И.А. Беккер, ст.преподаватель; А.М. Андреев, студент; С.Н. Петроченко, студент
(Белорусско-Российский университет, г. Могилев)

СОЗДАНИЕ ЭЛЕКТРОННОГО СРЕДСТВА ОБУЧЕНИЯ: РЕАЛИЗАЦИЯ МЕТОДА ГРАДИЕНТНОГО СПУСКА

При создании электронного web-средства обучения «Методы безусловной оптимизации» разрабатывались отдельные веб-модули для каждого конкретного метода оптимизации, которые затем интегрировались в единое приложение.

Web-приложение было спроектировано только как клиентская часть, поскольку в нем нет необходимости работать с базами данных и хранить их на сервере, серверная часть как таковая не является в этом проекте необходимой. Все необходимые задачи проекта: вычислительные, обучающие – были выполнены на FrontEnd.

Языком программирования был выбран JavaScript как скриптовый интерпретируемый язык, у которого обработка скрипта выполняется полностью на стороне клиента без необходимости обращаться к серверной части.

В реализации метода градиентного спуска для функции двух переменных использовались следующие библиотеки, которые помогли решить некоторые математические задачи: Nerdamer.js, Algebra.js, Solve.js, Calculus.js, Math.js.

Nerdamer - это небольшая по весу библиотека-анализатор математических выражений, написанный на JavaScript. Nerdamer является модульным, поэтому можно загружать только те части, которые понадобятся. Из-за этого и своей компактности он может быть легко встроен в веб-приложения.

Библиотека (начиная с версии 0.5.0) делится на ядро nerdamer.core.js и несколько дополнительных модулей (сами по себе которые являются библиотеками функций), которые загружаются после загрузки ядра, что позволяет выбрать нужную функциональность.

Обрабатывая математическое выражение в Nerdamer, его следует представить как объект типа nerdamer, который принимает строку в качестве основного параметра (как дополнительный параметр можно использовать объект-переменную, например, с заданным значением или рядом значений).

Библиотека Nerdamer имеет дополнительные модули Algebra.js, Solve.js, Calculus.js, сами по себе которые являются библиотеками функций.

Algebra.js работает с такими объектами, как дроби, выражения и уравнения и содержит множество функций для обработки этих объектов, например, функция SolveEquations решает систему линейных уравнений; SolveFor – решает уравнение, если решение не найдено, возвращается пустой массив.

Calculus.js включает такие функции, как Summation – суммирует выражение с учетом изменения указанной в виде параметра переменной от нижнего до верхнего предела; Product – вычисляет произведение выражения с переменной, изменяемой от нижнего до верхнего предела; Differentiate – вычисляет производную; Integrate – вычисляет интеграл (для этого используются приближенные методы вычислительной математики).

Math.js является мощной математической библиотекой для JavaScript и Node.js, имеющей большой набор встроенных функций и констант, совместимой со встроенной в JavaScript математической библиотекой.

Встроенная библиотека JavaScript является объектом и реализована как класс, она имеет набор свойств, возвращающих значения основных констант, и методов, возвращающих значения тригонометрических, логарифмической, экспоненциальной, показательной и других функций.

Все эти математические инструменты JavaScript позволяют решать самые разные вычислительные задачи.

При выборе метода градиентного спуска на начальной странице сайта все данные, необходимые для вычисления экстремума функции, вводятся пользователем в поля на html-странице: сама функция fn, точности ε1 и ε2, предельное число итераций h, начальная точка (как массив x с элементами x[0], x[1]).

Введенная пользователем функция считывается с формы html следующим образом:

```
<input type="text" name="f" />
```

Атрибут name в теге input определяет уникальное имя элемента формы, затем оно используется для доступа к введенным данным поля через скрипты:

```
fn = obj.f.value;
```

При считывании функции используется метод eval, чтобы выполнить программный код, записанный строкой.

Синтаксис функции eval библиотеки math следующий: math.eval(expr, scope), где первый параметр expr – это выражение для вычисления, второй параметр параметр scope – область для чтения или записи переменных. Второй параметр является необязательным.

Методы градиентного спуска относятся к методам поиска экстремума первого порядка, в которых требуется вычисление первых частных производных исходной функции f, что реализовано через функцию derivative библиотеки Math.js. Синтаксис данной функции: derivative (expr, variable), где expr – выражение для дифференцирования, variable – переменная, по которой производится дифференцирование.

Для оценки и анализа строкового выражения может быть использована также функция parse из библиотеки Math.js.

Пример использования функций derivative и parse библиотеки Math.js:

```
const f = math.parse('x^2+3*x');
const x = math.parse('x');
math.derivative(f, x);        // Результат: 2 * x + 3
```

В программном коде функция derivative использовалась согласно алгоритма метода следующим образом:

```
var pr = math.derivative(f, 'x'); //pr – переменная, в которую записывается производная функции f по переменной x.
```

Аналогично вычисляется производная функции f по переменной y в виде функции, формируется градиент функции f как массив, состоящий из этих двух значений. Нужно на каждом шаге вычислять значение $\nabla f(X^k)$ в текущей k-ой точке как точку с двумя координатами и евклидову норму задаваемого ею ра-

диус-вектора. Это первая проверка критерия окончания алгоритма, и если он выполнен, то расчет окончен и стационарная точка найдена. Иначе выполняется проверка второго критерия на максимально допустимое количество шагов, которое указал пользователь.

Условие для проверки количества пройденных шагов реализовано в цикле:

```
var h1 = 0; //h1 – счетчик шагов
    while (h1 < h) { //пока текущее значение количества шагов меньше заданного - выполнять тело цикла
… //тело цикла
 h1++; //увеличиваем количество шагов
}
```

Если количество шагов меньше указанного, то продолжается расчет и нужно выполнить шаг вдоль направления антиградиента и найти новое значение аргумента $X^1 = X^0 - \lambda * \nabla f(X^0)$:

```
var antigradX = x[0].toString() + "-" + "(" + GradArray[0].toString() + ")";
newPointX = antigradX + "*l";
```

Высчитывать новую точку удобно при помощи метода поиска-замены replace:

```
var pointX = newPointX.toString().replace(/l/g, l.toString());
var pointY = newPointY.toString().replace(/l/g, l.toString());
x[0] = eval(pointX);
x[1] = eval(pointY);
```

Метод градиентного спуска с постоянным шагом требует задания значения λ пользователем изначально (оно остается постоянным при выполнении условия приближения функции к минимуму). Метод наискорейшего градиентного спуска вычисляет на каждой k-ой итерации шаг (множитель λ_k) из условия $f(X^k - \lambda_k * \nabla f(X^k)) \rightarrow \min$. Поскольку в минимизируемой функции единственным аргументом будет λ_k, вопрос его нахождения разрешим любым методом нулевого порядка (например, Фибоначчи или «золотого сечения»). Для нахождения минимума можно применить и другой способ: использовать необходимое условие экстремума с дальнейшей проверкой подозрительных на экстремум точек.

Для применения необходимого условия экстремума требуется приравнять полученную производную pr функции f к нулю, сначала сформировав уравнение в виде символьного выражения с идентификатором f1:

```
var f1 = pr.toString() + "=0";
```

Решить уравнение f1 можно с помощью функции solve библиотеки Solve.js:

```
var root = nerdamer.solve(f1, 'x');
```

Здесь f1 - уравнение, которое требуется решить, x – переменная, относительно которой нужно разрешить уравнение. Таким образом, при реализации метода градиентного спуска студенты изучили математические средства JavaScript, организовали пошаговый вывод промежуточных результатов и ответа.