

Алгоритмы порождения допустимых суперпозиций существенно нелинейных моделей

Г. И. РУДОЙ

Аннотация. При восстановлении нелинейной регрессии предлагается рассмотреть набор индуктивно порожденных моделей с целью выбора оптимальной модели. В работе исследуются индуктивные алгоритмы порождения допустимых существенно нелинейных моделей. Предлагается алгоритм, порождающий все возможные суперпозиции заданной сложности за конечное число шагов, и приводится его теоретическое обоснование. В вычислительном эксперименте приводятся результаты для задачи моделирования волатильности опционов.

1. ВВЕДЕНИЕ

В ряде приложений [1–3] возникает задача восстановления регрессии по набору измеренных данных с условием возможности проинтерпретировать полученные данные экспертом.

Одним из методов, позволяющих получать интерпретируемые модели, является символьная регрессия [4, 5], в ходе которой измеренные данные приближаются некоторой математической формулой, например, $\sin x^2 + 2x$ или $\log x - \frac{e^x}{x}$. В ходе работы символьной регрессии данные приближаются различными формулами, являющимися произвольными суперпозициями функций из некоторого заданного набора. Одна из возможных реализаций этого метода предложена Джоном Коза [6, 7], использовавшим эволюционные алгоритмы для реализации символьной регрессии. Иван Зелинка предложил дальнейшее развитие этой идеи [8], получившее название аналитического программирования.

Алгоритм построения требуемой математической модели выглядит следующим образом: дан набор примитивных функций, из которых можно строить различные формулы (например, степенная функция, $+$, \sin , \tan). Начальный набор формул строится либо произвольным образом, либо на базе некоторых предположений эксперта. Затем на каждом шаге производится оценка каждой из формул согласно функции ошибки либо другого ¹ функционала качества. На базе этой оценки у некоторой части формул случайным образом заменяется одна элементарная функция на другую (например, \sin на \cos или $+$ на \times), а у некоторой другой части происходит взаимный попарный обмен подвыражениями в формулах.

Key words and phrases. Символьная регрессия, индуктивное порождение нелинейных моделей.

Научный руководитель В. В. Стрижов.

¹Сходу не нашел публикаций на тему использования других функционалов. Похоже, у Владиславовой что-то было, но пока не могу сослаться на что-то конкретное.

Получаемая формула является математической моделью [9] исследуемого процесса или явления — то есть, это математическое отношение, описывающее основные закономерности, присущие этому явлению.

Среди возможных путей улучшения качества символьной регрессии — анализ информативности различных признаков. Например, в ходе работы эволюционного алгоритма можно выявлять, какие из параметров слабо влияют на качество получающейся формулы, и либо убирать их совсем, либо обеспечивать неслучайность замены элементарных функций или обмена поддеревьев с целью замены этих параметров на другие в предположении, что они, возможно, окажутся более информативными.

Целью данной работы является теоретическое обоснование алгоритмов индуктивного порождения моделей и анализ этих алгоритмов.

Другим вопросом, возникающим при применении подобных эволюционных алгоритмов, является их принципиальная теоретическая корректность: способен ли вообще такой алгоритм породить искомую формулу.

Алгоритм индуктивного порождения моделей, предложенный в настоящей работе, решает некоторые типичные проблемы предложенных ранее методов, упомянутые, например, в [8], а именно:

- Порождение рекурсивных суперпозиций, суперпозиций, содержащих несоответствующее используемым функциям число аргументов, и т. д. — в предложенном алгоритме эти проблемы не возникают по построению.
- Проблемы с областью определений и/или значений.
- При ограничении числа примитивных функций, участвующих в суперпозиции, а также при соответствующем задании множества примитивных функций исключается проблема слишком сложных суперпозиций.

В части 2 данной работы формально поставлена задача построения алгоритма индуктивного порождения моделей. Затем, в части 3 строится искомый алгоритм для частного случая непараметризованных моделей и доказывается его корректность, а затем алгоритм обобщается на случай моделей, имеющих параметры. В части 4 описываются вспомогательные технические приемы, использованные в практическом алгоритме порождения моделей, описанном в части 5.

2. ПОСТАНОВКА ЗАДАЧИ

2.1. Алгоритмическая часть. Пусть дана регрессионная выборка:

$$D = \{(\mathbf{x}_i, y_i) \mid i \in \{1, \dots, N\}, \mathbf{x}_i \in \mathbb{X} \subset \mathbb{R}^n, y_i \in \mathbb{Y} \subset \mathbb{R}\}.$$

Требуется построить аналитическую функцию $f : \mathbb{R}^n \rightarrow \mathbb{R}$ из заданного множества элементарных функций G и доставляющую минимум некоторому функционалу ошибки.

То есть, если множество всех суперпозиций:

$$\mathcal{F} = \{f_r \mid f_r : (\boldsymbol{\omega}, \mathbf{x}) \mapsto \mathbb{Y}, r \in \mathbb{R}\},$$

то требуется найти:

$$r_{\min} = \arg \min_{r \in \mathbb{R}} S(f_r \mid \hat{\omega}_r, D),$$

где

$$\hat{\omega}_r = \arg \min_{\omega \in \Omega} S(\omega \mid f_r, D).$$

2.2. Теоретическая часть. Пусть $G = \{g_1, \dots, g_{n_g}\}$ — множество данных примитивных функций, а именно, для каждой $g_i \in G$ задано:

- Сама функция (например, \sin , \cos , \times).
- Арность функции и порядок следования аргументов.
- Домен и кодомен функции (например, $R \mapsto R$ или $R^2 \mapsto \{0, 1\}$).
- Область определения и область значений в рамках домена и кодомена (например, для $\log_{x_1} x_2 : x_1 \in (0; 1) \cup (1; +\infty), x_2 \in (0; +\infty)$).

Требуется:

- Построить алгоритм \mathfrak{A} , за конечное число итераций порождающий любую конечную суперпозицию данных примитивных функций.
- Указать способ проверки изоморфности двух суперпозиций.

Заметим, что мы не требуем для примитивных функций свойства их непорождаемости в наиболее общей формулировке типа принципиальной невозможности породить в ходе работы искомого алгоритма суперпозицию, изоморфную некоторой функции из G . Такое требование является слишком ограничивающим. В частности, невозможно было бы иметь в G одновременно, например, функции id , \exp и \log , так как $id \equiv \log \circ \exp$.

3. ПУТИ РЕШЕНИЯ ЗАДАЧИ: ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Введем некоторые понятия из теории графов:

Определение 1. *Дерево — связный ациклический граф.*

Иными словами, дерево Γ — это некоторый набор вершин V_i и соединяющих их ребер $E(U, V)$ обладающий следующими свойствами:

- Из каждой вершины V_i можно попасть в каждую другую вершину $V_j, j \neq i$ (условие связности).
- Для любой вершины V_i не существует пути из нее в нее же (условие ациклическости).

Определение 2. *Ориентированный граф (орграф) — граф, в котором каждому ребру E сопоставлено направление.*

Ребра в орграфах также называются *дугами*.

Определение 3. *Ориентированное дерево — связный ациклический орграф, в котором только одна вершина не имеет входящих в нее дуг, а все остальные вершины имеют только одну входящую в них дугу.*

Заметим, что для ориентированного дерева условие связности следует уточнить. А именно, связность означает, что для любой пары различных вершин U и V существует путь между ними без учета направления.

В дальнейшем мы будем работать в основном с ориентированными деревьями, поэтому для краткости будем называть их просто деревьями. В случае необходимости неориентированность будет указана явно.

Условимся называть вершину дерева, не имеющую входящих в нее дуг, *корнем дерева*:

Определение 4. *Корень дерева — вершина дерева, в которую не входит ни одна дуга.*

Условимся обозначать вершину дерева V_0 .

Определение 5. *Лист дерева — вершина, из которой не исходит ни одной дуги.*

Определение 6. *Уровень вершины — длина пути от корня дерева до данной вершины.*

Заметим, что каждой вершине V_i мы можем поставить в соответствие поддереву Γ^i , убрав дугу, приходящую в V_i , и убрав все вершины, недостижимые из V_i . Корнем такого дерева, очевидно, является V_i .

Введем понятия множества смежных вершин для данной вершины:

Определение 7. *Множество смежных вершин $S(U)$ для вершины U — вершины, в которые входит некоторая дуга, выходящая из U . Иными словами: $S(U) = \{V \mid \exists E(U, V)\}$.*

Сформулируем алгоритм поиска в глубину применительно к ориентированным деревьям. Будем считать, что вершины одного уровня расположены в некотором заранее заданном порядке.

Алгоритм 1. *Алгоритм поиска в глубину (Depth-First Search, DFS).*

```
map<Vertex, int> DFS (Vertex v, int current)
{
    map<Vertex, int> result;
    result [v] = current;

    auto children = S (v);

    for (Vertex child : children)
        result = Unite (result, DFS (child, ++current));

    return result;
}
```

DFS (v0, 0);

Отметим, что в общем случае алгоритм **DFS** строится более сложным образом для учета циклов и случаев несвязных графов.

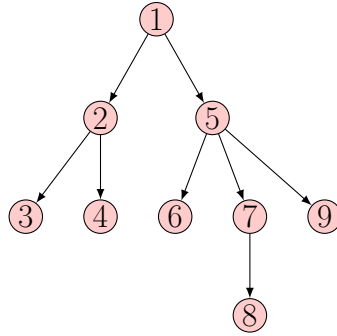


Рис. 1. Порядок обхода вершин при поиске в глубину

Условимся считать, что каждой суперпозиции f сопоставлено дерево Γ_f , эквивалентное этой суперпозиции и строящееся следующим образом:

- В вершинах V_i дерева Γ_f находятся соответствующие примитивные функции $g_s, s = s(i)$.
- Число дочерних вершин у некоторой вершины V_i равно аргументности соответствующей функции g_s .
- Порядок смежных некоторой вершине V_i вершин соответствует порядку аргументов соответствующей функции $g_{s(i)}$.
- В листьях дерева Γ_f находятся свободные переменные либо константы.
- Порядок вершин V_i в смысле уровня вершин определяет порядок вычисления примитивных функций: дерево вычисляется снизу вверх.

То есть, сначала подставляются конкретные значения свободных переменных, затем вычисляются значения в вершинах, все дочерние вершины которых — свободные переменные, и так далее до тех пор, пока не останется единственная вершина, бывшая корнем дерева, содержащая результат выражения.

Таким образом, вычисление значения выражения f в некоторой точке эквивалентно подстановке соответствующих значений свободных переменных в дерево Γ_f выражения.

Заметим важное свойство таких деревьев: каждое поддерево Γ_f^i дерева Γ_f , соответствующее вершине V_i , также соответствует некоторой суперпозиции, являющейся составляющей исходной суперпозиции f .

Для примера рассмотрим дерево, соответствующее суперпозиции $\sin(\ln x_1) + \frac{x_2^3}{2}$ (см. рис 2).

Здесь точками обозначены аргументы функций. Как видно, корнем дерева является вершина, соответствующая операции сложения, которая должна быть выполнена

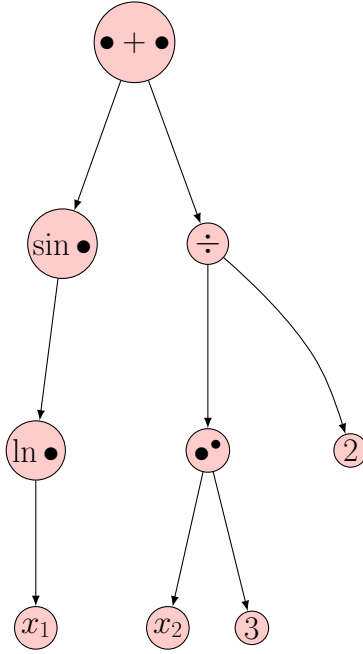


Рис. 2. Дерево выражения $\sin(\ln x_1) + \frac{x_2^3}{2}$

в последнюю очередь. Операция сложения имеет два различных поддеревя, соответствующих двум аргументам этой операции.

Заметим также, что здесь не использованы операции типа «разделить на два» или «возвести в куб». Вместо этого используются операции деления и возведения в степень в общем виде, а в данном конкретном дереве соответствующие аргументы зафиксированы соответствующими константами.

3.1. Алгоритм порождения суперпозиций. Итак, пусть дано множество примитивных функций $G = \{g_1, \dots, g_{n_g}\}$ и множество свободных переменных $X = \{x_1, \dots, x_{n_x}\}$. Сначала опишем итеративный алгоритм, позволяющий за конечное число итераций построить суперпозицию произвольной наперед заданной длины без учета числовых коэффициентов. Для удобства будем исходить из предположения, что множество G состоит только из унарных и бинарных функций, и разделим его соответствующим образом на два подмножества: $G = G_b \cup G_u \mid G_b = \{g_{b_1}, \dots, g_{b_k}\}, G_u = \{g_{u_1}, \dots, g_{u_l}\}$, где G_b — множество всех бинарных функций, а G_u — множество всех унарных функций из G . Потребуем также наличия id в G_b .

Алгоритм 2. Алгоритм \mathfrak{A} итеративного порождения суперпозиций.

- (1) Инициализируем вспомогательное множество $\mathcal{I}_f = \{(x, 0) \mid x \in X\}$.
- (2) Инициализируем множество $\mathcal{F}_0 = X$.

- (3) Для множества \mathcal{F}_i построим вспомогательное множество U_i , состоящее из результатов применения функций из G_u к элементам \mathcal{F}_i :

$$U_i = \{g_u \circ f \mid g_u \in G_u, f \in \mathcal{F}_i\}$$

- (4) Аналогичным образом построим вспомогательное множество B_i для бинарных функций:

$$B_i = \{g_b \circ (f, h) \mid g_b \in G_b, f, h \in \mathcal{F}_i\}$$

- (5) Обозначим $\mathcal{F}_{i+1} = \mathcal{F}_i \cup U_i \cup B_i$.
 (6) Для каждой суперпозиции f из \mathcal{F}_{i+1} добавим пару $(f, i+1)$ в множество \mathcal{I}_f , если суперпозиция f еще там не присутствует.
 (7) Перейдем к следующей итерации.

Тогда $\mathcal{F} = \bigcup_0^\infty \mathcal{F}_i$ — множество всех возможных суперпозиций конечной длины, которые можно построить из данного множества примитивных функций.

Вспомогательное множество \mathcal{I}_f позволяет запоминать, на какой итерации была впервые встречена данная суперпозиция. Это необходимо, так как каждая суперпозиция, впервые порожденная на i -ой итерации, будет порождена еще раз и на любой итерации после i .

Алгоритм \mathfrak{A} очевидным образом обобщается на множество G , содержащее функции произвольной (но имеющей конечную верхнюю грань) арности. Действительно, для такого обобщения достаточно строить аналогичным образом вспомогательные множества для этих функций, а именно, для множества функций G_n арности n построим вспомогательное множество H_i^n вида:

$$H_i^n = \{g \circ (f_1, f_2, \dots, f_n) \mid g \in G_n, f_i \in \mathcal{F}_i\}.$$

В этих обозначениях $U_i \equiv H_i^1$, а $B_i \equiv H_i^2$.

Тогда множество $\mathcal{F}_{i+1} = \mathcal{F}_i \cup_{n=0}^{n_{max}} H_i^n$, где n_{max} — максимальное значение арности функций из G .

Утверждение 1. *Алгоритм \mathfrak{A} действительно породит любую конечную суперпозицию за конечное число шагов.*

Доказательство. Чтобы убедиться в этом, найдем номер итерации, на котором будет порождена некоторая произвольная конечная суперпозиция f . Для этого достаточно представить суперпозицию f в виде соответствующего графа Γ_f и рекурсивно пройти от вершин к листьям, составляя цепочку соотношений на номера итераций по следующим правилам:

- Если вершина V , полученная на i -ой итерации — унарная функция, то это функция от выражения, полученного на $(i-1)$ -ой итерации.
- Если вершина V , полученная на i -ой шаге — бинарная функция, то это функция от двух выражений, как минимум одно из которых получено на $(i-1)$ -ой итерации, а другое — на $(i-1)$ -ой или ранее.
- Если это лист со свободной переменной, то он получен на нулевой итерации.

При помощи этой цепочки соотношений можно получить номер итерации, на которой суперпозиция f была порождена.

Иными словами, для любой суперпозиции мы можем указать конкретный номер итерации, на котором она будет получена, что и требовалось. \square

В предложенных ранее методах [8] построения суперпозиций необходимо было самостоятельно следить за тем, чтобы в ходе работы алгоритма не возникало «зацикленных» суперпозиций типа $f(x, y) = g(f(x, y), x, y)$. Заметим, что в предложенном алгоритме \mathfrak{A} такие суперпозиции не могут возникнуть по построению.

3.2. Порождение параметризованных моделей. Алгоритм в таком виде не позволяет получать выражения для численных коэффициентов. Покажем, однако, на примере конструирования множеств U_i и B_i , как исходный алгоритм может быть расширен с учетом таких коэффициентов путем введения параметров:

$$U_i = g_u \circ (\alpha f + \omega),$$

$$B_i = g_b \circ (\alpha f + \omega, \psi h + \phi).$$

Будем обозначать этот расширенный алгоритм как \mathfrak{A}^* .

Здесь параметры α, ω зависят только от комбинации g_u, f (или g_b, f, h для $\alpha, \omega, \psi, \phi$). Соответственно, для упрощения их индексы опущены.

Иными словами, мы предполагаем, что каждая суперпозиция из предыдущих итераций входит в следующую, будучи умноженной на некоторый коэффициент и с константной поправкой.

Очевидно, при таком добавлении параметров $\alpha, \omega, \psi, \phi$ мы не изменяем мощности получившегося множества суперпозиций, поэтому алгоритм и выводы из него остаются корректны. В частности, исходный алгоритм является частным случаем данного при $\alpha \equiv \psi \equiv 1, \omega \equiv \phi \equiv 0$.

$\alpha, \omega, \psi, \phi$ являются параметрами модели. В практических приложениях можно оптимизировать значения этих параметров у получившихся суперпозиций, например, алгоритмом Левенберга-Марквардта [10, 11].

Заметим также, что такая модификация алгоритма позволяет нам получить единицу, например, для построения суперпозиций типа $\frac{1}{x}: 1 = \alpha \text{ id } x + \omega \mid \alpha = 0, \omega = 1$.

Отдельно подчеркнем, что численные коэффициенты у различных суперпозиций различны. Однако, так как на разных итерациях алгоритма мы можем получить, вообще говоря, одну и ту же суперпозицию с точностью до этих коэффициентов, их необходимо не учитывать при тестировании различных суперпозиций на равенство.

Кроме того, опять же, заметим, что и этот алгоритм очевидным образом обобщается на случай множества G , содержащего функции произвольной ариности.

3.3. Множество допустимых суперпозиций. Предложенный выше алгоритм позволяет получить действительно все возможные суперпозиции, однако, не все они будут пригодны в практических приложениях: например, $\ln x$ имеет смысл только при $x > 0$, а $\frac{x}{0}$ не имеет смысла вообще никогда. Выражения типа $\frac{x}{\sin x}$ имеют смысл только при $x \neq \pi k$.

Таким образом, необходимо введение понятия множества *допустимых* суперпозиций — то есть, таких суперпозиций, которые в условиях некоторой задачи корректны.

Определение 8. *Допустимая суперпозиция f — такая суперпозиция, значение которой определено для любой комбинации входных данных, область значений \mathcal{X} которых определяется конкретной задачей, $\mathcal{X} \in R^n$, где n — максимально возможная размерность входных данных.*

Одним из способов построения только допустимых суперпозиций является модификация предложенного алгоритма таким образом, чтобы отслеживать совместность областей определения и областей значения соответствующих функций в ходе построения суперпозиций. Для свободных переменных это будет, в свою очередь, означать необходимость задания областей значений \mathcal{X} пользователем при решении конкретных задач.

Заметим, что, хотя теоретически возможно выводить допустимость выражений вида $\frac{x}{\sin x}$ исходя из заданных условий на свободную переменную (например, что $x \in (\frac{\pi}{4}, \frac{\pi}{2})$), в общем случае это потребует решения неравенств в общем виде, что вычислительно неэффективно.

Таким образом, можно сформулировать признак недопустимости суперпозиции:

Определение 9. *Признак недопустимости суперпозиции f : в соответствующем дереве Γ_f хотя бы одна вершина V_i имеет хотя бы одну дочернюю вершину V_j такую, что область значений функции $g_{s(j)}$ шире, чем область определения функции $g_{s(i)}$.*

Здесь мы считаем для общности формулировки, что вершина со свободной переменной x_i эквивалентна функции $id(x_i)$.

Для построения множества допустимых суперпозиций достаточно построить множество всех возможных суперпозиций при помощи алгоритма \mathfrak{A}^* , а затем удалить из этого множества все суперпозиции, не удовлетворяющие сформулированному признаку.

3.4. Множество «минимальных» суперпозиций. В ходе работы алгоритма могут возникать суперпозиции вида $x + x$ и $2x$, и хотя эти выражения эквивалентны, они представляются различными формулами. Аналогично эквивалентны $x + y$ и $y + x$, отличающиеся порядком следования слагаемых. Таким образом, необходим способ нормализации суперпозиций.

Во-первых, необходимо обеспечивать одинаковый порядок следования операндов, например, упорядочивая их каким-либо образом у коммутирующих бинарных функций.

Во-вторых, необходимо иметь набор правил, позволяющих проверить равенство $x + x$ и $2x$. Иными словами, необходимо иметь набор связей между различными функциями из множества данных примитивных функций. Заметим, что в общем случае эта задача требует введения значительного числа правил и по определению сводится к последовательному переборному их применению к различным подвыражениям суперпозиции.

В связи с этим может оказаться более эффективным иной подход к сравнению суперпозиций: так как по условию практической задачи значения искомой функции даны в конечном числе точек, то для проверки на равенство достаточно вычислить получившиеся суперпозиции в этих точках и сравнить их.

Другим способом, позволяющим избежать разрастания количества правил, может являться использование только «независимых» функций. Например, \sin и \cos связаны известным тригонометрическим соотношением с точностью до знака, а значит, \sin и $\tan = \frac{\sin}{\cos}$ также связаны, как и ряд прочих тригонометрических функций, поэтому предлагается среди примитивных функций оставить лишь \sin и стандартные арифметические действия для вывода прочих тригонометрических функций через соответствующие соотношения.

Однако, можно заметить два часто встречающихся шаблона правил, связывающих различные функции:

- Для унарных функций это $f \circ g = h$ (например, $\ln \circ \exp = id$).
- Для бинарных функций это $f(x, g(x, i)) = g(x, s(i))$. Например, $x + xi = x(i + 1)$: здесь $f = (+)$, $g = (\times)$, $s(i) = i + 1$.

В практических приложениях представляется целесообразным использование набора правил такого вида вкупе с использованием только «независимых» тригонометрических функций, то есть, по факту, какой-нибудь одной из них и еще одной обратной.

4. АЛГОРИТМ ЛЕВЕНБЕРГА-МАРКВАРДА И МУЛЬТИСТАРТ

Алгоритм Левенберга-Марквардта (\mathcal{LM}) [10, 11] предназначен для решения задачи минимизации функции, представляющей из себя сумму квадратичных членов. В частности, он используется для оптимизации параметров нелинейных регрессионных моделей в предположении, что в качестве критерия оптимизации используется среднеквадратичная ошибка модели на обучающей выборке:

$$S(\omega) = \sum_{i=1}^m [y_i - f(\omega, \mathbf{x}_i)]^2 \rightarrow \min,$$

где ω — вектор параметров модели (суперпозиции) f .

\mathcal{LM} может рассматриваться как комбинация методов Гаусса-Ньютона и градиентного спуска.

Перед началом работы алгоритма задается начальный вектор параметров ω_0 . На каждой итерации этот вектор заменяется новой оценкой, $\omega_{k+1} = \omega_k + \delta_k$. Для определения $\delta_k = \delta$ используется линейное приближение функции:

$$\mathbf{f}(\omega + \delta, \mathbf{X}) \approx \mathbf{f}(\omega, \mathbf{X}) + \mathbf{J}\delta,$$

где \mathbf{J} — якобиан функции \mathbf{f} в точке ω .

Приращение δ в точке ω , доставляющей минимум S , равно нулю, поэтому для нахождения последующего значения приращения δ приравняем нулю вектор частных производных S по ω . То есть, в векторной нотации:

$$S(\boldsymbol{\omega} + \boldsymbol{\delta}) \approx \|\mathbf{y} - \mathbf{f}(\boldsymbol{\omega}) - \mathbf{J}\boldsymbol{\delta}\|^2.$$

Дифференцирование по $\boldsymbol{\delta}$ и приравнивание нулю приводит к следующему уравнению для $\boldsymbol{\delta}$:

$$(\mathbf{J}^T \mathbf{J})\boldsymbol{\delta} = \mathbf{J}^T[\mathbf{y} - \mathbf{f}(\boldsymbol{\omega})].$$

Левенберг предложил заменить $(\mathbf{J}^T \mathbf{J})$ на $(\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I})$, где λ — некоторый параметр регуляризации. Марквардт дополнил это предложение с целью более быстрого движения по тем направлениям, где градиент меньше. Для этого вместо \mathbf{I} используется диагональ матрицы $\mathbf{J}^T \mathbf{J}$, и искомое уравнение на $\boldsymbol{\delta}$ выглядит как:

$$(\mathbf{J}^T \mathbf{J} + \lambda \text{diag}(\mathbf{J}^T \mathbf{J}))\boldsymbol{\delta} = \mathbf{J}^T[\mathbf{y} - \mathbf{f}(\boldsymbol{\omega})].$$

Решая это уравнение, получаем окончательное выражение для $\boldsymbol{\delta} = \boldsymbol{\delta}_k$:

$$\boldsymbol{\delta}_k = (\mathbf{J}^T \mathbf{J} + \lambda \text{diag}(\mathbf{J}^T \mathbf{J}))^{-1} \mathbf{J}^T[\mathbf{y} - \mathbf{f}(\boldsymbol{\omega})].$$

4.1. Выбор λ . Для определения параметра регуляризации λ в настоящей работе применяется следующая эвристика.

В начале работы \mathcal{LM} задается некоторое значение λ_0 , например, 0.01, и фиксируется коэффициент $\nu > 1$. Затем, на каждой итерации алгоритма вычисляется значение функционала ошибки для $\lambda = \lambda_i$ и $\lambda = \frac{\lambda_i}{\nu}$. В случае, если хотя бы одно из этих значений доставляет функционалу ошибки меньшее значение, чем до этой итерации, то λ_{i+1} принимается равным этому значению. Иначе λ_i умножается на ν до тех пор, пока значение функционала ошибки не уменьшится.

4.2. Мультистарт. Как и всякий подобный алгоритм оптимизации, \mathcal{LM} находит лишь локальный минимум. Для решения этой проблемы применяется метод *мультистарта*: случайным образом задается несколько начальных приближений, и для каждого из них запускается \mathcal{LM} . Если найдено несколько различных локальных минимумов, то выбирается тот из них, в котором значение $S(\boldsymbol{\omega})$ меньше всего.

5. ВЫЧИСЛИТЕЛЬНЫЙ ЭКСПЕРИМЕНТ

5.1. Алгоритм. Несмотря на то, что указанный ранее итеративный алгоритм порождения суперпозиций позволяет получить, в принципе, произвольную суперпозицию, для практических применений он непригоден, как и любой алгоритм, реализующий полный перебор, в связи с чрезмерной вычислительной сложностью. Вместо него можно использовать стохастические алгоритмы и ряд эвристик, позволяющих на практике получать за приемлемое время результаты, удовлетворяющие заранее заданным условиям «достаточной пригодности».

В настоящей работе предлагается следующий алгоритм:

Алгоритм 3. Алгоритм стохастического порождения суперпозиций.

Вход:

- Множество примитивных функций G .
 - Регрессионная выборка D .
 - N_{max} — максимальное число одновременно рассматриваемых суперпозиций.
 - I_{max} — максимальное число итераций алгоритма.
 - F_{min} — минимальная приспособленность суперпозиций.
- (1) Инициализируется начальный массив суперпозиций случайным образом.
 - (2) Оптимизируются параметры суперпозиций алгоритмом \mathcal{LM} .
 - (3) Для каждой еще не оцененной суперпозиции f рассчитывается значение функции ошибки S_f на обучающей выборке, и ставится в соответствие значение F_f , характеризующее «приспособленность» суперпозиции f : $F_f = \frac{1}{1+S_f}$. Таким образом, чем лучше результаты суперпозиции, тем ближе значение ее приспособленности к 1, и, наоборот, чем хуже — тем ближе к 0. Если данная суперпозиция точно описывает данные, то значение ее приспособленности в точности равно единице.
 - (4) Массив суперпозиций сортируется согласно их приспособленности.
 - (5) Наименее приспособленные суперпозиции удаляются из массива до тех пор, пока его размер не станет равен N_{max} .
 - (6) Отбирается некоторая часть наименее приспособленных суперпозиций. У этой части происходит случайная замена одной функции или свободной переменной на другую. Замена такова, чтобы сохранилась структура суперпозиции, а именно — в случае замены функции сохраняется арифметичность, а свободная переменная заменяется только на другую свободную переменную. При этом исходные суперпозиции сохраняются в множестве.
 - (7) Повторяются шаги 3 — 4.
 - (8) Производится случайный обмен поддеревьями наиболее приспособленных суперпозиций. При этом исходные суперпозиции сохраняются в массиве.
 - (9) Повторяются шаги 3 — 4.
 - (10) Проверяются условия останова: если либо число итераций больше I_{max} , либо в массиве есть хотя бы одна суперпозиция с приспособленностью больше, чем F_{min} , то алгоритм останавливается, и результатом является наиболее приспособленная суперпозиция, иначе переход к шагу 2.

5.2. Данные. В вычислительном эксперименте используются данные о волатильности опционов Brent Crude Oil.

Волатильность σ — финансовый показатель, характеризующий изменчивость цены. Волатильность является важным финансовым показателем и используется в управлении финансовыми рисками, так как представляет собой меру риска использования финансового инструмента (некоторого финансового документа, передача которого обеспечивает получение денежных средств) за некоторый заданный промежуток времени.

Волатильность пропорциональна стандартному отклонению σ_{SD} стоимости финансового инструмента и обратно пропорциональна квадратному корню из временного периода, обычно измеряемого в годах:

$$\sigma = \frac{\sigma_{SD}}{\sqrt{P}}.$$

Если P измеряется в годах, то σ называется среднегодовой волатильностью, и волатильность σ_T за интервал времени T , выраженный в годах, рассчитывается по формуле:

$$\sigma_T = \sigma\sqrt{T}.$$

Например, если стандартное отклонение стоимости в течение дня составляет 0.01, а в году 252 торговых дня, то волатильность будет равна:

$$\sigma = \frac{0.01}{\sqrt{\frac{1}{252}}} \approx 0.159.$$

Отсюда волатильность за месяц будет равна:

$$\sigma_M = 0.159\sqrt{\frac{1}{12}} \approx 0.0459.$$

СПИСОК ЛИТЕРАТУРЫ

- [1] P. Barmapalexis, K. Kachrimanis, A. Tsakonas, and E. Georgarakis. Symbolic regression via genetic programming in the optimization of a controlled release pharmaceutical formulation. *Chemometrics and Intelligent Laboratory Systems*, 107(1):75–82, 2011.
- [2] Peng Shi and Wei Zhang. A copula regression model for estimating firm efficiency in the insurance industry. *Journal of Applied Statistics*, 38(10):2271–2287, October 2011.
- [3] Selen Onel, Abe Zeid, Sagar Kamarthi, and Meredith Hinds Harris. Analysis of risk factors and predictive model for recurrent falls in community dwelling older adults. *Int. J. of Collaborative Enterprise*, 1:359–380, February 01 2011.
- [4] J. W. Davidson, D. A. Savic, and G. A. Walters. Symbolic and numerical regression: experiments and applications. In Robert John and Ralph Birkenhead, editors, *Developments in Soft Computing*, pages 175–182, De Montfort University, Leicester, UK, 29–30 June 2000. 2001. Physica Verlag.
- [5] Claude Sammut and Geoffrey I. Webb. Symbolic regression. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, page 954. Springer, 2010.
- [6] John R. Koza. Genetic programming. In James G. Williams and Allen Kent, editors, *Encyclopedia of Computer Science and Technology*, volume 39, pages 29–43. Marcel-Dekker, 1998. Supplement 24.
- [7] John R. Koza. Introduction to genetic algorithms, August 15 1998.
- [8] Ivan Zelinka, Zuzana Oplatkova, and Lars Nolle. I. ZELINKA et al: ANALYTICAL PROGRAMMING ... ANALYTIC PROGRAMMING – SYMBOLIC REGRESSION BY MEANS OF ARBITRARY EVOLUTIONARY ALGORITHMS, August 14 2008.
- [9] Ю. Н. Павловский. *Имитационные модели и системы*. Фазис, 2000.
- [10] D. W. Marquardt. An algorithm for least-squares estimation of non-linear parameters. *Journal of the Society of Industrial and Applied Mathematics*, 11(2):431–441, 1963.
- [11] J. J. Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In *G.A. Watson, Lecture Notes in Mathematics 630*, pages 105–116. Springer-Verlag, Berlin, 1978. Cited in Åke Björck’s bibliography on least squares, which is available by anonymous ftp from [math.liu.se](http://math.liu.se/pub/references) in `pub/references`.

МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ, ФУПМ, КАФ. «ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ»