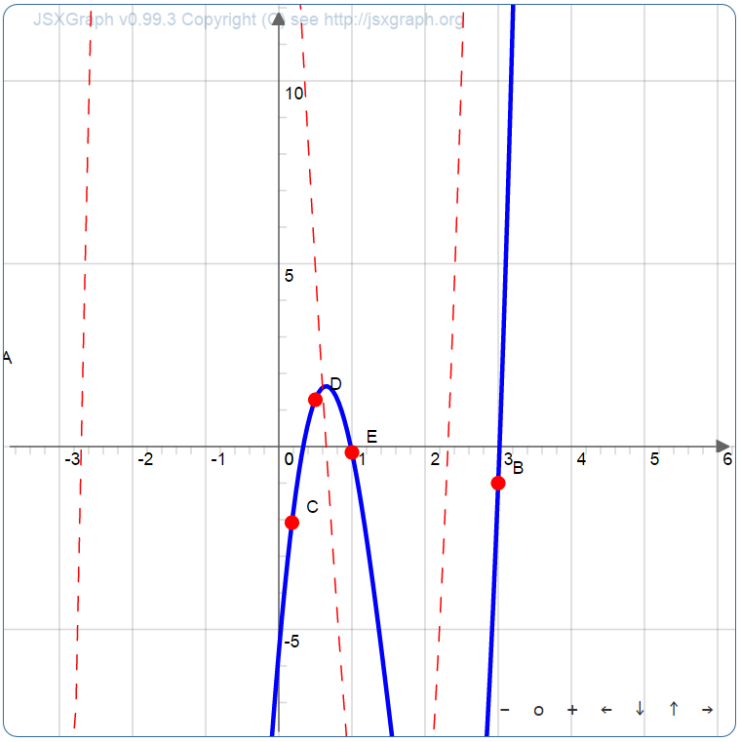

Рис. 1. Корректно нарисованный график

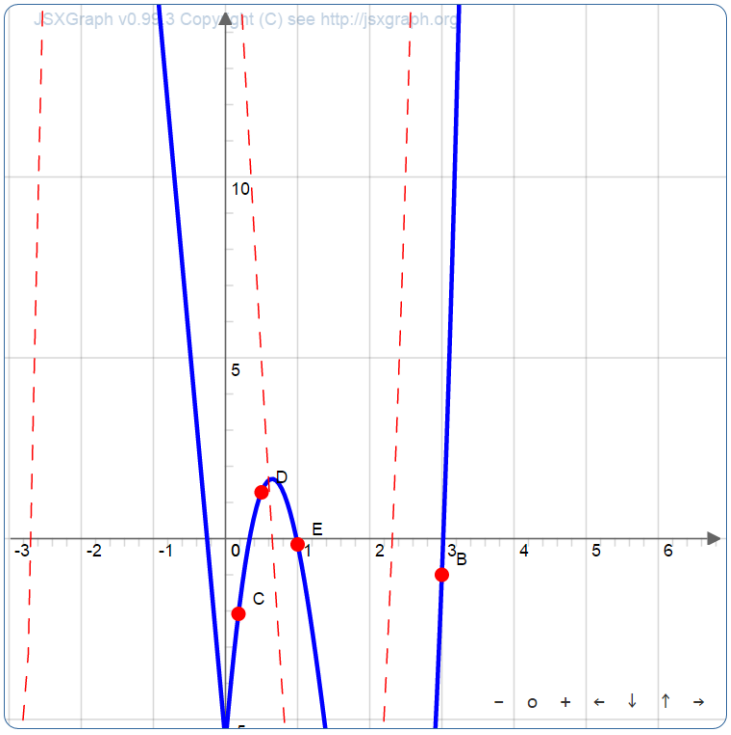

Рис. 2. Здесь происходит небольшое смещение, и в левой части графика появляется артефакт

Подобного характера ошибки исключены из инструмента skeleton, о котором идет речь. На рисунке ниже еще один пример некорректной прорисовки графика.

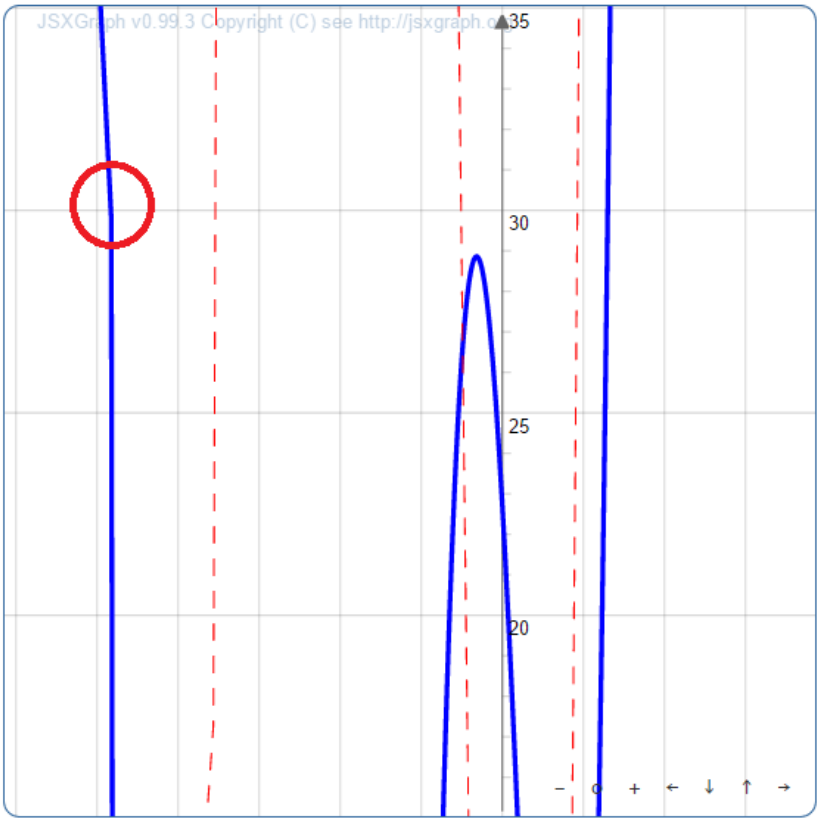

Рис. 3. Красным кружком обозначено место ошибки. График идет плавно, но в этом месте наблюдается излом.

Проблема с изломом графика наблюдалась часто. В некоторых случаях это было достаточно заметно для пользователя, чтобы оставлять эту ошибку неисправленной.

В JSXGraph нет возможности перемещать и масштабировать графики на сенсорных устройствах, а работа на стационарных компьютерах и ноутбуках затруднена, поскольку во многих программах нет возможности перемещать график

мышкой и масштабировать колесиком мыши. Это приходится делать, нажимая кнопки, которые находятся в правой нижней части графика (см. рисунки 2 и 3).

В skeleton указанные проблемы отсутствуют. Такие преимущества получены благодаря постановке конкретных задач, с которыми должна работать библиотека. Библиотека JSXGraph воплощает алгоритмы работы с двумерной графикой, когда как skeleton работает наивно. Будучи заточенным под определенный спектр задач, skeleton работает с ними лучше, чем JSXGraph. Такой подход оказывается быстрее, а также позволяет реализовать такую функцию как общение и синхронизация с сервером.

2.3. Типичный сценарий работы

Пользователями системы являются программисты, которые пишут код, опираясь на API библиотеки. Для объяснения взаимодействия с системой ниже приводятся доступные методы, а также параметры, принимаемые ими на вход.

Поскольку библиотека разрабатывается на языке CoffeeScript, который компилируется в JavaScript, то речи о полноценном объектно-ориентированном подходе идти не может. В JavaScript классы на момент написания этого текста не реализованы и даже не будут реализованы в стандарте Ecma-262 Edition 6. В этом стандарте ключевое слово «class» используется и работает похоже на классы из других языков программирования. Тем не менее, они являются *синтаксическим сахаром* над *прототипами*. Точно такая же модель используется в CoffeeScript. Далее, когда будет говориться «объект типа X», будет подразумеваться объект, образуемый от прототипа X. Есть разница в работе с классическими объектами из языков, реализующих объектно-ориентированную парадигму, и с объектами на базе прототипов. Нет особых различий в идее работы с классическими объектами и с объектами на базе прототипов, но есть некоторые существенные нюансы. Так, объекты на базе прототипов, как и классические объекты, могут быть созданы с

помощью конструктора. Этот абзац был введен специально для исключения неточности обозначения и понимания.

Пользователь создает объект класса Plotter, привязывая его к какому-то элементу DOM-дерева. С этого момента внутри соответствующего тега в HTML будет прорисовываться график посредством SVG (scalable vector graphics). Созданный объект служит строителем элементов на графике. Каждый раз, когда необходимо создать новый элемент, пользователь вызывает соответствующий метод, передает в него нужные аргументы и опциональные параметры. Метод возвращает объект, который взаимодействует с созданным SVG элементом.

2.4. Загрузка программ на сервер

Как выше сказано, выработан повторяемый процесс загрузки визуализаций на сервер. Скрипт создается в простом окружении, представляющем из себя страницу с подключенным файлом библиотеки и самой программы. После создания программы ее код переносится непосредственно на страницу, со страницы исключается файл библиотеки. Можно также опустить теги заголовка и тела страницы, оставив только те элементы, которые использует программа в процессе работы. Такими элементами могут быть различные кнопки, слайдеры, переключатели и т.д. Также обязательно есть блочный элемент, внутри которого будет прорисована визуализация. Для того чтобы гарантировать отсутствие артефактов прорисовки лекции, стоит избегать работы с телом страницы напрямую через методы манипуляции DOM-деревом.

Подготовленная страница загружается на сервер через специальную форму, где модулю с визуализацией пользователь присваивает имя и краткое описание его содержания. Точно таким же образом модуль может содержать текст, который может быть, например, теоремой, которую визуализирует программа. Модуль может не содержать презентацию вовсе, а быть лишь текстом.

Уже отмечено, что сервер отвечает не только за хранение страниц с модулями, но и предоставляет некоторые сервисы, которые помогают корректно отобразить модуль. Одной из таких вещей является подключаемая библиотека MathJax, которая

находит на странице текст в формате TeX, и заменяет его математическими формулами. Альтернативно, может использоваться katex, который удобнее в ряде случаев, чем MathJax. Другой сервис – это контейнер, который отвечает за регистрацию двумерных визуализаций и за передачу их на сервер для синхронизации с открытыми на других устройствах программами. В результате, когда преподаватель меняет на своем устройстве что-то в программе, аналогичное изменение происходит и у студентов.

Помимо создания среды для визуализаций, сервер предоставляет возможность создавать модули, которые включают в себя вопрос в формате теста. Когда преподаватель перелистывает лекцию на страницу с вопросом, он появляется на устройствах студентов. Студенты выбирают один из предложенных вариантов ответов. После того, как преподаватель завершает тест, он мгновенно получает информацию о количестве студентов, правильно и неправильно ответивших на вопрос. Такая быстрая обратная связь поможет контролировать, на каких вопросах лекции стоит остановиться подробнее.

2.5. Архитектура

Главный объект пользовательского взаимодействия – объект типа Plotter. Он выполняет две задачи: строит объекты по вызову соответствующих методов и хранит список существующих объектов. Для создания любого элемента на графике нужно вызвать какой-то метод (если не брать во внимание манипуляцию с SVG элементами напрямую или через d3.js). Каждый созданный объект помещается в список прослушиваемых объектов. Объект Plot (не путать с Plotter) стоит особняком. Такое его положение вызвано двумя причинами. Во-первых, объект Plot создается каждый раз, когда вызывается конструктор типа Plotter. Объект типа Plot отвечает за манипуляцию с сеткой графика, за регистрацию событий масштабирования и перемещения графика, а также обработку их. Во-вторых, от состояния объекта типа Plot зависят все объекты, нарисованные на графике. Когда меняется масштаб или смещение, то все остальные объекты нуждаются в перерисовке, так как необходимо пересчитать координаты на графике.

Как только в объекте типа Plot происходят изменения, которые влияют на состояние объектов на графике, вызывается событие ‘draw’, которое обрабатывается объектом типа Plotter. Он оповещает все объекты о произошедших изменениях, и те их применяют.

Любой объект на графике удаляется также через вызов метода remove объекта типа Plotter. Этот метод стирает объект из списка оповещения и вызывает у него метод clear. Подробнее в разделе 2.6.3.

2.6. Общий интерфейс объектов

2.6.1. Конструкторы

Каждый объект, который рисуется на графике, принимает в аргументах несколько обязательных аргументов. Первый аргумент – объект, хранящий свойства элемента, которые не зависят от реализации. Например, у точки такими свойствами являются ее координаты по оси абсцисс и по оси ординат. В объект «чистых» свойств в дальнейшем используется, когда нужно обратиться к соответствующим полям или поменять их. Второй аргумент – SVG-элемент в DOM-дереве, где рисуется график. Он не сохраняется внутри объекта, а используется лишь для того, чтобы создать элемент на SVG графике и привязать элемент к нему. Третий и четвертый аргументы – функции, которые переводят «чистые» координаты в координаты на графике. Например, если график строится от -5 до 5 (по оси абсцисс), а его ширина 800 пикселей, то функция в третьем аргументе будет линейно отображать [-5; 5] в [0; 800]. Четвертый аргумент тоже линейная функция, но работает для оси ординат. Они используются для расчета положения элемента на графике. Последний аргумент опциональный. Это объект, в который можно передать свойства вроде цвета, размера и так далее. Подробнее смотреть в приложении А.

2.6.2. Обновление на графике

Каждый объект также имеет метод update, который вызывается объектом типа Plotter, когда нужно обновить координаты элемента на графике.

2.6.3. Удаление с графика

Кроме того, у каждого объекта есть метод clear, который вызывается при удалении в объекте типа Plotter. Метод clear очищает DVG-элемент в DOM-дереве.

2.6.4. Цвета

Каждый объект, который обладает свойством цвета, динамически наследует методы для работы с цветом из интерфейса Colors.

2.6.5. Модели объектов

У каждого объекта есть методы setModel и getModel. Они используются для сохранения состояния объектов и для его установки.

2.6.5.1. Получение модели

Метод возвращает объект, который является моделью объекта соответствующего типа. Модель хранит в себе текущее состояние объекта. Полученная модель может быть передана в метод установки модели любого объекта соответствующего типа, чтобы присвоить ему состояние, сохраненное в модели. Также можно передать в качестве аргумента конструктора, чтобы конструктор построил объект по переданной модели.

Модель типа Plotter является составной, в отличие от моделей всех остальных объектов. Внутри она содержит модель объекта типа Plot, а также список из моделей всех нарисованных на графике объектов. Модель типа Plotter представляет полное состояние программы. Тем не менее, есть ограничения. Из-за того, что передача

моделей между клиентом и сервером и между клиентами происходит с промежуточным преобразованием в формат JSON, то накладывается ограничение на значения свойств объектов моделей. В частности, в JSON не может быть свойств-функций. Из-за этого методы объектов не добавляются в модель, которая возвращается методом Plotter.getModel().

2.6.5.2. Установка модели

Устанавливает у уже созданного объекта состояние. Первый аргумент обязателен и должен соответствовать возвращаемому объекту из метода получения модели. В объекте второго аргумента возможно указать, чтобы установка модели не вызывала обновление рисунка объекта.

Установка модели у объекта типа Plotter через этот метод вызовет рекурсивную установку моделей у всех нарисованных объектов. Внутри модели и внутри объекта типа Plotter модели нарисованных объектов должны быть перечислены в массиве в одинаковом порядке.

2.7. Особенности некоторых объектов

2.7.1. Поведение точки

Объект типа Point имеет набор возможных поведений в ответ на действия пользователя. По умолчанию используется поведение типа ‘free’. В этом случае точка движется вслед за курсором, когда пользователь перетаскивает ее. Дополнительно можно прописать методы движения вдоль прямой или вдоль графика какой-то функции.

2.7.2. Зависимость объекта типа Plotter от внешнего контейнера

Если в глобально области видимости есть объект типа OverContainer с именем overContainer, то объект типа Plotter в конструкторе вызовет метод overContainer.add,

передав в качестве аргумента this, то есть самого себя. Согласно требованию объекта типа OverContainer, объект типа Plotter вызывает на себе событие ‘drawn’ после того, как происходит прорисовка всего графика.

2.8. Общение с сервером

2.8.1. OverContainer

Объект типа OverContainer, который был упомянут в прошлом разделе, собирает и хранит в себе объекты, состояния которых нужно синхронизировать между устройством преподавателя и студента. Эти объекты должны вызывать на себе событие ‘drawn’, когда поменялось их внутреннее состояние и требуется синхронизация.

У объекта типа OverContainer есть методы setModel и getModel, которые автоматически вызывают такие же методы у всех объектов, которые хранятся внутри overContainer. Метод OverContainer.setModel принимает массив, который должен быть сгенерирован OverContainer.getModel. Метод OverContainer.getModel возвращает список моделей, отображая список объектов, хранящихся внутри себя, на модели посредством вызова метода getModel у каждого из объектов.

Введение объекта типа OverContainer обусловлено тем, что состояние программы, написанной с использованием библиотеки, не всегда зависит только от самого графика. Также могут быть элементы управления. В типичном сценарии работы с объектом типа OverContainer в нем автоматически регистрируется объект типа Plotter. Дальше нужно зарегистрировать в нем остальные объекты с элементами управления. Они должны вызывать событие ‘drawn’ и иметь методы getModel и setModel.

Конструктор OverContainer принимает единственный аргумент. Если это строка типа ‘sender’, то сконструированный объект будет вызывать метод Gate.send. Эта строка сохраняется в свойстве type. Другое ее возможное значение – ‘receiver’. Подробнее в следующем разделе.

2.8.2. Gate

Объект типа Gate отвечает за непосредственно отправку и получение сообщений с сервера. В конструкторе он принимает обязательный аргумент – объект типа OverContainer (описан в предыдущем разделе). Если свойство overContainer.type равно ‘receiver’, то объект типа Gate каждый раз при получении данных с сервера будет вызывать метод OverContainer.setModel, инициируя тем самым цепочку вызова методов setModel у всех объектов, попадающих под обновление.

Метод Gate.send собирает данные из overContainer и отправляет их на сервер. Общение с сервером осуществляется посредством библиотеки socket.io.

2.9. Выводы

Приведено описание внутренней структуры библиотеки и описаны ее особенности. В том числе, освещен метод взаимодействия с сервером и синхронизации данных. Дальше речь пойдет о результатах разработки, о внешнем виде и сравнении с JSXGraph.

3. Результаты и заключение

3.1. Внешний вид

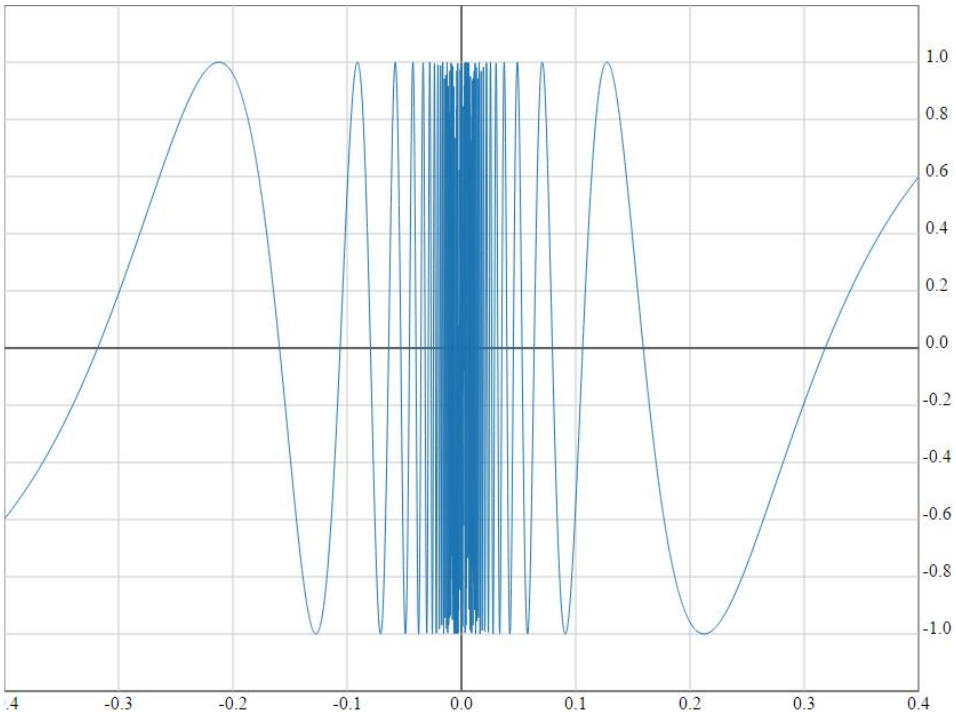

Рис.4. Пример внешнего вида программы. Здесь нарисован график функции sin(1/x).

По умолчанию рисуется график на белом фоне с подписями осей снизу и справа. Сетка представляет собой тонкие линии ярко-серого цвета, оси выделены более темным цветом и большей толщиной. Есть возможность поменять цвета на любые другие, смотреть приложение А. График отзывчивый, плавно реагирует на его перемещение и масштабирование пользователем. Легко встраивается в любой блочный элемент HTML кода.

Библиотека справляется с поставленными задачами. Тем не менее, есть пути для дальнейшего ее развития. В будущем есть планы сделать несколько улучшений, о которых речь идет ниже.

3.2. Примеры программ

В этом разделе приведены примеры разных программ, реализованных с помощью инструмента.

3.2.1. Функция Римана

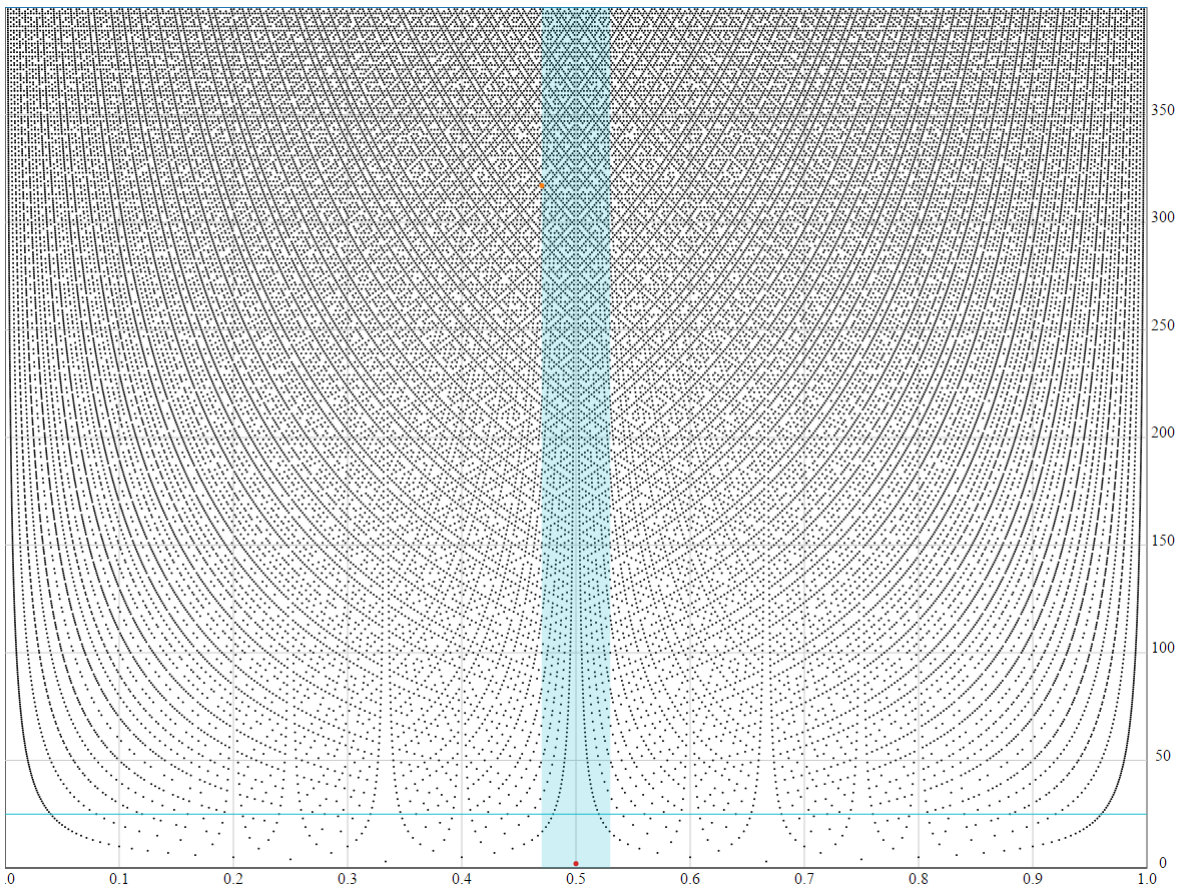

Рис. 5. Функция Римана

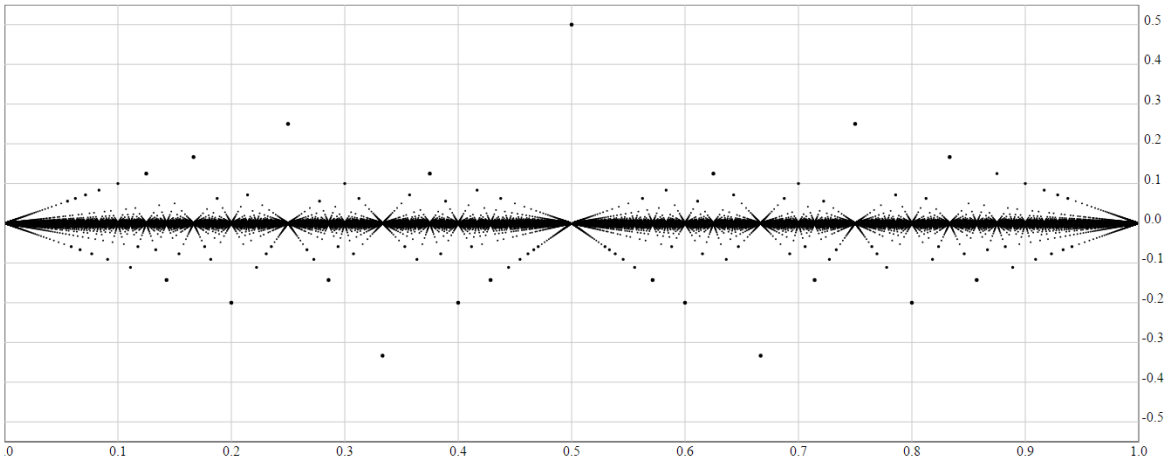

Рис. 6. Модифицированная функция Римана

Строится последовательность точек, отвечающих функции Римана (равная нулю в иррациональных точках и $1/q$ в точках вида p/q).

3.2.2. Последовательности

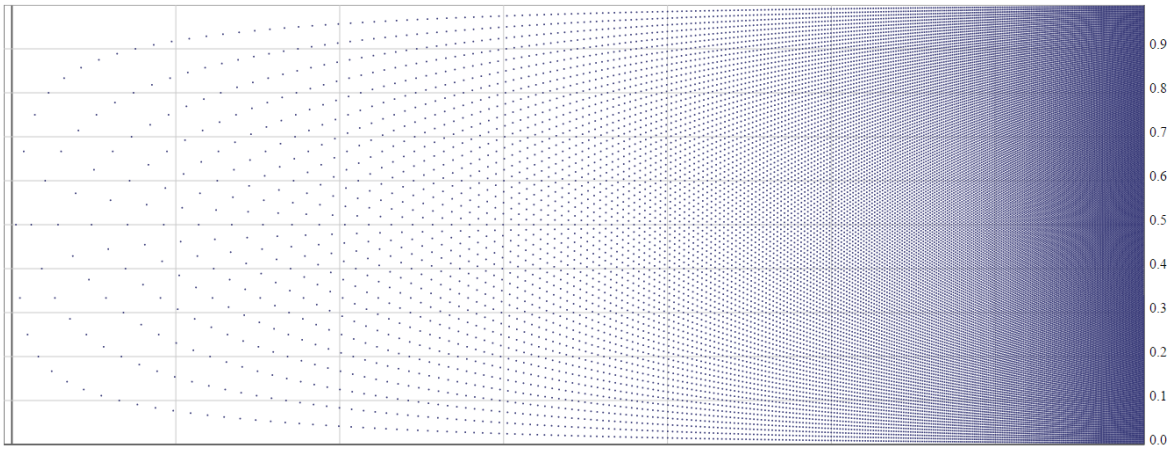

Рис. 7. Последовательность точек {1/2, 1/3, 2/3, 1/4} в логарифмической шкале

График может строиться как в линейной, так и в логарифмической шкале. Существует опция, которая запустит анимацию последовательного построения точек.

3.2.3. Константы при вычислении интеграла

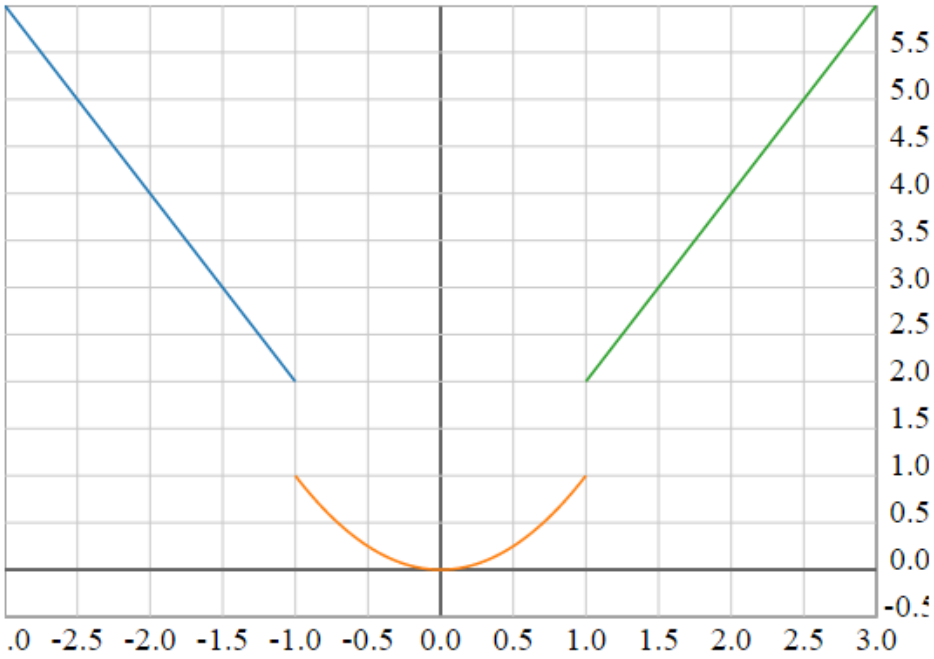

Рис. 8.

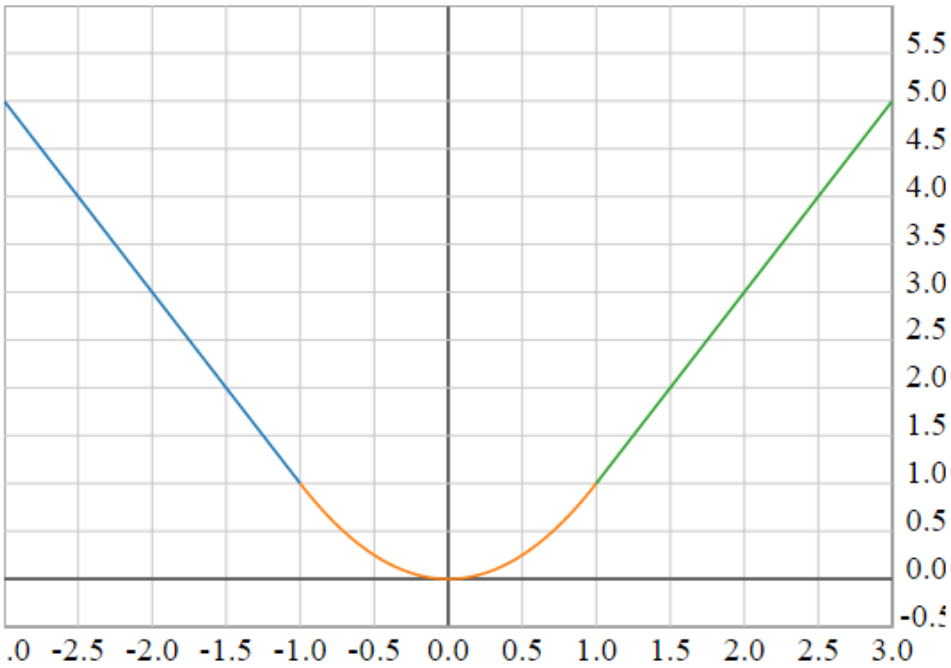

Рис. 9.

На рисунках 8 и 9 показана одна и та же программа. Это график первообразной функции. Слева от -1 функция имеет значение -1, справа от +1 функция имеет значение +1. В отрезке [-1; 1] функция имеет вид $y = x$. В первом случае показано, как выглядит график, если при взятии интеграла не было учтено требование непрерывности первообразной. Во втором случае показано, как график первообразной выглядит при правильном нахождении констант. При запуске программы график функции плавно меняется от одного вида к другому.

3.2.4. Функция Кантора

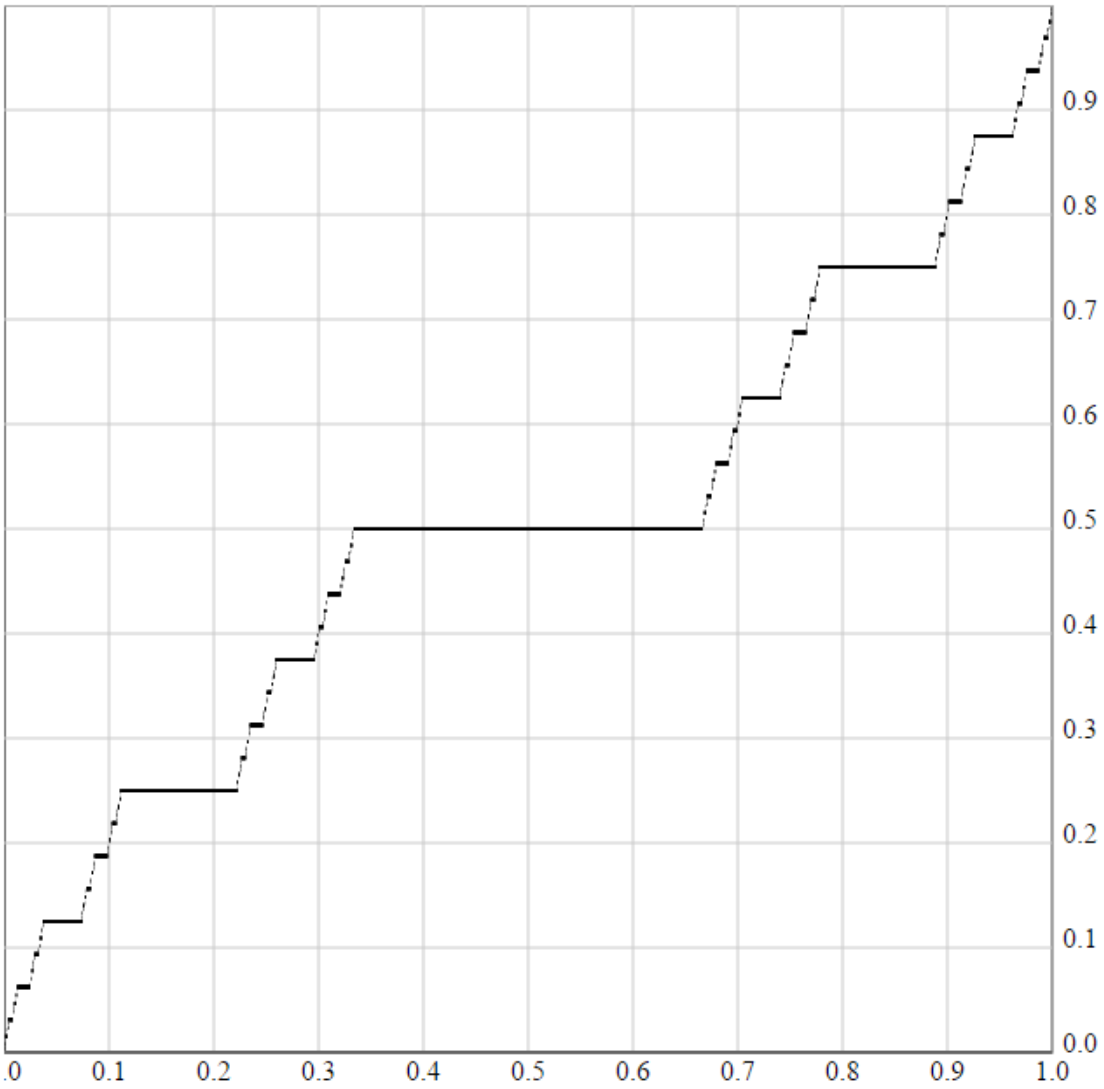

Рис. 10.

Программа рисует функцию Кантора. Предусмотрено масштабирование, которое показывает, что функция имеет одинаковую фрактальную структуру на любом уровне приближения.

3.3. EcmaScript 6

Сейчас библиотека написана на языке CoffeeScript. Он во время разработки зарекомендовал себя в некоторых местах не с самой лучшей стороны. Например, существуют проблемы в компиляторе языка, который не обрабатывает некоторые корректные конструкции. В некоторых случаях возникают ошибки компиляции, которые приводят к невозможности исполнить программу. Наконец, отладка программ с использованием карт исходного кода в случае этого языка бывает затруднительной из-за высокой абстрактности некоторых конструкций. Такие конструкции включают в себя сразу несколько строк кода в скомпилированном виде, либо используют синтаксис выражений. В таком случае создается отдельная область видимости для каждого блока кода вроде циклов или операторов ветвления. Следующий стандарт языка JavaScript (называемый EcmaScript 6) видится лучшей альтернативой, лишенный приведенных недостатков при компиляции практически полностью. Возможен постепенный перевод на другой язык, и это со временем положительно скажется на разработке.

3.4. Устранение внешних зависимостей

Если убрать внешние зависимости, такие как d3.js, lodash и некоторые другие, то это поможет существенно сократить размер файла библиотеки после сборки. Речь идет о сокращении в разы. Возможно, даже в десятки раз. Выгода от такого улучшения очевидна: скрипт быстрее будет загружаться у конечного пользователя.

3.5. Рендеринг с помощью WebGL

Сейчас есть проблемы со скоростью обновления графика при нескольких тысячах нарисованных объектов. И хотя это уже примерно в 10 раз быстрее, чем у JSXGraph (смотреть раздел 2.2.), этого не достаточно для десятков тысяч объектов.

Проблема заключается в скорости работы с DOM-деревом, и это общая проблема всего HTML API. Если рисовать графики с помощью WebGL, то HTML представление страницы не затрагивается, а само рисование происходит, в том числе, с помощью видеопроцессора. Такой подход поможет ускорить библиотеку в десятки или даже в сотни раз.

5. Библиографический список

Д. Флэнаган JavaScript. Подробное руководство. - 6 изд. - СПб.: Символ-Плюс, 2012.

Draft Specification for ES.next (Ecma-262 Edition 6) // wiki.ecmascript.org URL: http://wiki.ecmascript.org/lib/exe/fetch.php?id=harmony%3Aspecification_drafts&cache=cache&media=harmony:ecma-262_6th_edition_final_draft_-04-14-15.pdf (дата обращения: 18.05.2015).

Document Object Model (DOM) // Mozilla Developer Network URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model (дата обращения: 18.05.2015).

SVG // Mozilla Developer Network URL: https://developer.mozilla.org/ru/docs/Web/SVG (дата обращения: 18.05.2015).

Semantic Versioning 2.0.0 // Semantic Versioning 2.0.0 URL: http://semver.org/ (дата обращения: 18.05.2015).

Приемы объектно-ориентированного проектирования. Паттерны проектирования Подробнее: http://www.labirint.ru/books/87603/ / Гамма, Хелм, Джонсон, Влиссидес, Влиссидес Дж. и др.; под ред. Шалаев Н. - ISBN: 978-5-4590-1720-5 изд. - СПб: Питер, 2013.

Приложение A. Описание API.

1. Skeleton

Другое имя для **Plotter.**

i. Plotter

Главный объект, с которым нужно работать программисту, - это объект типа Plotter. Объект этого типа является основным. Он отвечает за прорисовку графика и всех объектов, которые на нем находятся. Для создания графика нужно передать в качестве первого аргумента в конструктор Plotter id *DOM-элемента*, где дальше будет рисоваться график.

Второй необязательный аргумент конструктора Plotter options -- это объект, в котором можно менять некоторые свойства объектов типа Plot и PlotPure, которые хранятся внутри объекта типа Plotter как объекты, содержащие информацию о деталях рисования на странице. Объекты типов Plot и PlotPure используется для хранения основных данных о графике, а также для основных действий над графиком. Ниже приведены доступные свойства, а также их значения по умолчанию.

Здесь перечислены свойства, которые могут быть переданы в объекте во втором необязательном аргументе конструктора Plotter. Все указанные здесь свойства, относящиеся к Plot и PlotPure, будут переданы в конструкторы соответствующих объектов. Иначе говоря, конструктор Plotter служит оберткой для конструкторов Plot и PlotPure.

1. PlotPure

Та часть свойств, которая описана здесь, отражает состояние графика как сущности, которая описывает состояние абстрактного графика. Эти свойства могут быть независимыми от реализации, при этом их достаточно, чтобы на этой основе

сделать любую другую реализацию. Подробнее об объектах типов –Pure в разделе с описание архитектуры библиотеки.

left: -4, левая граница рисуемого графика

right: 4, правая граница рисуемого графика

bottom: -3, нижняя граница рисуемого графика

top: 3, верхняя граница рисуемого графика

2. Plot

Свойства, описанные здесь, являются зависимыми от реализации. В самых общих словах, они описывают внешность и поведение объекта графика, но не его главные свойства, которые хранятся в PlotPure. Подробнее о разделении ответственности объектов и их взаимодействии также написано в разделе с описание архитектуры библиотеки.

width: 800, ширина области рисования в пикселях

height: 600, высота области рисования в пикселях

onDrawCallback: null, *функция обратного вызова*, которая вызывается каждый раз, когда происходит полная перерисовка графика. Такое происходит, например, при масштабировании и смещении графика. Каждый раз при вызове функции туда передается единственный аргумент: объект типа Plot (отличный от Plotter тип). Этот объект хранит в себе все описанные в этом разделе свойства, а также объект типа PlotPure, который также здесь описан и хранит границы графика.

zoom: true, при значении true возможность масштабирования включена. При значении false возможность масштабирования отключена.

stroke: "#000000", цвет рамки графика

vertical: 20, количество дополнительных пикселей *svg-элемента* снизу от графика, оставляемых для подписи оси числами

horizontal: 30, количество дополнительных пикселей *svg-элемента* справа от графика, оставляемых для подписи оси числами

ticks: 10, частота координатной сетки как по вертикали, так и по горизонтали. Иначе говоря, количество ячеек по вертикали и горизонтали.

transformX: функция, делающая преобразование числового значения из пользовательского диапазона в программный. В основе лежит линейная функция, вывод которой форматируется согласно требованиям SVG. Эта функция работает применительно к оси абсцисс. Как пример, если начальное значение левой границы -4, а у правой границы +4, и ширина графика 800 пикселей, то функция будет линейно отображать отрезок [-4; 4] на отрезок [0; 800]. При изменении масштаба и смещения графика функция автоматически меняется.

transformY: функция, делающая преобразование числового значения из пользовательского диапазона в программный. Работает аналогично предыдущему, но применительно к оси ординат.

lineWidth: функция, возвращающая «2» для нуля и «1» для всех остальных значений, используется для задачи толщины координатной сетки. По умолчанию прямые, проходящие через 0, выделяются толщиной 2.

lineColor: функция, возвращающая цвет #666000 для нуля и #ccc000 для всех остальных значений, используется для лучшего выделения прямых, проходящих через 0.

ii. Методы Plotter

Объект типа Plotter имеет методы для рисования объектов на координатной плоскости. Каждый метод для рисования возвращает объект, который содержит в себе методы для изменения состояния нарисованного объекта. Таким образом,

созданные графики не статичны, а могут меняться при условиях, описанных программистом. У объекта типа Plotter существуют методы для добавления закрашенной области, линии, точки и функции.

Объект типа Plotter предоставляет главные методы работы с библиотекой. Помимо этого, он хранит в себе все созданные на графике объекты и сообщает им, когда нужно сделать обновление координат на графике.

1. Plotter.func(Function, Object)

Другое имя для Plotter.addFunc().

2. Plotter.addFunc(Function, Object)

Первый аргумент - это функция, которую нужно нарисовать. Нужно передать функцию обратного вызова, которая принимает один числовой параметр (значение x) и возвращает тоже числовой параметр (значение y). Второй необязательный аргумент - это объект, в котором можно менять некоторые свойства.

Кроме того, функция addFunc также может принимать единственный аргумент: модель объекта типа Func, по которой можно восстановить его начальное состояние. Подробнее об этом в описании методов Func.getModel() и Func.setModel().

Метод Plotter.addFunc возвращает объект типа Func, в котором есть методы для изменения состояния графика функции.

accuracy: 800, точность прорисовки графика. По умолчанию равна ширине графика в пикселях. Количество точек, по которым строится график.

strokeWidth: 2, толщина линии графика

color: 0, цвет. Нулевой цвет - это синий. Если менять это свойство вручную, без вызова метода setColor, то нужно указывать цвет в формате rgb с символом решетки в начале шестизначной последовательности. Лучше не менять это свойство напрямую, а использовать сеттер, так как в нем можно выбрать цвет из заранее заданного набора. Заранее созданный набор состоит из 21 цвета, которые нумеруются от 0 до 20.

breaks: [], массив точек разрыва (по умолчанию пустой массив). Когда строится график функции, у которой есть разрывы, то их нужно указать в этом свойстве простым перечислением в массиве. Например, у графика сигнума точкой разрыва будет 0. Такая же точка разрыва будет у гиперболы и у графика sin(x)/x. Внутри библиотеки не реализован способ надежного автоматического определения точек разрыва. Это и не является целью библиотеки, поскольку она выступает не в качестве инструмента для анализа функций, а лишь средством описания того, что необходимо программисту (человеку, которому требуется визуализация).

left: null, левая граница прорисовки. Левее нее график никогда не будет рисоваться. null означает, что границы нет.

right: null, правая граница прорисовки. Правее нее график никогда не будет рисоваться. null означает, что границы нет.

fill: ‘none’, в объекте функции это свойство не используется. Отвечает за цвет закрашивания области под графиком функции. Это свойство необходимо для других объектов, которые наследуются от прототипа объекта функции. Тем не менее, может быть назначено, и тогда функция будет вести себя как закрашенная область.

fillOpacity: null, отвечает за прозрачность области под графиком функции. Аналогично предыдущему свойству, не используется в объекте функции. Как и предыдущее свойство, может быть назначено.

3. Plotter.addLine(x1, y1, x2, y2, options)

Первые четыре аргумента (числа) задают координаты двух точек, по которым будет строиться отрезок. Последний необязательный аргумент - это объект, в котором можно передавать свойства. Ниже описаны все доступные для объекта свойства. Слева указано имя свойства, а справа - его значение по умолчанию.

Кроме того, метод Plotter.addLine также может принимать единственный аргумент: модель объекта типа Line, по которой можно восстановить его начальное состояние. Подробнее об этом в описании методов Line.getModel() и Line.setModel().

Метод Plotter.addLine возвращает объект типа Line, в котором есть методы для манипуляции состоянием уже построенной линии.

strokeWidth: 2, толщина линии

color: 0, цвет линии. Нуль соответствует синему цвету. Если менять это свойство вручную, без вызова метода setColor, то нужно указывать цвет в формате rgb с символом решетки в начале шестизначной последовательности. Лучше не менять это свойство напрямую, а использовать сеттер, так как в нем можно выбрать цвет из заранее заданного набора. Заранее созданный набор состоит из 21 цвета, которые нумеруются от 0 до 20.

4. Plotter.area(Array, Object)

Другое имя для Plotter.addArea().

5. Plotter.addArea(Array, Object)

Строит произвольную закрашенную область на графике. Для построения нужно передать в качестве первого аргумента массив точек. Каждая точка - это объект, у которого есть два свойства: x и y – с численными значениями.

Второй необязательный аргумент метода Plotter.addArea - объект со свойствами. Ниже перечислены все доступные свойства, а также их значения по умолчанию.

Кроме того, метод Plotter.addArea также может принимать единственный аргумент: модель объекта типа Area, по которой можно восстановить его начальное состояние. Подробнее об этом в описании методов Area.getModel() и Area.setModel().

Метод Plotter.addArea возвращает объект типа **ParametricArray**, в котором есть методы для манипуляции состоянием уже построенной области.

color: 20, цвет границы области. Число 20 отвечает черному цвету. Если менять это свойство вручную, без вызова метода setColor, то нужно указывать цвет в формате rgb с символом решетки в начале шестизначной последовательности. Лучше не менять это свойство напрямую, а использовать сеттер, так как в нем можно выбрать цвет из заранее заданного набора. Заранее созданный набор состоит из 21 цвета, которые нумеруются от 0 до 20.

fillOpacity: 0.2, прозрачность закраски области. Может меняться от 0 до 1, где 1 - непрозрачная, а 0 - полностью прозрачная, невидимая.

strokeWidth: 0, толщина границы области. По умолчанию толщина нулевая, то есть граница невидимая

fill: 1, цвет закраски области. Единица соответствует голубому цвету.

6. Plotter.point(x, y, options)

Другое имя для Plotter.addPoint().

7. Plotter.addPoint(x, y, Object)

Рисует точку по указанным координатам. Третьим необязательным аргументом можно передать некоторые свойства, которые указаны ниже.

Кроме того, метод Plotter.addPoint также может принимать единственный аргумент: модель объекта типа Area, по которой можно восстановить его начальное состояние. Подробнее об этом в описании методов Point.getModel() и Point.setModel().

Метод Plotter.addPoint() возвращает объект типа Point, в котором есть методы для манипуляции состоянием уже построенной области.

movable: false, можно ли точку перетаскивать мышкой. По умолчанию нельзя. Если поставить значение true, то точка станет доступна для перемещения мышкой.

color: 6, цвет точки. Число 6 отвечает красному цвету. Если менять это свойство вручную, без вызова метода setColor, то нужно указывать цвет в формате rgb с символом решетки в начале шестизначной последовательности. Лучше не менять это свойство напрямую, а использовать сеттер, так как в нем можно выбрать цвет из заранее заданного набора. Заранее созданный набор состоит из 21 цвета, которые нумеруются от 0 до 20.

size: 3, радиус точки в пикселях.

behaviorType: 'free', тип поведения точки при перетаскивании. В настоящее время при указании любого другого значения, кроме значения по умолчанию, приведет к тому, что точка не будет перемещаться даже в случае, если свойство movable установлено как true. В будущем будет возможно менять значение свойства на другие, определяя альтернативное поведение точки. Например, перемещение вдоль прямых и перемещение вдоль функций.

onMove: null, функция обратного вызова, которая вызывается каждый раз, когда пользователь перемещает точку мышкой. В функцию передается два аргумента: положение точки по оси абсцисс и по оси ординат. Если при создании точки это свойство не равно null, то свойство movable автоматически устанавливается равным true. Если все-таки необходимо сделать точку

обездвиженной, то необходимо после ее создания отдельно присвоить свойству movable значение false.

8. Plotter.shadedArea(Function, Object)

Другое имя для Plotter.addShadedArea().

9. Plotter.addShadedArea(Function, Object)

Первый аргумент - это функция, которую нужно нарисовать и под графиком которой закрасить область. Нужно передать функцию обратного вызова, которая принимает один числовой параметр (значение x) и возвращает тоже числовой параметр (значение y). Второй необязательный аргумент - это объект, в котором можно менять некоторые свойства.

Кроме того, функция addShadedArea также может принимать единственный аргумент: модель объекта типа **Func**, по которой можно восстановить его начальное состояние. Подробнее об этом в описании методов **Func.getModel()** и **Func.setModel().**

Метод Plotter.addShadedArea возвращает объект типа **Func**, в котором есть методы для изменения состояния графика функции.

Plotter.addShadedArea ведет себя во многом как Plotter.addFunc, возвращая объект того же типа Func. addShadedArea, по сути, является расширением addFunc, возвращая объект с тем же интерфейсом, дополняя его. Некоторые свойства имеют другое значение по умолчанию, в отличие от Func, а некоторые трактуются иначе. Кроме того, есть новое свойство axe. Ниже описаны свойства, которые работают иначе, чем в объектах типа Func или отсутствуют. Все остальные свойства, которые есть у addFunc, но не перечислены здесь, работаю так же, как это написано в описании addFunc.

strokeWidth: 0, толщина линии графика. По умолчанию сама функция не рисуется в ShadedArea.

left: null, левая граница прорисовки. Левее нее график никогда не будет рисоваться. null означает, что границы нет. Область под графиком будет закрашена до левой границы, если она не равна null.

right: null, правая граница прорисовки. Правее нее график никогда не будет рисоваться. null означает, что границы нет. Область под графиком будет закрашена до правой границы, если она не равна null.

fill: 1, отвечает за цвет закрашивания области под графиком функции.

fillOpacity: 0.3, отвечает за прозрачность области под графиком функции.

axe: 'x', ось, к которой прилегает закрашенная область. По умолчанию область заполняет пространство между графиком функции и осью OX. Если указать значение 'y', то область заполнит пространство между графиком функции и отрезком [(0, func(left)), (0, func(right))], где func – функция обратного вызова, переданная в конструктор, left – левая граница (свойство), right – правая граница (свойство).

10. Plotter.parametricFunc(Array, Object)

Другое имя для Plotter.addParametricFunc().

11. Plotter.addParametricFunc(Array, Object)

Строит произвольную кривую на графике. Для построения нужно передать в качестве первого аргумента массив точек. Каждая точка - это объект, у которого есть два свойства: x и y – с численными значениями.

Второй необязательный аргумент метода Plotter.addParametricFunc - объект со свойствами. Ниже перечислены все доступные свойства, а также их значения по умолчанию.

Метод Plotter.addParametricFunc возвращает объект типа ParametricFunc, в котором есть методы для манипуляции состоянием уже построенной кривой.

color: 0, цвет границы области. Число 0 отвечает синему цвету. Если менять это свойство вручную, без вызова метода setColor, то нужно указывать цвет в формате rgb с символом решетки в начале шестизначной последовательности. Лучше не менять это свойство напрямую, а использовать сеттер, так как в нем можно выбрать цвет из заранее заданного набора. Заранее созданный набор состоит из 21 цвета, которые нумеруются от 0 до 20.

fillOpacity: 0, прозрачность закраски области. Может меняться от 0 до 1, где 1 - непрозрачная, а 0 - полностью прозрачная, невидимая.

strokeWidth: 2, толщина границы области.

fill: 1, цвет закраски области. Единица соответствует голубому цвету.

12. Plotter.redraw()

Альтернативное имя для Plotter.draw()

13. Plotter.draw()

Рисует график и все объекты на нем. Вызов инициирует обновление каждого элемента на графике.

14. Plotter.remove(element)

Принимает один обязательный аргумент. Объект, переданный в качестве аргумента, должен быть возвращен одной из следующих функций: Plotter.addPoint(), Plotter.addFunc(), Plotter.addArea(), Plotter.addShadedArea(), Plotter.addParametricFunc(), Plotter.addLine(). Удаляет этот объект с графика и очищает его DOM-представление.

15. Plotter.removeAll()

Удаляет все объекты, нарисованные на графике, и очищает их DOM-представления.

16. Plotter.getID()

Возвращает id DOM-элемента, к которому прикреплен график.

17. Plotter.getModel()

Возвращает объект, который является моделью объекта типа Plotter. Он хранит в себе текущее состояние объекта типа Plotter. Полученная модель может быть передана в метод Plotter.setModel любого объекта типа Plotter, чтобы присвоить ему состояние, сохраненное в модели. Также можно передать модель в конструктор Plotter как единственный аргумент, чтобы конструктор построил объект по переданной модели.

Модель типа Plotter является составной, в отличие от моделей всех остальных объектов. Внутри она содержит модель объекта типа Plot, а также список из моделей всех нарисованных на графике объектов. Модель типа Plotter представляет полное состояние программы. Тем не менее, есть ограничения. Из-за того, что передача моделей между клиентом и сервером и между клиентами происходит с промежуточным преобразованием в *формат JSON*, то накладывается ограничение

на значения свойств объектов моделей. В частности, в JSON не может быть свойств-функций. Из-за этого методы объектов не добавляются в модель, которая возвращается методом Plotter.getModel().

18. Plotter.setModel(model, options)

Устанавливает у уже созданного объекта типа Plotter состояние. Первый аргумент обязателен и должен соответствовать возвращаемому объекту из Plotter.getModel(). Второй аргумент необязателен. Если он указан, то должен быть объектом, у которого может быть свойство silent, которое должно быть равно либо true, либо false. В случае если свойство silent указано true, то после установки состояния объекта в соответствие с моделью не будет автоматически вызван метод Plotter.redraw().

Установка модели у объекта типа Plotter через этот метод вызовет рекурсивную установку моделей у всех нарисованных объектов. Внутри модели и внутри объекта типа Plotter модели нарисованных объектов должны быть перечислены в массиве в одинаковом порядке.

19. Plotter.version

Свойство в прототипе Plotter. Хранит строку, которая показывает текущую версию библиотеки. Версия отвечает правилам нумерации версий программ *SemVer*.

iii. Методы Func

В основном, методы получают и устанавливают различные свойства объекта.

1. Точность прорисовки

Func.getAccuracy(), возвращает текущую точность прорисовки

Func.setAccuracy(Number), установит указанную точность прорисовки. Число должно быть целым.

Func.Accuracy(Number), если передать необязательный аргумент, то устанавливает точность прорисовки. Если не указывать, то возвращает текущую точность прорисовки.

2. Толщина

Func.getStrokeWidth(), возвращает текущую толщину линии графика.

Func.setStrokeWidth(Number), установливает указанную толщину.

Func.StrokeWidth(Number), если передать необязательный аргумент, то устанавливает толщину. Если не указывать, то возвращает текущую толщину.

3. Цвет

Func.getColor(), вернет текущий цвет в формате rgb.

Func.setColor(Number or String), установит указанный цвет. Если передавать строку, то она должна быть в формате rgb, то есть первый символ - решетка, а дальше 6 шестандцатиричных цифр. Если передавать число, то оно может быть от 0 до 20 включительно. 20 соответствует черному цвету. Остальные номера соответствуют этим цветам: https://github.com/mbostock/d3/wiki/Ordinal-Scales#category20

Func.Color(Number or String), аргумент необязателен. Если он есть, то будет установлен указанный цвет, как в .setColor(Number or String). Если его нет, то будет возвращен текущий цвет в формате rgb.

Func.getColour(),другое имя для .getColor()

Func.setColour(Number or String), другое имя для .setColor(Number or String)

Func.Colour(Number or String), другое имя для .Color(Number or String)

4. Разрывы функции

Func.getBreaks(), возвращает массив точек разрыва. Модификации в этом массиве приведут к изменениям в массиве внутри объекта. Иначе говоря, копия не создается, а предоставляется ссылка на массив.

Func.setBreaks(Array), устанавливает указанный массив точек разрыва. Копия при этом создаваться не будет.

Func.Breaks(Array), если необязательный аргумент указан, то устанавливает новый массив точек разрыва. Если нет, то возвращает текущий массив.

5. Границы прорисовки

Func.getLeft(), возвращает левую границу прорисовки.

Func.setLeft(Number), устанавливает левую границу прорисовки.

Func.Left(Number), если необязательный аргумент указан, то устанавливает левую границу прорисовки. Иначе возвращает текущую.

Func.getRight(), возвращает правую границу прорисовки.

Func.setRight(Number), устанавливает правую границу прорисовки.

Func.Right(Number), если необязательный аргумент указан, то устанавливает правую границу прорисовки. Иначе возвращает текущую.

6. Функция

Func.getFunc(), возвращает текущую функцию, которая используется в качестве функции обратного вызова для построения графика.

Func.setFunc(Function), установит функцию обратного вызова для построения графика.

Func.Func(Function), если аргумент указан, то установит функцию обратного вызова. Если нет, то вернет текущую функцию обратного вызова.

7. Func.clear()

вызывается объектом Plotter, когда происходит удаление объекта типа Func. При использовании вне Plotter может привести к ошибкам исполнения программы в дальнейшем.

8. Func.update()

Вызывается всякий раз, когда функции необходимо обновить координаты точек, используемых в пути. Как правило, это происходит при масштабировании и смещении графика. Также может использоваться при связывании объекта типа Func с другими объектами на графике и с состоянием программы. Таким образом, программист имеет возможность принудительно обновить функцию на графике.

9. Func.getPoints(num, func)

Возвращает массив точек функции. Функция делиться на несколько путей, количество которых зависит от количества точек разрыва. Если точек разрыва нет, то используется один путь. Если точка разрыва одна, то используется два пути, и так далее. Первый аргумент – это неотрицательное целое число, которое обозначает номер пути, массив точек которого нужно вернуть (нумерация путей с нуля). Второй аргумент необязателен. Если он указан, то это должна быть функция обратного вызова, которая принимает одно число (значение x) и возвращает еще одно число

(значение y). Если второй аргумент присутствует, то функция будет возвращать путь с указанным номером так, будто в качестве функции обратного вызова у объекта типа Func указана функция из второго аргумента данного метода. Такое переопределение поведения используется внутри библиотеки для выполнения задачи плавного перемещения (трансформации) графика функции из одного в другой.

10. Func.yMax(y)

Аргумент – число. Если оно не входит в допустимый отрезок значений, то оно считается равным верхней или нижней границе отрезка, в зависимости от знака. В этом случае возвращается одно из граничных значений. Если число лежит внутри интервала, то возвращается то же самое число, что и было указано в аргументе. Этот метод используется объектами типа Func, чтобы ограничить сверху и снизу график функции, так как если он будет уходить на бесконечность, то SVG элементы могут работать некорректно, не отображая нужные данные.

11. Func.moveTo(func, options)

Устанавливает в качестве текущей функции обратного вызова функцию func, которая принимает в качестве первого аргумента одно число (значение x), а возвращает еще одно число (значение y). При этом график текущей функции плавно переходит в график указанной в аргументе функции. Например, если изначально функция была (x) => x (нотация анонимных функций в стандарте Ecma-262 Edition 6), а стала (x) => x * x, то функция плавно поменяется от линейной к параболической. Второй аргумент необязателен. Это объект, который может иметь свойства delay и duration. Свойство delay отвечает за то, на сколько миллисекунд будет отложена трансформация функции с момента вызова метода moveTo. Свойство delay по умолчанию имеет нулевое значение. Свойство duration устанавливает время в миллисекундах, в течение которого будет происходить трансформация. По умолчанию свойство duration имеет значение 500.

12. Func.getModel()

Возвращает объект, который является моделью объекта типа Func. Он хранит в себе текущее состояние объекта типа Func. Полученная модель может быть передана в метод Func.setModel любого другого объекта типа Func, чтобы присвоить ему состояние, сохраненное в модели. Также можно передать модель в метод Plotter.addFunc в качестве единственного аргумента. Тогда строитель Func сделает новый объект типа Func, используя указанную модель.

13. Func.setModel(model, options)

Устанавливает у уже созданного объекта типа Func состояние. Первый аргумент обязателен и должен соответствовать возвращаемому объекту из Func.getModel(). Второй аргумент необязателен. Если он указан, то должен быть объектом, у которого может быть свойство silent, которое должно быть равно либо true, либо false. В случае, если свойство silent указано true, то после установки состояния объекта в соответствие с моделью не будет автоматически вызван метод Func.update().

iv. Методы ShadedArea

ShadedArea наследуется от Func, поэтому все методы у нее работаю так же, как у Func. Исключение составляет метод ShadedArea.getPoints(), который возвращает массив на 2 ячейки длиннее, чем это сделал бы метод Func.getPoints(). ShadedArea.getPoints() добавляет одну точку в начале массива и одну точку в конце массива. Первая точка является проекцией крайней левой точки функции на ось, которая указана в свойстве ShadedArea.axe. Последняя точка является проекцией крайней правой точки функции на ось, указанной в свойстве ShadedArea.axe.

v. Методы Line

1. Координаты

Line.setX1(x1)

Line.getX1()

Line.X1(x1), принимает необязательный аргумент. Если он задан, то устанавливает x1. Иначе возвращает текущее значение x1.

Line.setX2(x2)

Line.getX2()

Line.X2(x2), принимает необязательный аргумент. Если он задан, то устанавливает x2. Иначе возвращает текущее значение x2.

Line.setY1(y1)

Line.getY1()

Line.Y1(y1), принимает необязательный аргумент. Если он задан, то устанавливает y1. Иначе возвращает текущее значение y1.

Line.setY2(y2)

Line.getY2()

Line.Y2(y2), принимает необязательный аргумент. Если он задан, то устанавливает y2. Иначе возвращает текущее значение y2.

2. Цвет

Line.getColor(), вернет текущий цвет в формате rgb.

Line.setColor(Number or String), установит указанный цвет. Если передавать строку, то она должна быть в формате rgb, то есть первый символ - решетка, а дальше 6 шестандцатиричных цифр. Если передавать число, то оно может быть от 0 до 20 включительно. 20 соответствует черному цвету. Остальные номера соответствуют этим цветам: https://github.com/mbostock/d3/wiki/Ordinal-Scales#category20

Line.Color(Number or String), аргумент необязателен. Если он есть, то будет установлен указанный цвет, как в .setColor(Number or String). Если его нет, то будет возвращен текущий цвет в формате rgb.

Line.getColour(),другое имя для .getColor()

Line.setColour(Number or String), другое имя для .setColor(Number or String)

Line.Colour(Number or String), другое имя для .Color(Number or String)

3. Line.getModel()

Возвращает объект, который является моделью объекта типа Line. Он хранит в себе текущее состояние объекта типа Line. Полученная модель может быть передана в метод Line.setModel любого другого объекта типа Line, чтобы присвоить ему состояние, сохраненное в модели. Также можно передать модель в метод Plotter.addLine в качестве единственного аргумента. Тогда строитель Line сделает новый объект типа Line, используя указанную модель.

4. Line.setModel(model, options)

Устанавливает у уже созданного объекта типа Line состояние. Первый аргумент обязателен и должен соответствовать возвращаемому объекту из Line.getModel(). Второй аргумент необязателен. Если он указан, то должен быть объектом, у которого может быть свойство silent, которое должно быть равно либо true, либо false. В случае если свойство silent указано true, то после установки состояния объекта в соответствие с моделью не будет автоматически вызван метод Line.update().

5. Line.clear()

Вызывается объектом Plotter, когда происходит удаление объекта типа Line. При использовании вне Plotter может привести к ошибкам исполнения программы в дальнейшем.

6. Line.update()

Вызывается всякий раз, когда функции необходимо обновить координаты точек, используемых в пути. Как правило, это происходит при масштабировании и смещении графика. Также может использоваться при связывании объекта типа Line с другими объектами на графике и с состоянием программы. Таким образом, программист имеет возможность принудительно обновить функцию на графике.

vi. Методы Area

1. Цвет

Area.getColor(), вернет текущий цвет в формате rgb.

Area.setColor(Number or String), установит указанный цвет. Если передавать строку, то она должна быть в формате rgb, то есть первый символ - решетка, а дальше 6 шестандцатиричных цифр. Если передавать число, то оно может быть от 0 до 20 включительно. 20 соответствует черному цвету. Остальные номера соответствуют этим цветам: https://github.com/mbostock/d3/wiki/Ordinal-Scales#category20

Area.Color(Number or String), аргумент необязателен. Если он есть, тобудет установлен указанный цвет, как в .setColor(Number or String). Если его нет, то будет возвращен текущий цвет в формате rgb.

Area.getColour(), другое имя для .getColor()

Area.setColour(Number or String), другое имя для .setColor(Number or String)

Area.Colour(Number or String), другое имя для .Color(Number or String)

2. Заливка

Area.setFill(Number or String), устанавливает цвет заливки области. Работает так же, как и.setColor.

Area.getFill(), возвращает цвет заливки области в формате rgb.

Area.Fill(String or Number), принимает необязательный аргумент. Если он есть, то устанавливает указанный цвет, иначе возвращает текущий цвет.

3. Прозрачность

Area.setFillOpacity(Number), принимает число от 0 до 1 и устанавливает указанную прозрачность заливки, где 1 - непрозрачная, а 0 - полностью прозрачная, невидимая.

Area.getFillOpacity(), возвращает текущую прозрачность заливки.

Area.FillOpacity(Number), принимает необязательный аргумент. Если он есть, то устанваливает указанную прозрачность, иначе возвращает текущую.

4. Толщина

Area.getStrokeWidth(), вернет текущую толщину границы области.

Area.setStrokeWidth(Number), установит указанную толщину.

Area.StrokeWidth(Number), если передать *необязательный* аргумент, то установит толщину. Если не указывать, то вернет текущую толщину.

5. Area.clear()

Вызывается объектом Plotter, когда происходит удаление объекта типа Area. При использовании вне Plotter может привести к ошибкам исполнения программы в дальнейшем.

6. Area.update()

Вызывается всякий раз, когда необходимо обновить координаты точек, используемых в границе области. Как правило, это происходит при масштабировании и смещении графика. Также может использоваться при

связывании объекта типа Area с другими объектами на графике и с состоянием программы. Таким образом, программист имеет возможность принудительно обновить объект на графике.

7. Area.getModel()

Возвращает объект, который является моделью объекта типа Area. Он хранит в себе текущее состояние объекта типа Area. Полученная модель может быть передана в метод Area.setModel любого другого объекта типа Area, чтобы присвоить ему состояние, сохраненное в модели. Также можно передать модель в метод Plotter.addArea в качестве единственного аргумента. Тогда строитель Area сделает новый объект типа Area, используя указанную модель.

8. Area.setModel(model, options)

Устанавливает у уже созданного объекта типа Area состояние. Первый аргумент обязателен и должен соответствовать возвращаемому объекту из Area.getModel(). Второй аргумент необязателен. Если он указан, то должен быть объектом, у которого может быть свойство silent, которое должно быть равно либо true, либо false. В случае если свойство silent указано true, то после установки состояния объекта в соответствие с моделью не будет автоматически вызван метод Area.update().

vii. Методы ParametricFunc

Имеет те же методы, что Area. Работают точно так же. Отличие этого типа от Area состоит в других значениях по умолчанию, которые делают область не

закрашенной и выделяют границу области, то есть саму кривую, заданную по точкам.

viii. Методы Point

1. Размер

Point.setSize(size), меняет размер точки. Можно передать либо неотрицательное число, либо одну из четырех строк: "large", "medium", "small", "tiny".

Point.getSize(), возвращает текущий размер в пикселях.

Point.Size(size), принимает необязательный аргумент. Если он есть, то устанавливает размер, иначе возвращает текущий размер.

2. Координаты

Point.getX()

Point.setX(x)

Point.X(x), аргумент необязательный. Если он есть, то метод работает аналогично Point.setX(x). Если аргумент не указан, то работает аналогично Point.getX()

Point.getY()

Point.setY(y)

Point.Y(y), аргумент необязательный. Если он есть, то метод работает аналогично Point.setY(y). Если аргумент не указан, то работает аналогично Point.getY()

3. Состояние точки: движимая или статичная

Point.getMovable(), возвращает состояние точки. Двоичное значение, которое показывает, может ли передвигать пользователь ее мышкой или нет.

Point.setMovable(movable)

Point.Movable(y), аргумент необязательный. Если он есть, то вызывается Point.setMovable, иначе вызывается Point.getMovable.

4. Цвет

Point.getColor(), вернет текущий цвет в формате rgb.

Point.setColor(Number or String), установит указанный цвет. Если передавать строку, то она должна быть в формате rgb, то есть первый символ - решетка, а дальше 6 шестандцатиричных цифр. Если передавать число, то оно может быть от 0 до 20 включительно. 20 соответствует черному цвету. Остальные номера соответсвуют этим цветам: https://github.com/mbostock/d3/wiki/Ordinal-Scales#category20

Point.Color(Number or String), аргумент необязателен. Если он есть, тобудет установлен указанный цвет, как в .setColor(Number or String). Если его нет, то будет возвращен текущий цвет в формате rgb.

Point.getColour(), другое имя для .getColor()

Point.setColour(Number or String), другое имя для .setColor(Number or String)

Point.Colour(Number or String), другое имя для .Color(Number or String)

5. Point.update()

Вызывается всякий раз, когда необходимо обновить координаты точек, используемых в границе области. Как правило, это происходит при масштабировании и смещении графика. Также может использоваться при связывании объекта типа Area с другими объектами на графике и с состоянием программы. Таким образом, программист имеет возможность принудительно обновить объект на графике.

6. Point.clear()

Вызывается объектом Plotter, когда происходит удаление объекта типа Point. При использовании вне Plotter может привести к ошибкам исполнения программы в дальнейшем.

7. Point.getModel()

Возвращает объект, который является моделью объекта типа Point. Он хранит в себе текущее состояние объекта типа Line. Полученная модель может быть передана в метод Point.setModel любого другого объекта типа Point, чтобы присвоить ему состояние, сохраненное в модели. Также можно передать модель в метод Plotter.addPoint в качестве единственного аргумента. Тогда строитель Point сделает новый объект типа Point, используя указанную модель.

8. Point.setModel(model, options)

Устанавливает у уже созданного объекта типа Point состояние. Первый аргумент обязателен и должен соответствовать возвращаемому объекту из Point.getModel(). Второй аргумент необязателен. Если он указан, то должен быть объектом, у которого может быть свойство silent, которое должно быть равно либо true, либо false. В случае если свойство silent указано true, то после установки состояния объекта в соответствие с моделью не будет автоматически вызван метод Point.update().