

# 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. В вычислительном эксперименте приводятся результаты для задачи моделирования волатильности опционов.

## 1. ВВЕДЕНИЕ

В ряде приложений [1] [2] [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  $\log x - \frac{e^x}{x}$ . 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.

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

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 <sup>1</sup> 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.

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

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.

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 дан набор  $(\mathbf{x}_i, y_i) \mid i \in \{1, \dots, N\}$ ,  $\mathbf{x}_i \in R^n, y_i \in R$ . Предполагается, что  $\mathbf{y} \in \mathcal{N}$ <sup>2</sup>.

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

**2.2. Теоретическая часть.** Пусть  $G = \{g_1, \dots, g_{n_g}\}$  — множество данных примитивных функций. Требуется:

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

## 3. ПУТИ РЕШЕНИЯ ЗАДАЧИ: ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

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

- В вершинах дерева находятся соответствующие примитивные функции.
- Число дочерних вершин у некоторой вершины равно аргументности соответствующей функции, а их порядок (в смысле обхода в глубину) соответствует порядку аргументов соответствующей функции.
- В листьях дерева находятся свободные переменные.

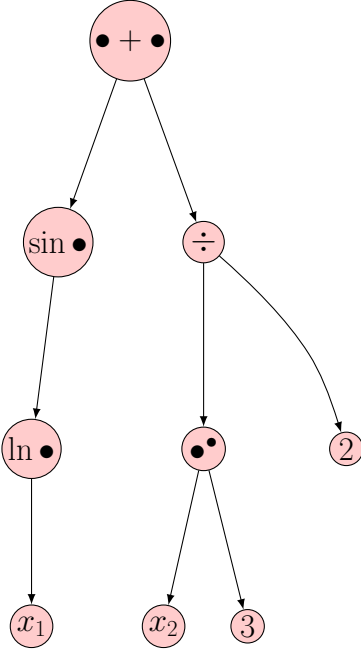
---

<sup>2</sup>А правда, зачем?

- Порядок вершин в смысле уровня относительно корня дерева 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 суперпозиции в некоторой точке эквивалентно подстановке соответствующих значений свободных переменных в граф выражения.

Для примера рассмотрим граф, соответствующий суперпозиции  $\sin(\ln x_1) + \frac{x_2^3}{2}$ :



**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 1.** 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  $\mathcal{A}$  очевидным образом обобщается на множество  $G$ , содержащее функции произвольной (но имеющей конечную верхнюю грань) ариности. Действительно, для такого обобщения достаточно строить аналогичным образом вспомогательные множества для этих функций.

**Утверждение 1.** Алгоритм  $\mathcal{A}$  корректен: любую конечную суперпозицию он действительно породит 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  $\Gamma_f$  и рекурсивно пройти от вершин к листьям, составляя цепочку соотношений на номера итераций по следующим правилам:

- Если вершина, полученная на  $i$ -ой итерации — унарная функция, то это функция от выражения, полученного на  $(i-1)$ -ой итерации.
- Если вершина, 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 узел 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.  $\square$

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:

$$U_i = g_u \circ (\alpha f + \beta)$$

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

Здесь параметры  $\alpha, \beta$  зависят только от комбинации  $g_u, f$  (или  $g_b, f, h$  для  $\alpha, \beta, \psi, \phi$ ). 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  $\alpha, \beta, \psi, \phi$  мы не изменяем мощности получившегося множества суперпозиций, поэтому алгоритм и выводы из него остаются корректны. В частности, исходный алгоритм является частным случаем данного при  $\alpha \equiv \psi \equiv 1, \beta \equiv \phi \equiv 0$ .

$\alpha, \beta, \psi, \phi$  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  $\frac{1}{x}: 1 = \alpha \text{ id } x + \beta \mid \alpha = 0, \beta = 1$ .

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 аргументности.

**3.2. Бесконечные суперпозиции.** 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.

**3.3. 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.

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 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 <sup>3</sup>.

**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 коммутарующих 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.<sup>4</sup>

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  $f = (+)$ ,  $g = (\times)$ ,  $s(i) = i + 1$ .

---

<sup>3</sup>А было бы вычислительно эффективно — все равно, похоже, было бы NP-сложной задачей

<sup>4</sup>Кстати, может, можно придумать какой-нибудь оптимальный алгоритм поиска расходящихся точек? Ну или эвристику хотя бы, позволяющую перебирать данные точки не в лоб, а более целенаправленно и позволяя находить точки, в которых значения различаются, более быстро.

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-МАРКВАРДТА И МУЛЬТИСТАРТ

Алгоритм Левенберга-Марквардта  $\mathfrak{LM}$  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:

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

где  $\beta$  — вектор параметров модели (суперпозиции)  $f$ .

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

Перед началом работы алгоритма задается начальный вектор параметров  $\beta_0$ . На каждой итерации этот вектор заменяется новой оценкой,  $\beta + \delta$ . Для определения  $\delta$  используется линейное приближение функции:

$$\mathbf{f}(\mathbf{x}, \beta + \delta) \approx \mathbf{f}(\mathbf{x}, \beta) + \mathbf{J}\delta,$$

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

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

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

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

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

Левенберг предложил заменить  $(\mathbf{J}^T \mathbf{J})$  на  $(\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I})$ , где  $\lambda$  — 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  $\delta$  выглядит как:

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

Решая это уравнение, получаем окончательное выражение для  $\delta$ :

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

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  $S(\beta)$  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 2.** *Алгоритм стохастического порождения суперпозиций.*

*Вход:*

- Множество примитивных функций.
  - Множество точек обучающей выборки.
  - $N_{max}$  — 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  $\sigma$  — 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  $\sigma_{SD}$  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  $\sigma$  называется среднегодовой волатильностью, и волатильность  $\sigma_T$  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»