

ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ  
ПРЯМОГО РАСПРОСТРАНЕНИЯ: ОПИСАНИЕ С ПОМОЩЬЮ  
РАСШИРЕННЫХ МАШИН ТЬЮРИНГА И ПРИМЕНЕНИЕ В  
АЭРОДИНАМИКЕ

## Оглавление

|  |            |
|--|------------|
| <b>Введение .....</b>  | <b>4</b>   |
| <b>Глава 1. Теоретический базис описания, построения и трансформации ИНС прямого распространения .....</b>                               | <b>10</b>  |
| 1.1. Расширенные машины Тьюринга.....  | 10         |
| 1.1.1. Разновидности расширенных машин Тьюринга .....  | 15         |
| 1.1.2. Представимость различных формализмов моделирования.....   | 29         |
| 1.2. Работа с ИНС прямого распространения в системе, описываемой в пределе РМТ .....   | 31         |
| 1.3. Построение интерполяционных моделей в системе, описываемой в пределе РМТ .....  | 36         |
| 1.3.1. Построение модели коэффициента преломления воды в инфракрасном дальнем диапазоне .....  | 41         |
| 1.3.2. Построение модели скорости выполнения фрагмента программы в метаслое .....  | 43         |
| Выводы к первой главе.....   | 47         |
| <b>Глава 2. Нейросетевое моделирование турбулентной вязкости в задачах, связанных с получением общей картины воздушных потоков .....</b> | <b>49</b>  |
| 2.1. Обзор подходов к моделированию картины воздушных потоков.....   | 49         |
| 2.2. Обзор моделей турбулентности.....   | 55         |
| 2.3. Модельные задачи .....  | 63         |
| 2.3.1. Распространение загрязнителя на улицах города.....  | 63         |
| 2.3.2. Обтекание одиночного здания .....   | 66         |
| 2.4. Локальная и интегро-локальная нейросетевые модели турбулентной вязкости .....   | 68         |
| 2.5. Контроль погрешности вычислений .....   | 76         |
| 2.5.1. Анализ разностных схем .....  | 78         |
| 2.5.2. Схема контроля погрешности счета.....   | 80         |
| Выводы ко второй главе.....  | 83         |
| <b>Глава 3. Получение приближенных выражений для ИНС.....</b>  | <b>85</b>  |
| 3.1. Общая постановка задачи .....   | 85         |
| 3.2. Первичное упрощение сети.....   | 91         |
| 3.3. Основное упрощение сети.....  | 93         |
| 3.4. Новая стратегия получения оптимальной комбинации заместительных функций.....  | 99         |
| Выводы к третьей главе .....   | 100        |
| <b>Глава 4. Символическое комбинирование приближенных выражений. Распараллеливание ..</b>  | <b>101</b> |
| 4.1. Обзор пакетов символьной математики .....   | 102        |
| 4.2. Обзор технологий распараллеливания вычислений .....   | 104        |
| 4.3. Общее описание иерархии классов символического ядра и их основных элементов .....   | 109        |
| 4.4. Общее ускорение алгоритмов.....   | 112        |
| 4.4.1. Применение маски .....  | 113        |
| 4.4.2. Новая схема контроля времени при делении полинома на полином .....  | 117        |
| 4.5. Новый алгоритм распараллеливания вычислений .....   | 119        |
| 4.5.1. О порождении подзадач .....   | 119        |
| 4.5.2. Векторное распараллеливания при делении полинома на полином.....  | 121        |

|   |            |
|---|------------|
| 4.5.3. Экспериментальное исследование схемы распараллеливания .....                                 | 122        |
| 4.5.4. Аналитическое обоснование временной зависимости от доли параллелизма «портфеля задач» .....  | 124        |
| 4.6. Краткое описание разработанного программного кода .....  | 126        |
| 4.7. Анализ эффективности предложенных алгоритмов .....   | 126        |
| Выводы к четвертой главе .....  | 129        |
| <b>Глава 5. Апробация предложенных подходов. Компактизация полей физических величин .....</b>       | <b>131</b> |
| 5.1. Апробация символического ядра .....  | 131        |
| 5.2. Апробация полученных упрощенных моделей .....  | 133        |
| 5.3. Краткие рекомендации по применению результатов работы в САПР объектов городской застройки .... | 138        |
| 5.4. Компактизация полей физических величин в задачах аэродинамики.....                             | 140        |
| 5.4.1. Кластеризация .....  | 141        |
| 5.4.2. Построение нейросетевых описаний в кластерах .....   | 142        |
| 5.4.3. Экспериментальная часть.....   | 145        |
| Выводы к пятой главе.....   | 151        |
| <b>Заключение.....</b>  | <b>153</b> |
| <b>Библиографический список.....</b>  | <b>157</b> |

## Введение

Эта работа посвящена искусственным нейронным сетям (ИНС) прямого распространения [55, 66]. Такие сети достаточно хорошо известны и широко применяются при решении ряда проблем, сводимых к задачам интерполяции, экстраполяции и классификации. Это проблемы моделирования различных явлений, прогнозирования, распознавания образов, сжатия данных, автоматического управления, искусственного интеллекта и другие. При этом существенным недостатком классических подходов к работе с ИНС является существующий разрыв между описанием, построением и трансформацией ИНС в том смысле, что описание ИНС выполняется с помощью классических математических соотношений, а построение требуемой сети и ее трансформация в процессах обучения и функционирования являются алгоритмическими процессами, которые затруднительно описать в чисто математической форме. Этот разрыв создает существенные неудобства и снижает эффективность работы, если ставится, например, задача разработки самоорганизующейся системы на базе ИНС. Очевидно, что основная проблема заключается в отсутствии некоей единой математико-алгоритмической базы для ИНС прямого распространения, что и определяет *актуальность* рассмотрения данного вопроса в настоящей работе. На практике указанная выше проблема обычно решается [45, 50, 51, 60] переходом к чисто алгоритмическому *внешнему* подходу<sup>1</sup>, который вполне позволяет запрограммировать основные математические соотношения, представляющие функционирование и обучение ИНС. В данной же работе предлагается повысить эффективность работы с ИНС прямого распространения

---

<sup>1</sup> Отметим также комбинированный подход, развитый, например, в работе [36], где нейронная сеть и любые дополнительные управляющие (а также, возможно, обучающие/трансформирующие) компоненты сводятся к графам, которые комбинируются неким способом, отражающим взаимодействие между сетью и данными компонентами. Такие сети в указанной работе обозначены как *нейроподобные*. При всей привлекательности данного подхода нельзя не отметить его громоздкость и недостаточную формализацию процесса трансформации, который, вероятно, должен производиться некими программными графами, присоединенными к нейронной сети. Нам же требуется подход, который позволил бы строго и формально непротиворечиво реализовать в рамках единого формализма не только описание/обучение, но и построение/трансформацию.

(это *первая главная цель* данной работы) путем разработки математико-алгоритмического формализма для таких ИНС. Особенно важно при этом разумно использовать ресурсы какой-либо из существующих систем моделирования (которая будет являться, по сути, инструментальной средой для построения, трансформации и запуска ИНС) на базе тех или иных формализмов, к которым должен быть близок разрабатываемый формализм. Достаточно хорошим вариантом представляется задействование ресурсов системы порождения программ PGEN++ [45, 52], которая, фактически, является системой поддержки объектно-событийных моделей (ОСМ) [45, 52]. В таком случае разрабатываемый формализм должен являться базовым (или предельным) по отношению к ОСМ. Поскольку ОСМ в пределе сводятся к машинам Тьюринга (МТ), разрабатываемый формализм, вероятно, будет неким расширением МТ.

Любопытно отметить, что иногда задача ставится противоположным образом, когда ИНС выбирается в качестве базового формализма [41] для описания и решения задач, определенных в рамках иных дискретных, непрерывных и дискретно-непрерывных подходов. В частности, в работе [41] ставится задача построения нейропроцессора на рекуррентных ИНС, который в состоянии решать задачи для цифровых вычислителей (дискретных систем). При всей возможной оправданности такого подхода при решении ряда интеллектуальных задач, нельзя не признать, что при решении более частных задач все же *более удобно строить дискретные и дискретно-непрерывные модели/машины, успешно описывающие непрерывные функции* (применяемые в ИНС и реализуемые с помощью ИНС) с помощью дискретизации. В частности на данном подходе базируется такая мощная отрасль науки как вычислительная математика.

Исходя из вышесказанного, можно сделать вывод, что тематика данной работы не теряет актуальности и в настоящее время.

*Второй главной целью* данной работы является повышение эффективности решения ряда важных практических задач аэродинамики путем адекватного применения ИНС. В качестве таких задач предлагается рассмотреть следующие проблемы:

а) приближенное быстрое моделирование турбулентной вязкости в задачах, связанных с получением картины воздушных потоков на улицах города и в окрестности предприятия;

б) компактное хранение приближенных (сжатых с потерями) данных о распределении скоростей воздушных потоков и турбулентной вязкости, изменяющихся в зависимости от некоторого параметра, в задачах, указанных в предыдущем пункте.

Обе указанные проблемы достаточно *актуальны*. Рассмотрим *первую проблему*. При проектировании объектов городской застройки существенное значение имеет точное знание картины распределения воздушных потоков при различных направлениях ветра. Во-первых, это требуется для расчета потенциальной ветровой нагрузки, во-вторых, желателен анализ проектного решения по фактору экологической безопасности, подразумевающий расчет предполагаемых концентраций твердых пылевых и газообразных загрязнителей, переносимых воздушными потоками. Источниками таких загрязнителей является, прежде всего, автомобильный транспорт, кроме того, ими могут быть различные предприятия (например, ТЭЦ), расположенные преимущественно в пределах городской черты. Неблагоприятная экологическая ситуация часто является следствием принятия проектных решений, плохо обоснованных с точки зрения экологической безопасности. Проблема, обычно, заключается в использовании недостаточно точных оценок предполагаемых уровней загрязнений при анализе проектных вариантов.

Повышение точности оценок ветровой нагрузки и уровней загрязнений требует применения достаточно сложных методик математического моделирования воздушных потоков, основанных на решении систем дифференциальных уравнений газовой динамики. Такого рода методики достаточно хорошо известны и отличаются большим количеством требуемых расчетов, соответственно и потенциально очень большим временем счета. Данная проблема обычно решается путем применения многопроцессорных/многоядерных суперкомпьютеров (специализированных или кластерных, наиболее популярных в настоящее время, систем). Однако такие системы не всегда доступны, поэтому актуальна задача упрощения

сложных методик моделирования воздушных потоков, основанных на использовании систем дифференциальных уравнений, путем замены части таких уравнений более простыми приближенными соотношениями. Наиболее перспективной представляется замена той части модели, которая относится к моделированию турбулентности. Решение задачи такого рода позволит ускорить моделирование даже на обычных компьютерах с одним или несколькими ядрами.

Далее отметим, что в известных автору отечественных и зарубежных программных продуктах, позволяющих проводить достаточно точное моделирование воздушных потоков (например, Fluent [83], Star-CD [19], AirEcology-P [48, 49]), используются, преимущественно, либо простые алгебраические, либо сложные и более точные интегральные модели турбулентности. В рамках предлагаемого нами подхода могут использоваться промежуточные, компромиссные модели на базе ИНС прямого распространения. При этом ИНС интегрируется непосредственно в математическую модель аэродинамической задачи. Поскольку ИНС, видимо, будет обучаться на конечных данных о готовых распределениях турбулентной вязкости, логично предположить, что такого рода нейросетевая подмодель позволит получить более быстрое общее схождение к решению. Целесообразно рассмотреть вопрос о получении для таких подмоделей некоторых упрощенных (по сравнению с исходной ИНС) соотношений, приближающих ее поведение на основных рабочих участках входных данных. То есть, необходимо рассмотрение задачи анализа и упрощения ИНС. Это позволит дополнительно ускорить моделирование воздушных потоков. Отметим, что такого рода соотношения должны применяться с известной осторожностью, по меньшей мере, необходим адекватный контроль погрешности расчета по аэродинамической модели. Такого рода задача также может быть рассмотрена в данной работе.

Рассмотрим *вторую проблему*. В практике расчета экологической безопасности задача получения распределения загрязнителей может решаться неоднократно, с варьированием как направления и скорости ветра, так и расположения и интенсивности источников загрязнителей. При этом целесообразно прибегнуть к однократному решению повторяющейся подзадачи расчета картины воздушных пото-

ков для каждого конкретного направления/скорости ветра с сохранением результатов расчета в файлах. Отметим, что в некоторых случаях указанная подзадача может решаться путем интерполяции по данным из сохраненных результатов расчета для иных, близких к текущему, направлений/скоростей ветра. При больших размерах расчетной сетки (от нескольких сотен до миллионов узлов для областей сложной формы, включающих, например, несколько улиц со строениями разной высоты) объем сохраняемых конечных данных даже для одной подзадачи при использовании вещественных значений одинарной точности (4-байтных) может достигать нескольких десятков мегабайт.

Учитывая, что число подзадач определяется произведением количества возможных направлений ветра ( $15 \div 30$ ) на количество градаций скорости ветра ( $10 \div 15$ ), его величина может составлять от 150 до 450 подзадач. В результате общий объем сохраняемых данных может достичь одного гигабайта и выше.

Генерация компактных описаний получаемых вариативных полей физических величин (в частности, компонент скорости и турбулентной вязкости), соответственно, является актуальной задачей. Как показывает практика, для таких данных максимальный коэффициент сжатия без потерь, полученный при использовании достаточно качественного алгоритма (например, PPMD или LZMA) не превышает  $0,50 \div 0,65$ . Для получения большей степени сжатия (например, с коэффициентом  $0,01 \div 0,02$ ) необходимо воспользоваться приближенными методами (сжатие с потерями), например, интерполяцией данных ИНС с сохранением их весовых коэффициентов. Применение ИНС также решит проблему получения интерполированных значений физических величин для случая таких направлений/скоростей ветра, для которых нет сохраненных (сжатых или несжатых) данных. Может потребоваться построение компактных описаний на базе целых комплексов ИНС, построенных для кластеров схожих фрагментов данных. Не исключено, что может понадобиться некая специальная процедура выбора оптимальных комбинаций ИНС для таких данных.

Резюмируя, *целями данной работы* являются:

1. Повышение эффективности описания, построения и трансформации ИНС.

2. Повышение эффективности решения некоторых важных прикладных задач из области аэродинамики путем применением ИНС.

Для достижения поставленных целей сформулируем следующие *задачи*:

1. Разработка теоретического базиса для описания, построения и трансформации ИНС

2. Разработка технологии анализа и упрощения ИНС на примере задач аэродинамики при адекватном контроле погрешности решения.

3. Разработка новых применений ИНС в задачах аэродинамики, повышающих эффективность получения и хранения физических данных.

## Глава 1. Теоретический базис описания, построения и трансформации ИНС прямого распространения

Первой целью данной работы является повышение эффективности описания, построения и трансформации искусственных нейронных сетей прямого распространения. С этой целью в данной главе решается несколько более общая задача — поиск и расширение какого-либо из известных базовых формализмов для описания и моделирования последовательных и параллельных процессов, которые вполне характерны для искусственных нейронных сетей (параллельны процессы, протекающие в нейронах одного слоя, последовательны процессы, относящиеся к разным слоям). Помимо теоретической ценности такого исследования, могут быть предпосылки и для его практического применения: например, если для найденного формализма существует программная система моделирования, которая в тех или иных аспектах может стать базовым инструментом для работы с нейронными сетями прямого распространения.

Второй задачей данной главы, после того, как базовый формализм и программная система будут определены, будет, вероятно являться демонстрация возможностей такой системы по применению нейронных сетей для решения какой-либо практически важной проблемы, например, построения комбинированных интерполяционных моделей с применением нейронных сетей.

### 1.1. Расширенные машины Тьюринга

Математические формализмы описания *непрерывных* последовательных и параллельных процессов (функциональные соотношения [в том числе различные интерполяторы, такие как нейронные сети [66] или каскады полиномов, полученные методом группового учета аргументов (МГУА, [23])], системы алгебраических уравнений [САУ, в том числе линейных — СЛАУ], обыкновенные дифференциальные уравнения (ОДУ), дифференциальные уравнения в частных производных (ДУЧП), оптимизационные модели, модели системной динамики Форре-

стера [67], марковские цепи) либо имеют аналитическое решение, либо сводятся к дискретному численному решению. В частности, интегрирование ДУЧП обычно проводится с применением разностных схем на расчетных сетках. Данные формализмы не содержат алгоритмического компонента, поэтому их обучение/модификация/моделирование ведутся внешними по отношению к ним средствами, что несколько снижает их гибкость и описательную ценность.

К формализмам описания *дискретных* последовательных и параллельных процессов относятся объектные (множество объектов, обменивающихся сообщениями в моделирующей среде, как это сделано, например, в системе Флора/FloraWare) и событийные модели (чисто событийные или объектно-событийные), классические автоматы, клеточные автоматы, различные алгоритмические машины (Тьюринга, Поста), сети Петри, системы массового обслуживания. Ряд этих формализмов уже содержит алгоритмический компонент, что существенно повышает их гибкость и описательную ценность.

Наиболее перспективным, вероятно, является формализм описания *кусочно-линейных* процессов. В первую очередь назовем модели Бусленко [8], которые могут представлять как дискретные, так и непрерывные (разбивая их на серию кусочно-линейных процессов малой длительности) процессы, и при этом содержат алгоритмический компонент. Модели Бусленко, к сожалению, плохо определяют поведение системы при наличии циклических связей — возможны неоднозначности толкования порядка срабатывания агрегатов, иногда требующие введения дополнительных допущений. Кроме того, модели Бусленко не являются конструируемыми, алгоритмы их построения и модификации являются внешними по отношению к модели. Это существенно затрудняет описание динамики сложных систем, как непрерывных (например, процесса построения нейронной сети по данным или процесса перестроения сетки при моделировании на основе уравнений в частных производных), так и дискретных (например, процесса ввода новых элементов в систему массового обслуживания в процессе ее функционирования). Далее отметим некоторые разновидности специальных машин Тьюринга [6, 9], в которых присутствует эволюционный оператор общего вида. Однако в этих фор-

мализмах приоритет отдан именно эволюционированию, в ходе которого возможны «скачки» от одного из достигнутых состояний к другому. В эту парадигму, к сожалению, не очень хорошо вписываются такие характерные для ИНС процессы как обучение и переобучение — это итеративные процессы, заранее не специфицирующие конкретных значений внутренних переменных для исходного и конечного состояний итерации<sup>1</sup>. Поэтому в нашем случае приоритет должен быть отдан дискретному процессу, а непрерывное (кусочно-линейное) изменение производится между состояниями. В такую схему хорошо вписывается как обучение/переобучение, так и функционирование ИНС.

По нашему мнению, наилучшим вариантом кусочно-линейных моделей могут быть те или иные *универсальные алгоритмические машины*, в которые следует лишь ввести дополнительный линейный оператор перехода в промежутке между обработкой состояний или событий. Здесь особо следует выделить машины Тьюринга (МТ, для которых сформулирован тезис Тьюринга-Черча) и объектно-событийные модели (ОСМ, для которых также утверждается реализуемость любого алгоритма, [52]). Однако прямое использование классических МТ как для описания, так и применения моделей является достаточно сложной и неудобной задачей. Кроме того, они не являются конструирующими. Для непосредственного применения более пригодны ОСМ, которые лишены активирующих циклических связей (создающих сложности в моделях Бусленко), являются конструирующими и сформулированы в двух трактовках: абстрактной (здесь их описательная сила эквивалентна возможностям формализма МТ) и реальной, позволяющей использовать ОСМ для разработки описаний/моделей и программ. Дополнительно отметим, что для ОСМ существует программная система моделирования PGEN++, которая, после расширения формализма ОСМ, вполне может стать, например, платформой для работы с искусственными нейронными сетями.

---

<sup>1</sup> Даже если описать обучение каким-либо простым математическим оператором, например, минимизации функционала ошибок или решения системы дифференциальных уравнений, реализация такого оператора в общем случае все же будет выполнена по итеративной схеме (в соответствии с базовыми принципами вычислительной математики), поскольку достаточно затруднительно построить универсальный непрерывный (аналоговый) решатель приемлемой для наших задач сложности.

Итак, расширим ОСМ линейным оператором перехода между состояниями. Кроме того, не исключена и иная, дополнительная проработка базовой теории ОСМ с точки зрения повышения ее значимости, четкости и логичности.

Расширение возможностей реальных ОСМ, основанных на программировании на ЯВУ, осуществляется, вообще говоря, достаточно элементарным образом. Нас более интересует теоретический аспект, а именно — расширение возможностей абстрактных предельных ОСМ. Заметим, что абстрактные последовательные предельные ОСМ представляют собой МТ. Поэтому расширение описательных возможностей ОСМ должно означать, помимо всего прочего и расширения понятия МТ.

Теперь вспомним *абстрактные параллельные предельные ОСМ*. Ранее автором был введен ряд утверждений о том, что их роль может играть группа связанных объектов, каждый из которых сводим к МТ. Было бы логично сделать следующий шаг — отказаться от такой трактовки как базовой (которая потребовала бы от нас применения сетей МТ и, возможно, многоленточных МТ, см., например, [104]) и, вместо этого, расширить формализм МТ, введя в него ряд особенностей, характерных для *реальных последовательных и предельных параллельных ОСМ*, то есть ПППВ. Это придало бы теории ОСМ большую строгость и логичность. Заметим, что в таком случае мы могли бы говорить об одном из вариантов параллельной машины Тьюринга, отличающимся от известных аналогов [85, 101, 102], основанных на введении понятия множественного перехода из текущего состояния в несколько иных состояний с возможным порождением дополнительных головок, несколько большей конструктивностью описания параллелизма — в нашем случае не требуется жесткое описание всех вариантов параллелизма состояний, поскольку предполагается динамическое определение множества состояний, в которые переходят головки МТ при их параллельном инициировании.

В дальнейшем, говоря о предлагаемых нами модификациях МТ, будем использовать термин РМТ — расширенные машины Тьюринга.

Теперь, когда мы определили, что абстрактными предельными ОСМ являются МТ и РМТ, вторым шагом, исходя из вышеизложенного, должно стать даль-

нейшее расширение понятия МТ путем включения в нее линейного лямбда-оператора (лямбда-функции) перехода между состояниями. Как уже было отмечено выше, работы на эту тему существуют (в частности, в работах [6, 9] в МТ включается оператор перехода общего вида, аналогичный подход используется для элементарных мемпроцессоров [97]), однако сформулированные в них формализмы не вполне подходят для решаемого нами круга задач. Целесообразно сразу разделить два случая: а) когда оператор является локальным для РМТ и фиксированным по структуре, б) когда оператор может быть как локальным, так и глобальным (то есть может содержать ссылки на элементы в разных РМТ), при этом не фиксирован. Первый случай — частный, наиболее употребимый, например, для одиночных РМТ, описывающих работу некоторого простого интерполятора. Второй случай — более общий, позволяющий, в частности, описывать задачи моделирования в частных производных на сетке, каждому узлу которой соответствует РМТ, и выполняется решение некоторой глобальной СЛАУ, представляющей разностный оператор для всей сетки. Нефиксированный оператор, видимо, должен генерироваться РМТ динамически. Целесообразно представить его, например, в виде текстовой записи множества уравнений, ссылающихся на именованные элементы на лентах РМТ. Поэтому РМТ следует, в таком случае, расширить возможностью генерации символьных цепочек. Предполагается, что РМТ способна решать системы уравнений, записанные в математической текстовой форме. При этом полученная разновидность РМТ прекрасно описывает важный частный случай — некую предельную ОСМ, применяемую для порождения программ. Действительно, если порожденная цепочка будет содержать некую специальную пометку, обозначающую, что она не должна распознаваться как система уравнений, то ее вполне можно интерпретировать как порожденную программу.

Далее, с теоретической же точки зрения, вероятно, было бы также интересно представить РМТ предельную систему объектов. Здесь, соответственно, следовало бы задуматься и о динамическом конструировании (модификации) моделей, решив РМТ представлять самомодифицирующиеся предельные составные ОСМ.

### 1.1.1. Разновидности расширенных машин Тьюринга

Руководствуясь понятием о реальных предельных ОСМ [52], необходимо расширить формализм МТ понятием плана исполнения, в котором, видимо, будут храниться спланированные состояния и барьеры, разделяющие группы этапов плана, а также возможностью параллельного исполнения процессов в рамках каждой группы плана. Кроме того, исходя из сказанного ранее, необходимо, для начала, ввести в формализм МТ понятие простейшего линейного оператора перехода между состояниями, который, видимо, будет описываться последовательностью коэффициентов и свободных членов соответствующей СЛАУ. Таким образом, в первом приближении мы вводим понятие о трех расширениях МТ:

а) *элементарной РМТ (ЭлРМТ)*, реализующей простейшую последовательную работу с планом исполнения,

б) *эволюционной РМТ (ЭвРМТ)*, в которую будет введен линейный оператор перехода,

в) *параллельной РМТ (ПарРМТ)*, реализующей деление плана на группы и запуск исполнения процессов группы в соответствии с планом в параллель.

Опишем эти три первые разновидности формально.

Сконструируем *элементарную РМТ*. Пусть, как и классическая машина Тьюринга она обладает лентой, алфавитом  $A$ , множеством состояний  $Q$ , правилами перехода  $\delta$  и начальным состоянием  $p_0 \in Q$ . Правила перехода имеют вид

$$\delta = Q \times A \rightarrow Q \times A \times S \times P_1 \cup \{-\},$$

где  $S = \{R, L, E\}$ ,  $P_1 = (\{\uparrow\} \times Q) \cup (\{\downarrow\} \times Q)$ .

Пусть в начале работы ЭлРМТ головке соответствует начальное состояние  $p_0$ . Также пусть ЭлРМТ имеет *план работы*  $L$ , представляющий собой дек. План хранит состояния  $q \in Q$ . Исполнение команды, содержащей  $\uparrow q_A$ , вставляет состояние  $q_A$  в начало плана (в позицию  $L_1$ , при этом элементы плана сдвигаются вправо). Исполнение команды, содержащей  $\downarrow q_B$ , вставляет состояние  $q_B$  в конец плана (в позицию  $L_{n+1}$ , где  $n$  — длина плана до вставки состояния). При исполне-

нии оператора вида  $q_i a_j \rightarrow -$  из плана  $L$  извлекается первый элемент, головка переводится в состояние, соответствующее извлеченному элементу. Если же план пуст, ЭлРМТ завершает работу.

ЭлРМТ способна *гибко планировать последовательность переходов* — использовать разнообразные линейные подходы к изменению порядка исполнения блоков программы, например, близкие к применению подпрограмм. В качестве примера можно привести небольшую программу, которая читает с ленты строку из  $N$  двоичных символов («0» и «1») и записывает ее в обратном порядке после символа « $\delta$ » (примыкает справа к строке), причем затирать исходную строку запрещено. При этом считаем, что пустое пространство ленты заполнено символами « $\lambda$ ». В таких условиях достаточно непростым является вопрос о переходе к очередному символу исходной строки, который и предлагается разрешить с помощью планирования  $N$  переходов к различным строкам подпрограммы перемещения головки влево от символа « $\delta$ ». Тогда с увеличением  $N$  на единицу программа будет увеличиваться всего на две строки: планирующую и перемещающую на шаг влево. *Для сравнения:* в классической МТ увеличение  $N$  на единицу потребовало бы или добавления  $N$  строк или добавления нескольких строк и одного дополнительного символа алфавита, который детерминировал бы количество переходов влево на очередном шаге. Как в том, так и в другом случае решение было бы более громоздким.

На рис. 1.1 приведен пример программы для  $N = 3$ . Считается, что в начале работы программы ( $p_0 = q_1$ ) головка ЭлРМТ установлена на символ « $\delta$ ».

| Состояния       | Символы алфавита                  |                                   |   |  |
|-----------------|-----------------------------------|-----------------------------------|---|--|
|                 | 0                                 | 1                                 | $\delta$                                  | $\lambda$                                  |
| q <sub>1</sub>  | 0Eq <sub>2</sub> ↓q <sub>14</sub> | 1Eq <sub>2</sub> ↓q <sub>14</sub> | $\delta$ Eq <sub>2</sub> ↓q <sub>14</sub> | $\lambda$ Eq <sub>2</sub> ↓q <sub>14</sub> |
| q <sub>2</sub>  | 0Eq <sub>3</sub> ↓q <sub>13</sub> | 1Eq <sub>3</sub> ↓q <sub>13</sub> | $\delta$ Eq <sub>3</sub> ↓q <sub>13</sub> | $\lambda$ Eq <sub>3</sub> ↓q <sub>13</sub> |
| q <sub>3</sub>  | 0Eq <sub>4</sub> ↓q <sub>12</sub> | 1Eq <sub>4</sub> ↓q <sub>12</sub> | $\delta$ Eq <sub>4</sub> ↓q <sub>12</sub> | $\lambda$ Eq <sub>4</sub> ↓q <sub>12</sub> |
| q <sub>4</sub>  | -                                 | -                                 | -   | -  |
| q <sub>5</sub>  | 0Rq <sub>6</sub> ∅                | 1Rq <sub>9</sub> ∅                | -   | -  |
| q <sub>6</sub>  | 0Rq <sub>6</sub> ∅                | 1Rq <sub>6</sub> ∅                | $\delta$ Rq <sub>6</sub> ∅                | 0Lq <sub>7</sub> ∅                         |
| q <sub>7</sub>  | 0Lq <sub>7</sub> ∅                | 1Lq <sub>7</sub> ∅                | $\delta$ Eq <sub>8</sub> ∅                | -  |
| q <sub>8</sub>  | -                                 | -                                 | -   | -  |
| q <sub>9</sub>  | 0Rq <sub>9</sub> ∅                | 1Rq <sub>9</sub> ∅                | $\delta$ Rq <sub>9</sub> ∅                | 1Lq <sub>10</sub> ∅                        |
| q <sub>10</sub> | 0Lq <sub>10</sub> ∅               | 1Lq <sub>10</sub> ∅               | $\delta$ Eq <sub>11</sub> ∅               | -  |
| q <sub>11</sub> | -                                 | -                                 | -   | -  |
| q <sub>12</sub> | 0Lq <sub>13</sub> ∅               | 1Lq <sub>13</sub> ∅               | $\delta$ Lq <sub>13</sub> ∅               | $\lambda$ Lq <sub>13</sub> ∅               |
| q <sub>13</sub> | 0Lq <sub>14</sub> ∅               | 1Lq <sub>14</sub> ∅               | $\delta$ Lq <sub>14</sub> ∅               | $\lambda$ Lq <sub>14</sub> ∅               |
| q <sub>14</sub> | 0Lq <sub>5</sub> ∅                | 1Lq <sub>5</sub> ∅                | $\delta$ Lq <sub>5</sub> ∅                | $\lambda$ Lq <sub>5</sub> ∅                |

Рис. 1.1. Программа переворачивания строки для ЭлРМТ

Теперь сконструируем *эволюционную РМТ*. От ЭлРМТ она будет отличаться, прежде всего, расширенными правилами перехода, которые имеют вид

$$\delta = Q \times A \rightarrow Q \times A \times S \times P_{12} \cup \{-\},$$

где  $P_{12} = P_1 \cup P_2$ ,  $P_2 = \{\uparrow\#, \downarrow\#, \#\#\}$ . Считается, что сразу после исполнения команды  $\#\#$  (окончания разметки СЛАУ), ЭвРМТ в промежутке между состояниями *исполняет линейный оператор перехода*: изменяет содержимое некоторых из специальным образом помеченных групп ячеек на ленте (начало каждой очередной группы предварительно помечается командой  $\uparrow\#$ , конец — командой  $\downarrow\#$ ), решая локальную для данной машины СЛАУ из  $K$  уравнений, где  $K$  определяется в момент решения (перехода между состояниями). Каждая группа ячеек на ленте представляет собой или константу (коэффициент при переменной или свободный член)

или переменную. Как константы, так и переменные записываются на ленту либо в двоичной, либо в иной, заранее оговоренной форме. Предназначение группы определяется по позиционному принципу: считается, что первые  $K^2$  групп представляют коэффициенты (матрица коэффициентов развертывается построчно), следующие  $K$  групп — переменные, а последние  $K$  групп — свободные члены (также развертываются построчно). Легко убедиться, что если всего было размечено  $N$  групп, то по  $N$  можно однозначно восстановить  $K$  и, тем самым, определить местоположение всех основных элементов СЛАУ. В самом деле, учитывая, что  $N \geq 0$ , уравнение относительно  $K$ ,

$$K^2 + K + K = N,$$

имеет только один положительный или равный нулю корень

$$K = -1 + \sqrt{N+1}.$$

**Теорема о представимости сложных функций ЭВРМТ (ТКЛ1).** В процессе функционирования ЭВРМТ способна реализовать произвольную (сколь угодно сложную) функцию.

**Доказательство.** Пусть дана сложная функция  $f(x)$ , определенная на интервале  $[A; B]$ . Разобьем данный интервал на  $R$  сегментов равной ширины  $\frac{B-A}{R}$  таким образом, чтобы в каждом сегменте  $f(x)$  или имела разрыв или с достаточно высокой для расчетов точностью приближалась отрезком прямой. Ограничимся рассмотрением только тех сегментов, на которых  $f(x)$  не имеет разрыва (сегменты с разрывом численному расчету не подлежат изначально). Получаем кусочно-линейную аппроксимацию. Поскольку в каждом сегменте  $[a; b]$  переход к любой из его точек описывается линейным уравнением

$$f(x) \approx f(a) + \frac{x-a}{b-a}(f(b) - f(a)),$$

он может быть представлен линейным оператором перехода ЭВРМТ. Таким образом, задача сводится к поиску для заданного  $x$  соответствующего интервала  $[a; b]$ . Эта задача решается последовательным перебором всех возможных комбинаций  $a$  и  $b$ , из числа которых будет выбрана такая, что  $a \leq x$  и  $x \leq b$ . Перебор легко реали-

зуется МТ, расширением которой является ЭВРМТ. Операция сравнения вида  $m \leq n$  может быть реализована с помощью вычитания: если вычитание  $m$  из  $n$  не приводит к заему за старшим разрядом, то  $m \leq n$ . Операция вычитания достаточно тривиально реализуется МТ, а значит и ЭВРМТ. Доказано для случая функции одной переменной.

Для общего случая функции многих переменных доказывается аналогично.

**Следствие (СКЛ1).** Поскольку ИНС прямого распространения реализует сложную интерполирующую функцию многих переменных, ЭВРМТ способна вычислить отклик как произвольной такой ИНС в целом, так и любого из ее нейронов в частности.

**Следствие (СКЛ2).** Заметим, что обучение ИНС реализуется неким алгоритмом, включающим математические соотношения, которые могут быть сведены к кусочно-линейным аналогам (с помощью методов вычислительной математики) и решению неких СЛАУ. Поскольку ЭВРМТ является надмножеством МТ, она способна исполнить любой алгоритм, а с помощью линейного оператора перехода ЭВРМТ способна реализовать произвольные кусочно-линейные схемы и решить возникающие СЛАУ. Таким образом, ЭВРМТ способна реализовать обучение ИНС.

**Теорема о наблюдаемости ЭВРМТ на интервале выполнения оператора перехода (ТНБ1).** Содержимое групп ячеек-переменных в период исполнения оператора перехода (после вызова команды, содержащей ##, в промежутке между двумя состояниями ЭВРМТ) наблюдаемо и, при соблюдении некоторых допущений, изменяется по линейному закону.

**Доказательство.** Пусть интервал исполнения оператора перехода имеет длительность  $\Delta t = t_2 - t_1$ , где  $t_1$  — время начала исполнения оператора перехода, а  $t_2$  — время конца исполнения этого оператора. В начале исполнения некоторая произвольно выбранная группа ячеек-переменная  $H(t)$  имеет некоторое значение, интерпретируемое как число  $H_1$ . Введем допущение, что время решения СЛАУ, соответствующей оператору перехода,  $t_{\text{реш}} \ll \Delta t$ . Тогда можно считать, что ко-

нечное значение данной переменной  $H_2$  также известно в начале исполнения оператора перехода и мы можем сразу определить линейный закон изменения  $H(t)$  на интервале  $[t_1; t_2]$  с помощью линейной интерполяции:

$$H(t) = (1 - \alpha(t))H_1 + \alpha(t)H_2,$$

$$\alpha(t) = \frac{t - t_1}{t_2 - t_1}.$$

Доказано.

**Теорема об абстрактной предельной ОСМ с линейным оператором эволюции (ТП1-1).** Предельной абстрактной (атомарной) объектно-событийной моделью с линейным оператором перехода назовем соответствующую модель, включающую единственный узел-объект. Такая модель эквивалентна и, следовательно, равномощна ЭВРМТ.

**Доказательство.**

1. Как и объект, ЭВРМТ имеет линейный оператор перехода.
2. Данные, представляющие поля объекта и глобальные данные, могут храниться на ленте ЭВРМТ.
3. Эквивалентом календаря событий модели является план исполнения  $L$  с состояниями. Следовательно, событиям мы можем сопоставить специфические состояния ЭВРМТ. Тогда начальному событию соответствует начальное состояние  $P_0$ .
4. Условное планирование событий ОСМ представим командами включения соответствующих им состояний в план исполнения ЭВРМТ.
5. Эквивалентами методов обработки событий ОСМ будут фрагменты программы (подмножества  $\delta$ ) ЭВРМТ, начинающиеся с состояний, указанных в плане исполнения.

Эквивалентность доказана.

Сконструируем *параллельную РМТ*. От ЭВРМТ она отличается расширенными правилами перехода, имеющими вид:

$$\delta = Q \times A \rightarrow Q \times A \times S \times P_{123} \cup \{-\},$$

где  $P_{123} = P_1 \cup P_2 \cup P_3$ ,  $P_3 = \{\uparrow \infty, \downarrow \infty, \emptyset\}$ .

В отличие от ЭвРМТ, ПарРМТ имеет *динамически изменяемое множество головок*  $H$ , каждая из которых имеет *собственные* позицию и активное состояние. Головки работают параллельно и независимо друг от друга по программе  $\delta$ . Головка прекращает работу и удаляется из множества  $H$  при исполнении оператора вида  $q; a_j \rightarrow -$ . В начале работы ПарРМТ головка одна и ей соответствует начальное состояние  $p_0$ . *План работы*  $L$  расширен: хранит состояния  $q \in Q \cup \{\emptyset\}$ . Исполнение команды, содержащей  $\downarrow \infty$ , ставит в позицию  $L_{n+1}$  ( $n$  — число элементов плана до исполнения текущей команды) барьерное псевдосостояние  $\emptyset$ , которое выполняет роль *разделителя групп элементов плана*. Исполнение команды, содержащей  $\uparrow \infty$ , приводит к извлечению из плана всех элементов до барьера (который при этом также извлекается из плана) или до конца, если барьера нет. Тогда для каждого извлеченного элемента (этапа работ) порождается дополнительная головка (пополняется множество  $H$ ), находящаяся в соответствующем этому этапу состоянии.

Каждая головка работает с лентой независимо от других головок, при этом считается, что операции (чтение и запись) с одной и той же ячейкой ленты атомарны. Во избежание неоднозначностей атомарными также считаются команды из множества  $P_2$ :  $\uparrow \#, \downarrow \#, \#\#$ .

Если на определенном этапе работы ПарРМТ множество головок  $H$  оказывается пустым, то из плана  $L$  извлекается первый элемент, для которого порождается новая головка с соответствующим состоянием. Если же план пуст, ПарРМТ завершает работу.

**Теорема об абстрактной предельной параллельной ОСМ (ТПЗ-1).** Параллельная РМТ является простым вариантом абстрактной предельной параллельной ОСМ.

**Доказательство.** Верно по построению. Это минимальная РМТ, имеющая функции параллельного исполнения.

В качестве примера рассмотрим программу параллельной очистки ленты (рис. 1.2). Пусть на ленте находится участок, непрерывно заполненный некими символами « $a_1$ ».. « $a_n$ », прочее же пространство ленты заполнено символом « $\lambda$ », то есть алфавит ПарРМТ  $A = \{a_1, \dots, a_n, \lambda\}$ . Необходимо добиться того, чтобы символом « $\lambda$ » была заполнена вся лента. Пусть в начальном состоянии ( $p_0 = q_1$ ) начальная головка машины установлена на один из символов непрерывного участка. Программа начинается с того, что порождает два этапа плана для последующего запуска двух дополнительных параллельно работающих головок, одна из которых будет очищать ленту, двигаясь влево, а вторая — двигаясь вправо. Далее запускается исполнение плана в параллельном режиме (порождаются две вышеуказанные головки), а начальная головка завершает работу.

| Состояния | Символы алфавита            |     |                             |                           |
|-----------|-----------------------------|-----|-----------------------------|---------------------------|
|           | $a_1$                       | ... | $a_n$                       | $\lambda$                 |
| $q_1$     | $a_1 E q_2 \uparrow q_5$    | ... | $a_n E q_2 \uparrow q_5$    | -                         |
| $q_2$     | $a_1 E q_3 \uparrow q_6$    | ... | $a_n E q_3 \uparrow q_6$    | -                         |
| $q_3$     | $a_1 E q_4 \uparrow \infty$ | ... | $a_n E q_4 \uparrow \infty$ | -                         |
| $q_4$     | -                           | ... | -                           | -                         |
| $q_5$     | $\lambda L q_5 \emptyset$   | ... | $\lambda L q_5 \emptyset$   | -                         |
| $q_6$     | $a_1 R q_7 \emptyset$       | ... | $a_n R q_7 \emptyset$       | $\lambda R q_7 \emptyset$ |
| $q_7$     | $\lambda R q_7 \emptyset$   | ... | $\lambda R q_7 \emptyset$   | -                         |

Рис. 1.2. Программа параллельной очистки ленты для ПарРМТ

Выше мы рассмотрели три случая РМТ, которые последовательно выводились для построения абстрактной предельной параллельной ОСМ (ПарРМТ). Соответственно, эти случаи наиболее естественны для представления изолированной РМТ как одного объекта в составе ОСМ. Однако на практике также весьма интересны предельные случаи для более общего варианта ОСМ — ОСМ как самомодифицирующейся системы объектов. Система объектов, вероятно, тоже будет одним из вариантов РМТ (*системной РМТ* — *СРМТ*), которая добавляет, удаляет и

хранит на своей ленте входящие в нее объекты-РМТ, определяет порядок их запуска в соответствии с общей логикой интерпретации ОСМ, запускает их. Объекты, в таком случае, должны иметь возможность планировать оператор перехода со ссылками на внешние по отношению к ним элементы, принадлежащие иным объектам. Это будут варианты РМТ (*эволюционные глобального решения РМТ — ЭГРРМТ*) с возможностью глобального расчета для оператора эволюции. Здесь необходимо решить проблему составления эволюционного оператора в общем случае, где простая позиционная запись такого оператора становится громоздкой и неудобной. Как уже упоминалось выше, целесообразно прибегнуть к текстовой записи СЛАУ<sup>1</sup>, которая генерируется объектами-РМТ, входящими в систему. Видимо, речь может идти о *порождающей РМТ (ПорРМТ)*, которая также станет предельным случаем ОСМ, применяемой для порождения программ.

Итак, мы вводим понятие о следующих трех расширениях МТ:

а) *порождающей РМТ (ПорРМТ)*, в которой вводится возможность порождения символьной цепочки (для порождения текста программы или оператора перехода РМТ);

б) *эволюционной глобального решения РМТ (ЭГРРМТ)*, которая позволяет определять переменные и константы оператора перехода для группы РМТ;

в) *системной РМТ (СРМТ)*, позволяющей работать с системой РМТ.

Рассмотрим последовательно эти три разновидности РМТ.

Сконструируем *порождающую РМТ*. От ПарРМТ она отличается наличием порождаемой цепочки Т и расширенными правилами перехода, имеющими вид:

$$\delta = Q \times A \rightarrow Q \times A \times S \times P_{1-4} \cup \{-\},$$

где  $P_{1-4} = P_1 \cup P_2 \cup P_3 \cup P_4$ ,  $P_4 = (\{\uparrow'\} \times M(G_1)) \cup (\{\downarrow'\} \times M(G_1)) \cup \{\uparrow \varepsilon\}$ ,  $G_1$  — грамматика, определяющая правила порождения цепочек,  $M(G_1)$  — множество цепочек терминальных символов, выводимых по грамматике  $G_1$ . Изначально цепочка Т пуста. Команда, содержащая  $\uparrow\omega$ , совершает операцию конкатенации сле-

<sup>1</sup> Интересно заметить, что в таком случае возможно рассмотрение и еще более общего случая РМТ как машин с произвольным оператором перехода, если он может быть представлен в простой текстовой форме, которую РМТ умеет интерпретировать.

ва  $T = \omega T$ , а команда, содержащая  $\downarrow\omega$ , совершает операцию конкатенации справа  $T = T\omega$ . Исполнение команды, содержащей  $\uparrow\varepsilon$ , приводит к очистке цепочки  $T$ . Все команды работы с цепочками являются атомарными.

**Теорема о предельной порождающей ОСМ (ТП6).** Предельной порождающей ОСМ является ПорРМТ.

**Доказательство.** Верно по построению. Это минимальная РМТ, имеющая средства генерирования символьных цепочек.

Сконструируем *эволюционную РМТ глобального решения*. От порождающей РМТ она отличается, прежде всего, тем, что генерируемая символьная цепочка наделяется особым смыслом — это некоторая текстовая запись (например, в обычной алгебраической форме, то есть в синтаксисе, характерном для большинства математических систем) фрагмента глобального эволюционного оператора системы, состоящей из множества ЭГРРМТ. Можно считать, что порядок конкатенации фрагментов, сгенерированных отдельными ЭГРРМТ, не важен (если они содержат полностью законченные уравнения), поскольку порядок следования блоков уравнений, входящих в общую СЛАУ, не имеет значения.

ЭГРРМТ имеет расширенные правила перехода, имеющие вид:

$$\delta = Q \times A \rightarrow Q \times A \times S \times P_{1-5} \cup \{-\},$$

где  $P_{1-5} = P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5$ ,  $P_5 = (\{\uparrow\#\} \times M(G_2)) \cup (\{\downarrow\#\} \times M(G_2)) \cup \{\downarrow\#\varepsilon\}$ ,  $G_2$  — грамматика, определяющая правила порождения имен элементов (констант и переменных) общего эволюционного оператора,  $M(G_2)$  — множество имен (цепочек терминальных символов), выводимых по грамматике  $G_2$ .

Начало группы ячеек на ленте, соответствующих элементу (константе или переменной) с именем  $\psi$ , отмечается исполнением команды, содержащей  $\uparrow\#\psi$ . Тип элемента определяется при пометке конца соответствующей группы ячеек: команда, содержащая  $\downarrow\#\psi$ , помечает элемент как *переменную*, а команда, содержащая  $\downarrow\#\varepsilon$ , помечает элемент как *константу*. Очевидно, что эти три команды также должны быть атомарными. Если различные ЭГРРМТ пометили на своих лентах группы ячеек под одним именем, то в эволюционном операторе они счи-

таются одним элементом — если это переменная, то после выполнения оператора ее значение копируется на ленту каждой из определивших ее ЭГРРМТ.

Подчеркнем, что ЭГРРМТ не самостоятельна, а обязательно входит в состав некоей СРМТ. Это означает, что инициирование исполнения глобального эволюционного оператора должно производиться командой  $\#\#$ , выполняемой СРМТ, а не какой-либо из отдельных ЭГРРМТ, причем этот эволюционный переход произойдет только после полной остановки ЭГРРМТ, входящих в СРМТ. Очевидно, что СРМТ, видимо, будет хранить входящие в нее ЭГРРМТ на своей ленте, имея некоторую специфическую команду запуска ЭГРРМТ с определенным начальным состоянием, которое для ЭГРРМТ выполняет роль идентификатора произошедшего события (фрагмент программы ЭГРРМТ, стартующий с данного начального состояния, может рассматриваться как обработчик произошедшего события, по аналогии с идеологией ОСМ).

Вышесказанное вполне определяет форму и содержание *системной РМТ*. Алфавит СРМТ помимо простых символов содержит некоторое множество ЭГРРМТ — множество  $E$ . Пусть  $\tilde{Q}(e)$  — множество внутренних состояний машины  $e \in E$ . Тогда

$$Q_E = \bigcup_{e \in E} \tilde{Q}(e),$$

причем считается, что

$$\forall a \in E, \forall b \in E, a \neq b: \tilde{Q}(a) \cap \tilde{Q}(b) = \emptyset.$$

Правила перехода имеют вид:

$$\delta = Q \times A \rightarrow Q \times A \times S \times P_{1-6} \cup \{-\},$$

где  $P_{1-6} = \bigcup_{i=1}^6 P_i$ ,  $P_6 = \{\uparrow\uparrow\uparrow\} \times Q_E$ . Команда, содержащая  $\uparrow\uparrow\uparrow q$ , осуществляет син-

хронный запуск ЭГРРМТ, находящейся в текущей ячейке ленты СРМТ, с начальным состоянием  $q$ . Если же в текущей ячейке находится не ЭГРРМТ, а произвольный простой символ, то выполняется глобальный оператор эволюции (то есть происходит то же самое, как если бы СРМТ выполнила команду, содержащую  $\#\#$ ). Начальное положение (до первого запуска) головки запускаемой ЭГРРМТ огово-

ривается заранее, между запусками это положение сохраняется. Поскольку старт ЭГРРМТ выполняется синхронно, для обеспечения асинхронности, видимо, целесообразно перед запуском породить две головки СРМТ (формированием в плане двух состояний и запуске их в параллель), одна из которых, собственно, запускает машину с ленты и ожидает завершения ее функционирования, а вторая — продолжает работу.

**Теорема о реализуемости анализа и трансформации СРМТ (ТТР1).** Системная РМТ способна к анализу собственной структуры и ее трансформации.

**Доказательство.** Поскольку в структуру СРМТ входят отдельные ЭГРРМТ, хранимые на ее ленте на правах символов алфавита, и поскольку любое надмножество МТ способно к чтению символов с ленты и принятию на основании этой информации решения об удалении имеющихся символов и/или помещении новых символов, то СРМТ способна как к анализу, так и к трансформации своей структуры.

**Следствие.** Если СРМТ описывает нейронную сеть как множество нейронов-ЭГРРМТ, хранящихся на ее ленте, то СРМТ *способна к анализу и трансформации нейронной сети.*

До сих пор, говоря об абстрактных предельных ОСМ, мы придерживались того принципа, что это должны быть минимальные по численности входящих в них объектов системы — однообъектные. Однако представляет интерес и случай предельной многообъектной (составной) системы.

**Теорема о предельной абстрактной составной ОСМ (ТП7).** Предельной абстрактной многообъектной (составной) ОСМ является СРМТ. При этом объекты составной ОСМ представляются ЭГРРМТ, входящими в состав СРМТ, хранимыми на ее ленте, а связи между объектами закодированы последовательностями символов алфавита СРМТ, также хранимыми на ее ленте.

**Доказательство.** Составная ОСМ содержит сеть объектов, инициируемую в соответствии с динамически пополняемым календарем событий. При обработке каждого события объекты активизируются покаскадно, в соответствии с И-логикой активации каждого объекта по входам (то есть с логикой сетевого графика).

ка работ). При этом объекты обмениваются данными по связям и через общую базу данных «почтовый ящик». При этом возможно порождение ОСМ символьной цепочки и выполнение линейного оператора перехода между событиями.

В соответствии с вышесказанным, составная ОСМ может быть представлена СРМТ, поскольку содержит множество объектов-ЭГРРМТ, каждый из которых является надмножеством МТ, способных выполнить любое множество алгоритмов. При этом МТ эквивалентна абстрактной предельной последовательной ОСМ согласно теореме ТП1 [52], что позволяет говорить и о возможности работы *вложенных ОСМ*. Эти ЭГРРМТ имеют собственные ленты, что эквивалентно наличию полей у объекта составной ОСМ, и могут запускаться с указанием контекстного (для обработки текущего события) начального состояния (это эквивалентно вызову соответствующего метода обработки события составной ОСМ в объекте-ЭГРРМТ) и косвенно обмениваться данными путем исполнения СРМТ оператора эволюции. В самом деле, если нам необходимо передать данные из группы ячеек *a*, принадлежащей ленте машине *M1*, в группу ячеек *b* ленты машины *M2*, то необходимо, используя команды ЭГРРМТ, маркировать ячейки *a* как константу, а ячейки *b* как переменную, после чего породить символьную цепочку вида «*a=b*», представляющую собой элемент соответствующего линейного оператора перехода СРМТ, и, пользуясь командой СРМТ, содержащей ##, исполнить этот оператор. В результате осуществляется передача данных из области *a* машины *M1* в область *b* машины *M2*. Аналогично моделируется обмен данными в рамках парадигмы «почтовый ящик», с той разницей, что данные передаются между группой ячеек ленты самой СРМТ (собственно «почтовым ящиком») и группой ячеек ленты одной из ЭГРРМТ, входящих в СРМТ.

Логика работы составной ОСМ в рамках календаря событий вполне соответствует логике работе СРМТ в рамках плана исполнения *L*, с тем непринципиальным ограничением, что включать новые события можно только в начало или конец плана. Механизм каскадной активации ЭГРРМТ по принципу сетевого графика работ является достаточно простым алгоритмом (при условии что на ленте СРМТ хранится закодированная символами ее алфавита *A* полная информация о

связях между объектами-ЭГРРМТ), который может быть представлен МТ, а значит и СРМТ, которая является надмножеством МТ.

В процессе работы СРМТ может породить символьную цепочку, поскольку является надмножеством ПорРМТ, а также способна выполнять глобальный линейный оператор перехода между двумя произвольно взятыми состояниями, которые, при необходимости, могут являться границами раздела событий составной ОСМ (назовем их *метасобытиями*), каждому из которых соответствует исполнение множества внутренних состояний-событий. Наступлению каждого метасобытия можно сопоставить (программным путем, используя команды, унаследованные из ЭлРМТ) переход СРМТ к очередному состоянию, взятому из плана исполнения (из календаря событий составной ОСМ в терминах доказываемой теоремы), хотя не следует исключать также вариант, когда метасобытие такому переходу не соответствует.

Таким образом, все вышеперечисленные требования к форме и содержанию составной ОСМ выполнены. Доказано.

**Теорема о векторном и конвейерном видах параллелизма (ТВК1-1).** Как векторный, так и конвейерный виды параллелизма по объектам-ЭГРРМТ могут быть реализованы в рамках формализма СРМТ.

**Доказательство.** Векторный параллелизм по  $K$  элементам может быть обеспечен параллельным запуском  $K$  головок, используя команды ПарРМТ (подмножества СРМТ). Каждая из этих головок может запустить ЭГРРМТ в начальном состоянии  $q$ , используя команду, содержащую  $\uparrow\uparrow\uparrow q$ . Таким образом получаем вектор из  $K$  параллельно работающих ЭГРРМТ.

Конвейерный параллелизм по  $K$  элементам подразумевает работу  $K$  объектов-стадий в параллельном режиме с последовательной передачей данных между стадиями. Следовательно, достаточно запустить вектор из  $K$  параллельных ЭГРРМТ и организовать передачу данных между ними с помощью глобального линейного оператора перехода (эволюции) так, как это указано в доказательстве теоремы ТП7. Поскольку такой оператор будет выполнен только *после остановки* всех ЭГРРМТ, данная схема обеспечивает исполнение только одного такта работы

конвейера. Следовательно, для полноценной реализации конвейерного параллелизма ее необходимо заключить в элементарный цикл, используя базовые команды МТ.

Доказано.

### 1.1.2. Представимость различных формализмов моделирования

#### Утверждение о представимости формализмов моделирования (УПФ1).

Любой формализм моделирования, сводимый к некоторому алгоритму исполнения множества последовательных и/или параллельных процессов, выполняющих одну или множество итераций с решением одной или нескольких СЛАУ, может быть реализован СРМТ.

**Доказательство.** Возможность исполнения последовательных процессов очевидна по построению МТ (подмножества СРМТ). Возможность исполнения параллельных процессов следует из ТВК1-1 (векторный и конвейерный параллелизм), а также из определения ПарРМТ (параллелизм «портфеля задач») — подмножества СРМТ. Любой из этих процессов включает множество команд ЭвРМТ, являющихся подмножеством ЭГРРМТ и СРМТ, поэтому может быть итеративным, причем на любой из этих итераций может быть произведено произвольное множество исполнений линейного оператора перехода, подразумевающего решение СЛАУ. Доказано.

**Следствие.** Модели, базирующиеся на аппаратах нейронных сетей прямого распространения, дискретных марковских цепей (к которым сводятся также марковские системы массового обслуживания), дифференциальных уравнений в частных производных, обыкновенных дифференциальных уравнений и ряде других, могут быть представлены в рамках формализма СРМТ. Прокомментируем этот тезис:

1. **Нейронные сети прямого распространения.** Рассмотрены ранее, см. следствия СКЛ1 и СКЛ2, а также следствие из ТТР1

**2. Дискретные марковские цепи (в том числе марковские СМО).** Здесь функционирование цепи описывается итерационной схемой, на каждой итерации которой применяются следующие линейные соотношения:

$$P(i+1) = P(i)J,$$

где  $P(i)$  — вектор вероятностей нахождения системы в некотором множестве состояний в  $i$ -й момент времени,  $J$  — матрица вероятностей перехода. Данные соотношения представляют собой СЛАУ. Таким образом, условия УПФ1 выполнены.

**3. Дифференциальные уравнения в частных производных.** Здесь для переменных-полей, входящих в уравнения, вводится дискретизирующая расчетная сетка, таким образом, вводится множество новых локальных переменных — значений полей в узлах расчетной сетки. Дальнейший расчет для нестационарных задач сводится ко вводу дополнительной дискретизации по времени и организации итерационной схемы, описывающей эволюцию значений в узлах во времени. При этом, в зависимости от разностной схемы, возникают системы линейных или нелинейных уравнений [26, 35, 56, 79, 82, 94], причем решение вторых (например, методом Ньютона или Левенберга-Марквардта или сопряженных градиентов), в свою очередь, сводится к итерационной схеме, на каждой итерации которой также решается СЛАУ. Если же расчет ведется для стационарной задачи, то он может либо сводиться к решению эквивалентной нестационарной задачи до установления, либо к непосредственному решению возникающей системы линейных или нелинейных уравнений [26, 35, 56, 79, 82, 94] относительно конечных значений полей в узлах. Таким образом, в любом случае процесс моделирования в частных производных сводится к более или менее сложной итерационной схеме с решением неких конечных СЛАУ. Этот расчет может вестись как в последовательном, так и в параллельном режиме. Таким образом, условия УПФ1 выполнены.

**4. Обыкновенные дифференциальные уравнения.** Здесь используются принципы решения, схожие с теми, которые привлекаются для уравнений в частных производных, с той разницей, что дискретизация по пространству отсутствует. Поэтому решение таких уравнений в конечном итоге также сводится к итерационным схемам с решением возникающих СЛАУ. Условия УПФ1 выполнены.

**Замечание о кусочно-линейных агрегатах Бусленко.** Модель в данном формализме представляют собой граф с узлами-агрегатами. Модель функционирует в соответствии с динамически расширяемым календарем событий, причем между событиями состояние агрегатов изменяется по некоторым линейным законам, а при обработке событий агрегаты могут активизировать друг друга, выдавая на соответствующие межагрегатные связи некоторую новую информацию. Данная схема может быть реализована СРМТ, причем ЭГРРМТ будут выступать в качестве агрегатов. План исполнения СРМТ будет исполнять роль календаря событий, причем существующее ограничение о том, что состояние может быть помещено только в начало или конец плана должно быть учтено логикой СРМТ при формировании плана-календаря. Логика и порядок активизации агрегатов-ЭГРРМТ определяются достаточно простыми алгоритмами, которые реализуемы произвольной МТ, в том числе СРМТ. Агрегаты могут работать как последовательно, так и параллельно, что вполне реализуемо СРМТ. Передача данных между агрегатами может быть определена применением линейного оператора перехода СРМТ, им же могут быть определены законы линейного изменения состояния агрегатов в промежутках между обработкой событий.

## **1.2. Работа с ИНС прямого распространения в системе, описываемой в пределе РМТ**

Система порождения программ PGEN++ ориентирована на работу с классическими неопредельными ОСМ, которые являются расширениями ЭГРРМТ (порождающие классы) и СРМТ (общая модель, дедуктивные решающие классы и вложенные ОСМ). Для описания элементов ИНС использовались преимущественно расширения ЭГРРМТ, тем не менее, отметим, что возможно и применение расширений СРМТ, например, для построения готовых нейронных сетей фиксированной структуры. Был разработан ряд порождающих классов для описания элементов ИНС и связанной с ней инфраструктуры: **Источник данных, Вход, Выход, Ней-**

рон, Учитель. На рис. 1.3 показан пример построенной модели, описывающей ИНС структуры  $4 \times 2 \times 1$ .

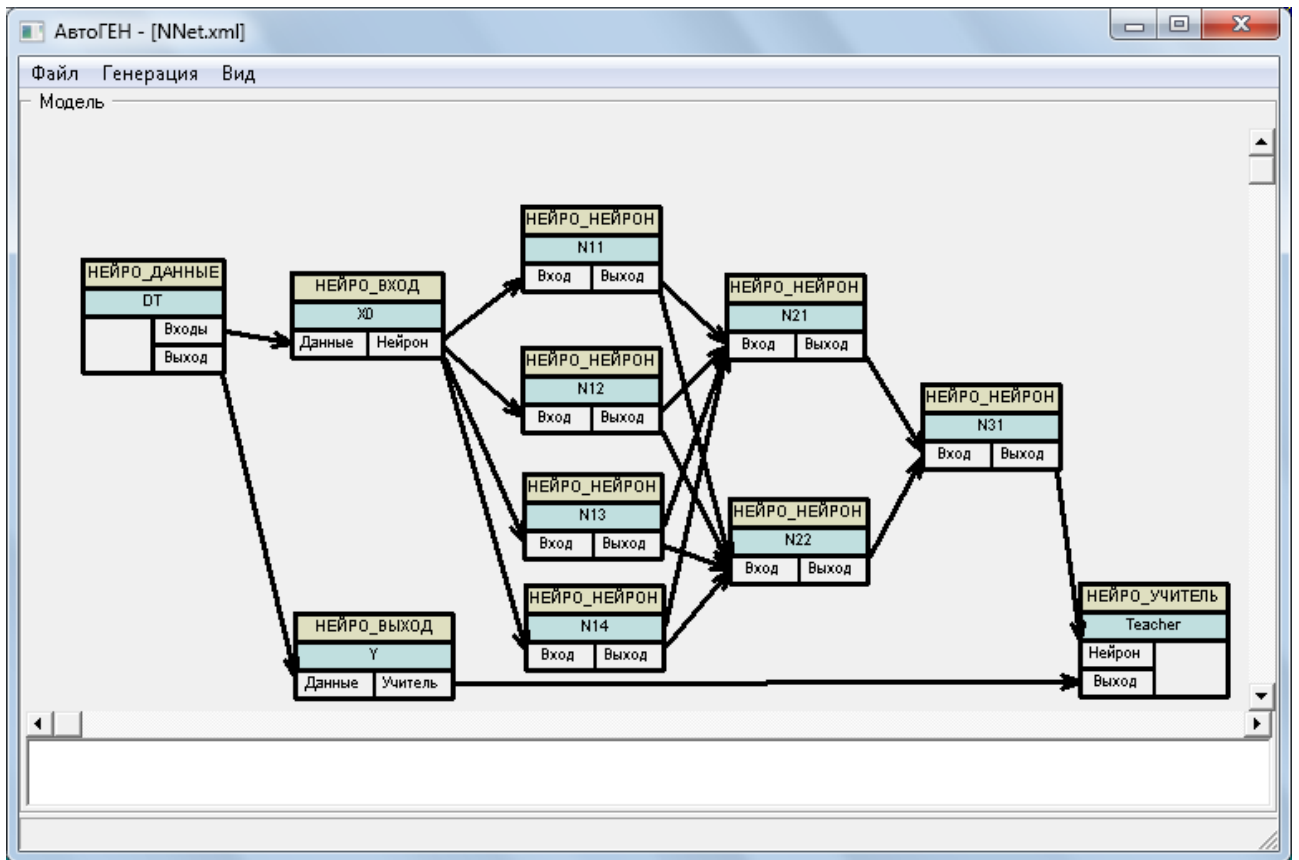


Рис. 1.3. Пример построенной ИНС в системе PGEN++

В процессе функционирования объекты данных классов порождают глобальный эволюционный оператор, представляющий собой набор соотношений для получения текущего отклика сети и обратного распространения ошибки. В настоящее время в системе PGEN++ эволюционный оператор представляет собой набор выражений на языке PНР, которые либо напрямую вычисляют внутренние данные объектов, либо формируют систему уравнений в матричной форме и вызывают соответствующие функции решения. Такой подход является, видимо, достаточно универсальным и вполне соответствует концепциям ЭвРМТ и ЭГРРМТ. На рис. 1.4 показан пример фрагмента сгенерированного системой для ИНС эволюционного оператора.

```

Трансляция
-----
финальная лента системы (финальный оператор эволюции)
-----

global $nPair;
$nPair = 374;
global $alpha;
$alpha = 0.2;
global $nu;
$nu = 0.05;
global $err;
global $DATA;
global $ROWS;
global $COLS;
$X0_Out = $DATA[$nPair][0];
$Y_Out = $DATA[$nPair][1];
global $N11_B;
global $N11_W;
global $N11_DB;
global $N11_DW;
$N11_Y = $N11_B + $X0_Out*$N11_W[0];
$N11_Out = 1.0/(1.0+exp(-$N11_Y));
$N11_delta = 0.0;
global $N12_B;
global $N12_W;
global $N12_DB;
global $N12_DW;
$N12_Y = $N12_B + $X0_Out*$N12_W[0];
$N12_Out = 1.0/(1.0+exp(-$N12_Y));
$N12_delta = 0.0;

Закреть Копировать Сохранить Выполнить Обучить

```

Рис. 1.4. Фрагмент сгенерированного системой эволюционного оператора

Необходимо заметить, что представленный выше на рисунке оператор не является полностью линейным, поскольку содержит вызовы нелинейной функции расчета экспоненты. Это вполне допускается в PGEN++ для удобства работы, хотя, несомненно, оригинальная PMT, взятая в чистом виде, в таком случае потребовала бы разбиение оператора на несколько более простых операторов, некоторые из которых, в сочетании с соответствующей программой PMT, занимались бы расчетом экспонент, сведенных к сериям линейных операций. В данном случае мы просто воспользовались стандартными возможностями, предоставленными языком PHP.

Сгенерированный для ИНС эволюционный оператор итерационно (каждой итерации соответствует переход между двумя последовательными событиями

обучения) применяется к внутренним данным объектов требуемое количество раз, в результате чего получаем обученную нейронную сеть. Таким образом, были *автоматизированы построение и применение ИНС*.

Также был разработан решающий класс (**Сеть**), который по заданному формальному описанию ИНС строит (с помощью генерирующих скриптов на языке GNU Prolog) соответствующую модель этой ИНС из элементарных порождающих объектов, относящихся к указанным выше классам. Таким образом, была *автоматизирована генерация ИНС*.

К системе был подключен параллельный модуль, осуществляющий *анализ и упрощение ИНС* (соответствующие основные принципы и алгоритмы будут описаны в настоящей работе далее) до элементарных алгебраических моделей. Модуль генерирует множество вариантов упрощения и выбирает из них три наиболее компактных, которые предоставляет системе PGEN++ на выходе. Пример сгенерированных системой результатов показан на рис. 1.5.

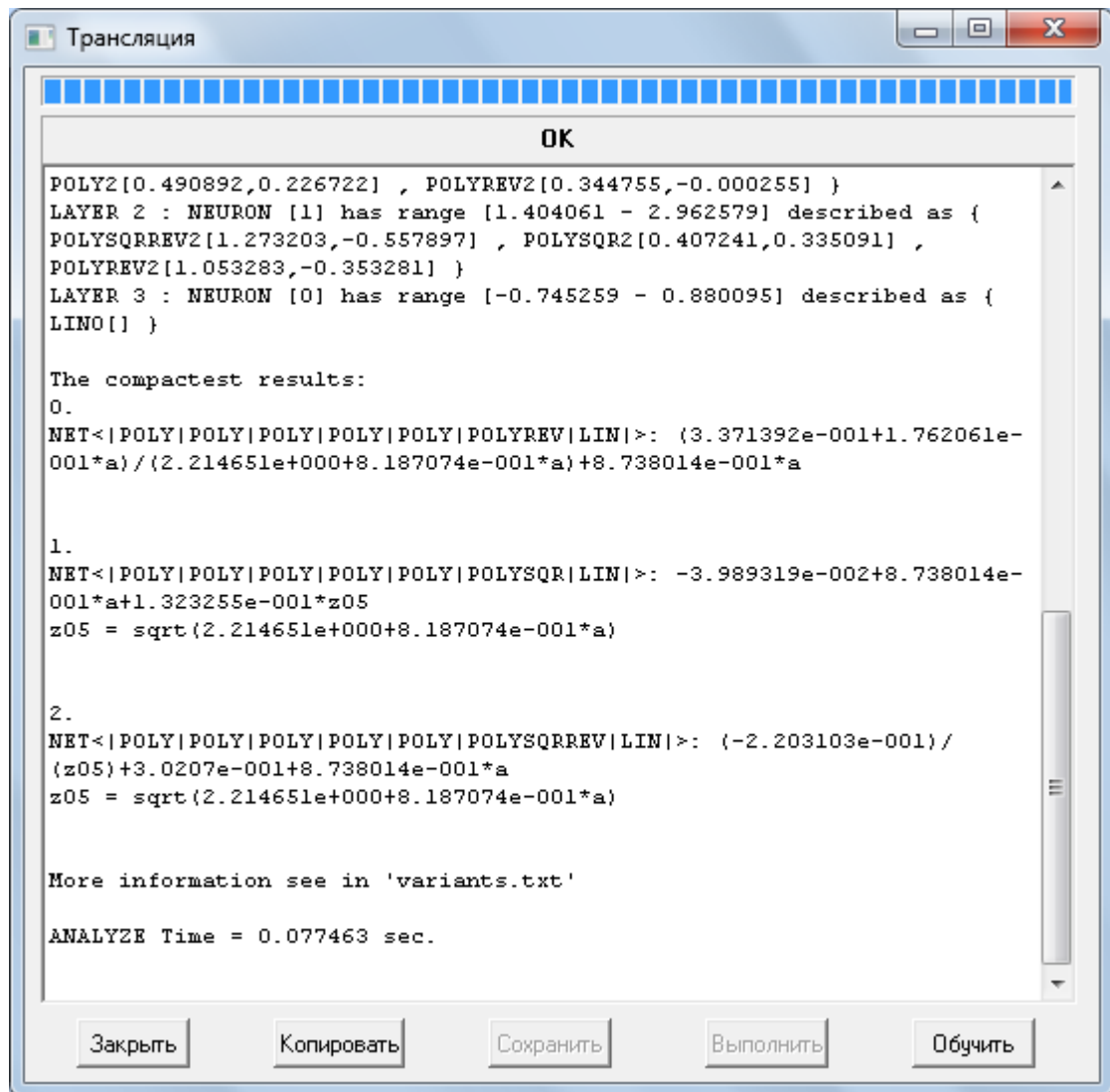


Рис. 1.5. Пример со сгенерированными для ИНС упрощенными алгебраическими моделями

Таким образом, удалось добиться того, что генерация/построение, обучение, анализ и редукция ИНС реализуются в унифицированной среде с единым модельным формализмом, что обеспечивает упрощение и, следовательно, повышение эффективности выполнения данных задач для исследователя. Разработанная иерархия классов, согласно идеологии ОСМ, является расширяемой путем наследования, что и обеспечивает открытость и расширяемость данной системы в плане работы с ИНС.

### 1.3. Построение интерполяционных моделей в системе, описываемой в пределе PMT

Рассмотрим проблему построения интерполяционных моделей общего вида в системе PGEN++. При этом будем исходить из принципа возможности визуального построения и редактирования моделей. В качестве идеологической основы для интерполяции выберем метод группового учета аргументов (МГУА, см., например, [23, 61]), в который будут внесены некоторые дополнительные возможности. Сразу же отметим, что нам не удалось найти прецедентов такого рода систем визуального моделирования на базе МГУА. Известные системы (ППП МГУА, АСТРИД, GMDH Shell, FAKE GAME и другие, см., например, обзор, данный в работе [40]) не содержат графического интерфейса проектирования МГУА-подобных моделей. При этом нельзя отрицать, что интерфейс такого рода должен повысить качество построения интерполяционных моделей, поскольку всегда оставляет исследователю возможность вручную дополнить модель, введя минимум необходимых блоков (системы автоматической генерации интерполирующих моделей могут выдавать несколько избыточные результаты). Поэтому данное направление, несомненно, актуально. Некоторым аналогом разрабатываемой технологии, возможно, могла бы считаться технология среды TensorFlow<sup>1</sup>, позволяющая визуально проектировать обучаемые модели данных, однако данная среда представляет весьма низкоуровневый (и поэтому не вполне удобный) математический интерфейс и ориентирована все же преимущественно на плодотворную работу с ИНС вне рамок МГУА.

---

<sup>1</sup> Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.— <http://download.tensorflow.org/paper/whitepaper2015.pdf>

Будем априорно исходить из идеи, что достаточно совершенная интерполяционная модель может являться комбинированной моделью, осуществляющей аддитивное, мультипликативное или иное агрегирование частных подмоделей, описываемых строго определенными формализмами. Кроме того, вполне уместен конкурентный подход, когда одновременно проектируется целый набор альтернативных (возможно комбинированных) моделей, из которых выбирается модель, дающая наилучшее приближение. На том же принципе может основываться и процедура выбора подмодели, описывающей тренд данных. Вышесказанное позволяет говорить о наличии элементов планирования эксперимента по построению интерполяционных моделей.

В качестве частных формализмов для подмоделей могут рассматриваться: а) полиномы, определяемые или непосредственно по принципам МГУА или, в частном случае, исходя из принципов метода наименьших квадратов (МНК), б) произвольные функции от одного или двух аргументов, в) нейронные сети<sup>1</sup>. Все эти формализмы вполне описываемы РМТ, а следовательно и их непределённым случаем — ОСМ, являющимся базовым высокоуровневым формализмом системы PGEN++. Логично предположить, что для каждого из частных формализмов целесообразно предусмотреть собственный класс (тип) объекта:

а) **Унарная функция (UnaryFunction)** общего вида  $y = f(x)$ ;

б) **Унарный полином (UnaryPoly, наследник UnaryFunction)** вида 
$$y(x) = \sum_i a_i x^i;$$

в) **Унарная нейронная сеть (UnaryNetwork, наследник UnaryFunction)** вида  $y = \text{NET}(x)$ ;

г) **Многовходовая сеть (MultiNetwork, наследник UnaryNetwork)** вида  $y = \text{NET}(\bar{X})$ ;

---

<sup>1</sup> Необходимо пояснить, что, хотя среда моделирования PGEN++ и содержит принципиальную возможность описания ИНС понейронно, базируясь на введенном нами ранее аппарате РМТ, здесь такой необходимости нет. Более того, исходя из необходимости максимально повысить производительность, отдельные нейронные сети при построении интерполяционных моделей (описанным здесь способом) моделируются неделимыми объектами.

д) **Бинарная функция (BinaryFunction)** общего вида  $y = f(x_1, x_2)$ .

План эксперимента по поиску интерполяционной модели (в частном случае план представляет собой схему единственного интерполятора) согласно идеологии неопределенных ОСМ представляет собой сеть. В таком случае логично в качестве входов сети определить объекты некоторого класса **Значение (Value)**, которые будут представлять входы модели  $X$ , а выходами сети, соответственно, будут объекты класса **Назначение (Dest)**, представляющие выходы модели  $Y$ . Между входами и выходами сети будут располагаться объекты вышеуказанных и, возможно, некоторых иных классов, соединения между которыми будут определять, соответственно, значения выходов каких блоков будут подаваться на входы других блоков. Таким образом мы и определим схему интерполятора или, в более общем случае, схему эксперимента по подбору интерполятора.

Предполагается, что объект класса **Назначение** в общем случае будет выполнять функции выбора наилучшего интерполятора, если он является конечным для  $K$  ветвей плана (нескольких интерполяторов  $INTERP_j$ ,  $j = \overline{1, K}$ ). Выбор наилучшего интерполятора с номером  $f$  осуществляется по критерию минимума суммарной квадратичной ошибки:

$$f = \arg \min_j \left( \sum_{p=1}^N (Y_p - INTERP_j(\bar{X}_p))^2 \right),$$

где  $N$  — количество исходных точек  $(\bar{X}_p, Y_p)$ , по которым строится интерполяционная модель.

Следует заметить, что сказанного выше недостаточно, чтобы доопределить процесс работы по схеме МГУА (не хватает группировки функций по слоям и процедуры отбора лучших функций), а также для выбора наилучшего частного интерполятора для описания тренда при агрегировании нескольких частных подмоделей. Пусть, в целом, процесс поиска общего интерполятора протекает по схеме МГУА: каждый блок-интерполятор ставит себе целью наилучшим образом описать набор данных  $(\bar{X}', Y)$ , где  $\bar{X}'$  — вектор значений, поступивших на вход блока, рассчитанных предыдущими блоками. При этом блок вычисляет значения

$Y'$ , которые, соответственно, будут являться входными для последующих блоков. *Первым важным отличием* такого подхода от классического МГУА является наличие среди блоков не только элементарных полиномиальных функций, но и таких крупных блоков как нейронные сети. Для решения же задачи в постановке классического МГУА необходимо ввести еще несколько классов объектов:

а) **Фильтр (Filter)** — объект-селекционер, на вход которого поступают вектора данных  $\overline{X}'^k$  с  $K$  ветвей ( $k = \overline{1, K}$ ): выбирается  $M$  векторов, наиболее близких к  $Y$  по принципу минимизации суммарной квадратичной ошибки

$$Q_k = \sum_{p=1}^N \left( Y_p - \overline{X}'_p^k \right)^2, \text{ где } M \text{ — количество выходных связей объекта } (M \leq K, \text{ на}$$

каждую выходную связь подается вектор  $\overline{X}'^m$ , где  $m$  — номер одного из отобранных входных векторов);

б) **Разветвитель (Conn)** — объект, работающий совместно с фильтром, транслирующий значение своего входа на каждый из своих выходов. Данный класс является вспомогательным и решает очевидную проблему: количество входных связей  $S$  очередного слоя МГУА может существенно превышать количество  $M$  отобранных объектом класса **Фильтр** в предыдущем слое интерполяторов за счет того, что выход любого из отобранных интерполяторов  $T$  может являться входом для *нескольких* интерполяторов следующего слоя. В этом случае и ставятся разветвители от каждого из выходов фильтра ко всем соответствующим входам интерполяторов следующего слоя (обычно количество разветвителей, таким образом, равняется  $M$ , а суммарное количество их выходов равняется  $S$ ).

Необходимо особо упомянуть о том, что план эксперимента по идеологии МГУА должен быть наращиваемым, поскольку при классическом подходе количество слоев МГУА увеличивается до тех пор, пока снижается ошибка интерполяции входных данных. На практике такое наращивание выполняется либо исследователем вручную, путем простого ввода дополнительных блоков в план эксперимента, либо могут быть определены дедуктивные блоки, которые на каждом очередном этапе эксперимента будут самостоятельно усложнять исходную структуру

плана, вводя в него дополнительные блоки, не являющиеся дедуктивными и описывающие очередной слой. В этом случае задействуется *машина вывода ОСМ*.

Теперь обсудим *задачу выбора наилучшего частного интерполятора для описания тренда при агрегировании нескольких частных подмоделей*. Пусть она решается агрегирующими элементами, в качестве которых, видимо, будут выступать бинарные функции  $y = f(x_1, x_2)$ , при условии, что для них *определены обратные функции* расчета значения входа по выходу и иному входу, то есть функции

$$x_1 = g_1(y, x_2),$$

$$x_2 = g_2(y, x_1),$$

или, в частном случае, хотя бы одна из них.

В самом деле, такая функция имеет два входа, значения на которых определяются двумя подмножествами интерполяторов  $V_1$  и  $V_2$ , таких, что  $V_1, V_2 \subseteq V$ , где  $V$  — множество всех объектов плана эксперимента по построению интерполятора. К моменту активизации бинарной функции входные интерполяторы уже рассчитали свои выходные значения  $\bar{X}'_1$  и  $\bar{X}'_2$ . Предлагается *выбрать и зафиксировать* тот  $r$ -й вход, значение на котором ближе к  $Y$  по уже неоднократно приводившемуся выше критерию минимизации суммарной квадратичной ошибки, считая его трендовым. Тогда остается пересчитать частный интерполятор на оставшемся входе с номером  $(3 - r)$ . Для этого, следуя идеологии непределельных ОСМ, для блоков подмножества  $V_{3-r} \cap \bar{V}_r$  определяется подписка на новое, следующее событие с целью адаптировать интерполятор для приближения остаточных (не трендовых) данных  $\tilde{Y} = g_{3-r}(Y, x_r)$  вместо  $Y$ . Это достаточно естественное для функционирования ОСМ решение, включающее многократный пересчет параметров объектов при обработке реакции на различные события при наличии индивидуальных целей для этих объектов, которые обычно планируются помещением соответствующих записей в общую базу данных (общий кортеж) «почтовый ящик».

Данное решение существенно повышает гибкость построения интерполяторов и, вероятно, качество приближения (особенно при наличии ярко выраженных

трендов). Такого рода схема отсутствует в МГУА я является *ключевой новой особенностью* предлагаемого в данном пункте подхода к планированию эксперимента по построению комбинированных интерполяторов.

Проиллюстрируем все вышесказанное на примере решения двух задач построения интерполяционных моделей.

### 1.3.1. Построение модели коэффициента преломления воды в инфракрасном дальнем диапазоне

Воспользуемся данными о величине коэффициента преломления воды в зависимости от длины волны из работы [95]. Это достаточно нетривиальная зависимость, поэтому будем рассматривать три варианта интерполяционной модели:

а) полином 16-го порядка  $y_1(x) = \sum_{k=0}^{16} a_k x^k$ ;

б) трехслойная ИНС прямого распространения структуры  $12 \times 7 \times 1$  нейронов  $y_2(x) = \text{NET}[12,7,1](x)$ ;

в) произведение полинома 7-го порядка на результат работы трехслойной ИНС прямого распространения структуры  $10 \times 5 \times 1$  нейронов

$$y_3(x) = \text{poly}(x) \cdot \text{net}(x);$$

$$\text{poly}(x) = \sum_{k=0}^7 b_k x^k;$$

$$\text{net}(x) = \text{NET}[10,5,1](x).$$

Так мы проиллюстрируем работу при наличии *всех новых существенных особенностей* предложенной технологии планирования эксперимента: а) в качестве блоков используются не только полиномы, но и ИНС, б) бинарная функция умножения будет выбирать вход с лучшим описанием тренда, а интерполятор на втором входе будет обучаться по остаточному принципу. Был составлен план эксперимента, включающий три вышеуказанные модели в качестве трех альтерна-

тивных ветвей между блоками классов **Значение** и **Назначение**. План эксперимента был введен в систему PGEN++ (показан на рис. 1.6).

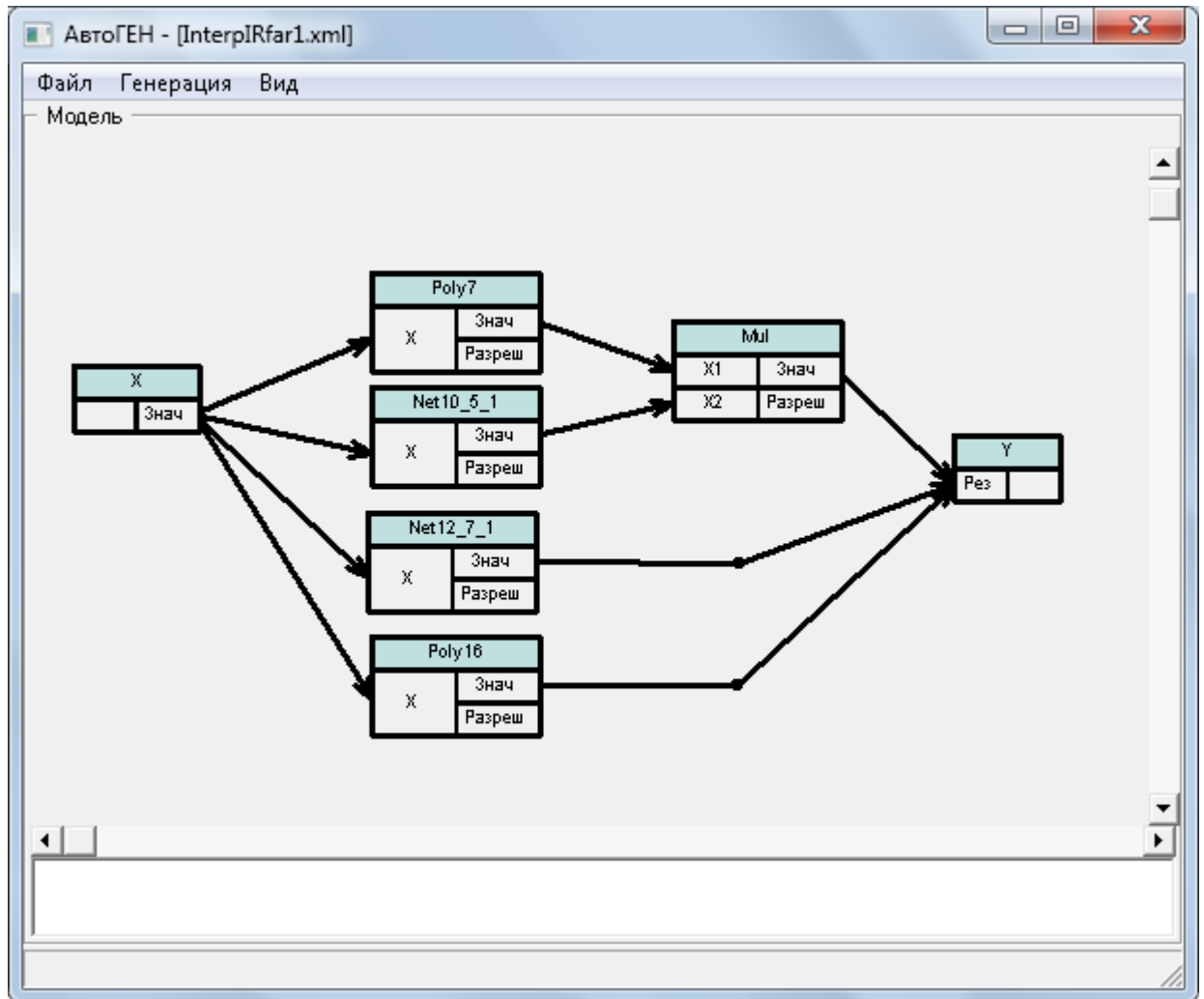


Рис. 1.6. План эксперимента по построению модели коэффициента преломления воды в системе PGEN++

Вышеуказанная модель была проинтерпретирована. В ходе работы выяснилось, что наилучшие показатели дает третья комбинированная модель (что, кстати, дополнительно подтверждает ценность таких моделей на практике), которая на языке М-скриптов записывается следующим образом:

```
Net10_5_1 = inline('sim(getfield(load(
'c:\work\Net10_5_1.mat'), 'BestK'), X)', 'X');
Poly7 = inline('polyval([-3.759E-012, 1.584E-009, -2.681E-
007, 2.323E-005, -0.0011, 0.0261, -0.276, 2.188], X)', 'X');
Mul = inline('X1.*X2', 'X1', 'X2');
```

$Y = \text{Mul}(\text{Poly7}(X), \text{Net10\_5\_1}(X));$

В данной модели Net10\_5\_1.mat — файл с данными обученной нейронной сети, соответствующей блоку **Net10\_5\_1** модели. На следующем рис. 1.7 приведены графики табличного и интерполированного значений коэффициента преломления.

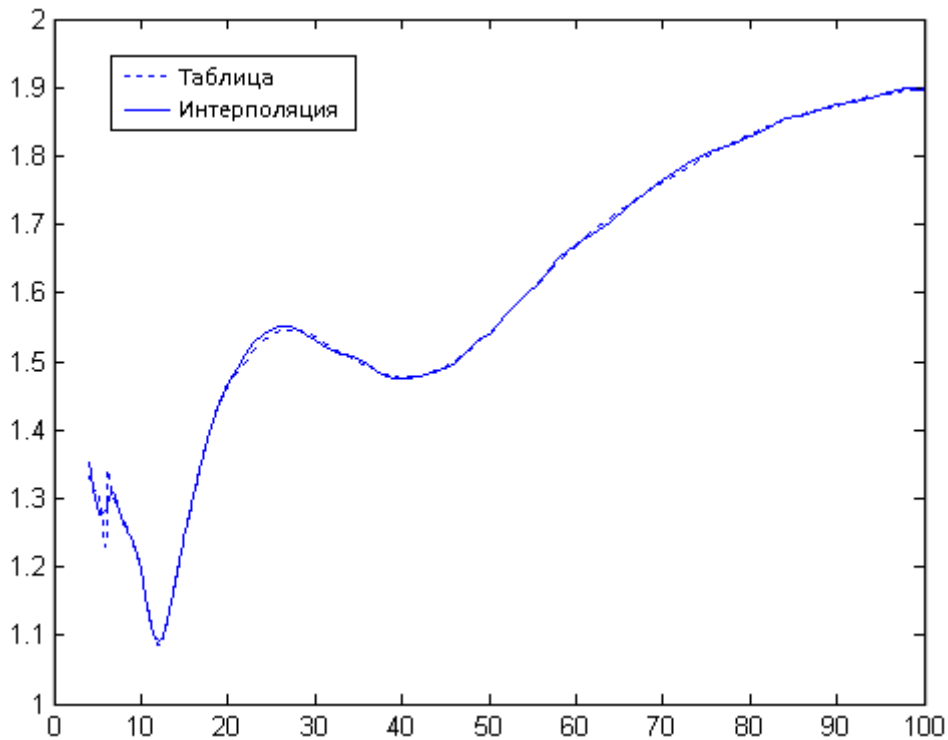


Рис. 1.7. Графики табличного и интерполированного коэффициентов преломления

Судя по приведенным выше графикам, удалось получить достаточно хорошее качество интерполяции. Таким образом, мы проиллюстрировали работу при наличии *всех новых существенных особенностей* предложенной технологии планирования эксперимента

### 1.3.2. Построение модели скорости выполнения фрагмента программы в метаслое

Понятие метаслоя программы введено нами в работах [47, 52]. Это надпрограммный слой, имеющий специальный интерфейс взаимодействия с программой, включающий специальные алгоритмы для моделирования логики программы, ее

данных (внутренних переменных) и функциональных характеристик, к которым, в частности, относится время исполнения ее фрагментов, зависящее как от логики программы, так и от значений ее внутренних переменных. В последнем случае в метаслое широко используется интерполяция полиномами, в том числе с применением МГУА. В данном пункте мы проиллюстрируем данный подход, построив приближенную модель времени исполнения программы с помощью МГУА в системе PGEN++, задействовав упоминавшиеся ранее блоки.

Был взят небольшой фрагмент программы на C++, содержащий несколько вложенных циклов по внутренним переменным  $\_N, j, p, s$ . Внутренний алгоритм, время исполнения которого нас интересует, изображен работой функции задержки хода исполнения, параметризованной сложным выражением от внутренних переменных. Данные о значениях внутренних переменных и времени исполнения сохраняются в файле.

```
FILE * FF = fopen("mgua_meta.dat", "w+t");
for (int _N = 1; _N<6; _N++) {
    for (int j=1; j<_N; j++) {
        for (int p=1; p<_N; p++) {
            for (int s=1; s<_N; s++) {
                double timer = milliseconds();
                Sleep((0.017*p*j+0.01*j*s+0.00005*(j+2)*p*s-0.001
*j)*100);
                timer = milliseconds() - timer;
                fprintf(FF,"%i %i %i %i %lf\n", _N, j, p, s,
timer);
            }
        }
    }
}
fclose(FF);
```

Было сделано предположение о том, что приближенная модель времени исполнения  $y = t(\_N, j, p, s)$  может быть построена с применением двух слоев (рядов) МГУА, в каждом из которых используются элементарные интерполяторы вида

$$y(x_1, x_2) = k_0 x_1 + k_1 x_2 + k_2 x_1 x_2.$$

Был составлен соответствующий план эксперимента, включающий единственную ветвь между блоками классов **Значение** (четыре штуки —  $_N, j, p, s$ ) и **Назначение**. Первый слой содержит шесть элементарных интерполяторов (класса **Бинарная функция**), затем включен объект класса **Фильтр**, выбирающий три лучших результата. Далее следуют три разветвителя, выходы которых идут на второй слой из трех интерполяторов, таких же, что и в первом слое. Далее идет еще один **Фильтр** и объект **Назначение**. План эксперимента был введен в систему PGEN++ (показан на рис. 1.8).

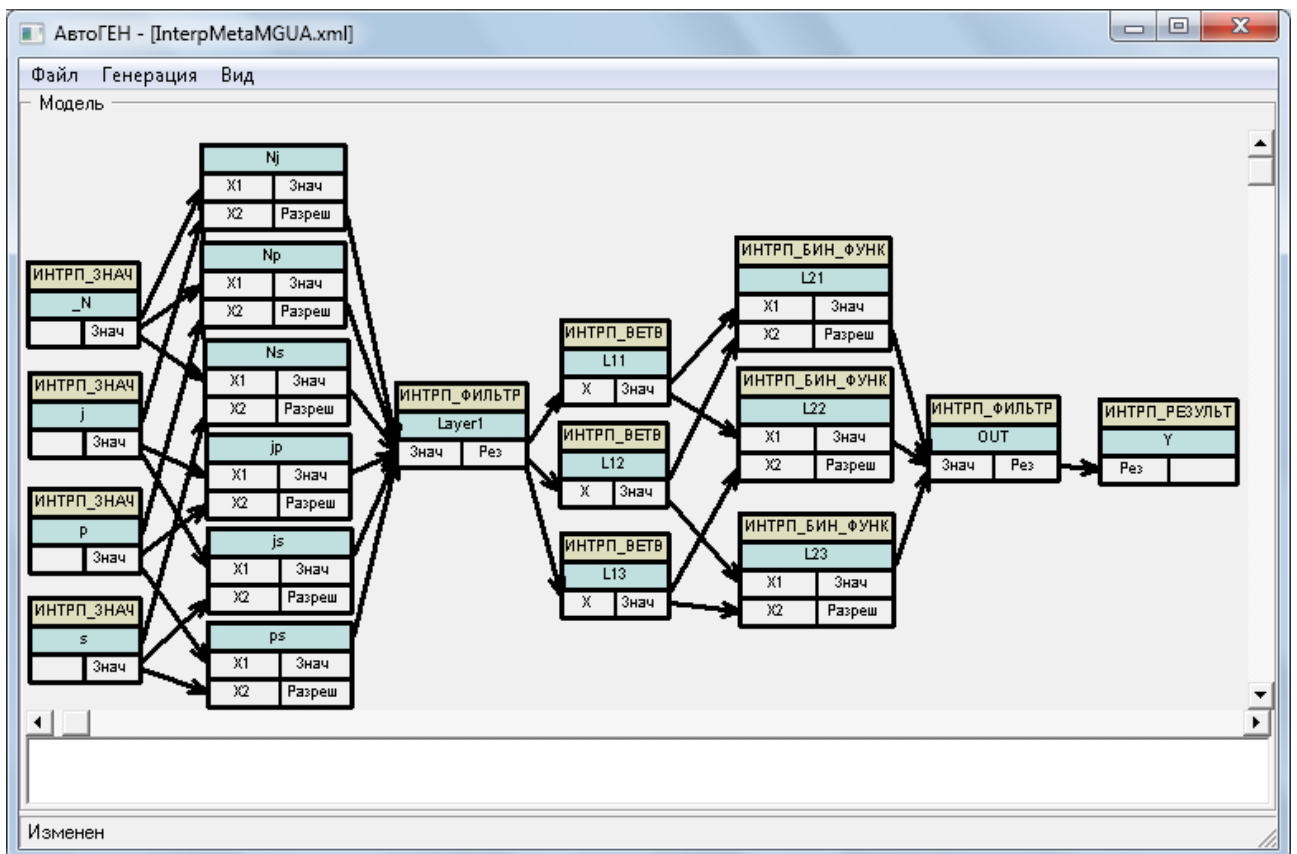


Рис. 1.8. План эксперимента по построению приближенной модели времени исполнения фрагмента программы в системе PGEN++

Необходимо подчеркнуть, что необходимость ручного ввода в план эксперимента достаточно большого количества блоков успешно *устраняется* разработ-

кой дедуктивных блоков [52], автоматически строящих произвольную МГУА-модель по заданному описанию ее структуры.

В результате интерпретации вышеуказанного плана была получена следующая приближенная модель, записанная на языке М-скриптов:

```

jp    =    inline(' (1.91) .*X1+(-0.319) .*X2+(1.914) .*X1.*X2 ',
'X1', 'X2');
js    =    inline(' (3.537) .*X1+(-0.436) .*X2+(1.268) .*X1.*X2 ',
'X1', 'X2');
L21   =    inline(' (0.582) .*X1+(0.226) .*X2+(0.011) .*X1.*X2 ',
'X1', 'X2');
Y = L21(jp(j,p),js(j,s));

```

На следующем рис. 1.9 приведены графики реального (исходные данные) и приближенного (рассчитанные данные) времен исполнения.

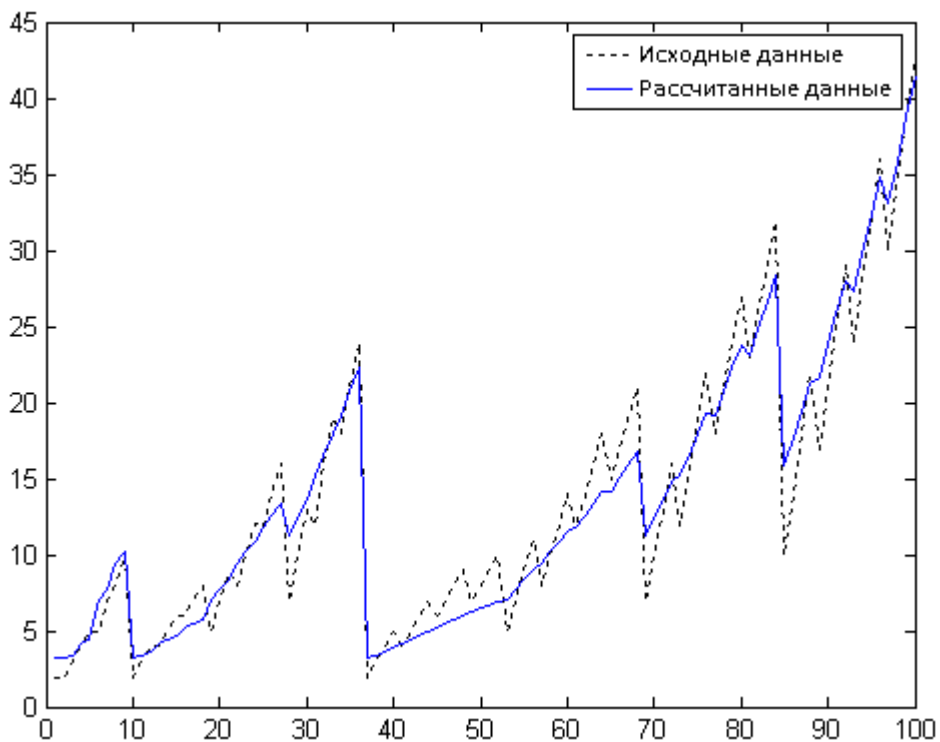


Рис. 1.9. Графики исходного и рассчитанного времен исполнения

По графикам достаточно хорошо видно, что расчетное время исполнения отличается от реального в пиках на 3÷5 миллисекунд. Это приемлемый показатель, если учесть простоту выбранной нами модели. Вероятно, более сложная, например, трехслойная (трехрядная) модель могла бы показать даже лучшие резуль-

таты. Возможно, что лучшие результаты были бы показаны и при использовании иных элементарных интерполяторов. Итак, в данном пункте мы проиллюстрировали возможность применения разработанной нами технологии планирования эксперимента для реализации классического МГУА.

### **Выводы к первой главе**

В данной главе проведен краткий обзор формализмов, используемых для описания и моделирования последовательных и параллельных процессов. Показано, что наиболее перспективными являются формализмы описания кусочно-линейных процессов. Сделан вывод о том, что наилучшим вариантом такого формализма является тот, который может быть получен в результате включения эволюционного линейного оператора перехода в универсальную алгоритмическую машину (ОСМ или МТ). Предложен ряд соответствующих расширенных машин Тьюринга (РМТ), предназначенных для индивидуальной работы (ЭвРМТ) или работы в составе группы таких машин (ЭГРРМТ в составе СРМТ). Показано, что такие машины способны представить произвольные сложные функции, в том числе функции, реализуемые ИНС прямого распространения, при этом они обладают возможностями к обучению и трансформации представляемых ими ИНС.

Предложена параллельная РМТ (ПарРМТ) — простой вариант абстрактной предельной параллельной ОСМ, являющийся, возможно, несколько более конструктивным вариантом параллельной МТ по сравнению с иными известными автору аналогами.

Предложена порождающая РМТ (ПорРМТ), являющаяся предельной порождающей ОСМ. Показано, что системная РМТ (СРМТ) является минимальной абстрактной составной ОСМ.

Доказано, что в рамках СРМТ могут быть реализованы векторный и конвейерный параллелизм, что в сочетании с реализацией парадигмы «портфеля задач» в рамках ПарРМТ/ЭГРРМТ покрывает основные потребности в описании параллелизма процессов. Доказано, что СРМТ позволяют представить модели, базирую-

щиеся на аппаратах нейронных сетей прямого распространения, дискретных марковских цепей (к которым сводятся также марковские системы массового обслуживания), дифференциальных уравнений в частных производных, обыкновенных дифференциальных уравнений. Сформулировано и более общее утверждение о представимости СРМТ формализмов, сводимых к множеству последовательных и параллельных процессов, подразумевающих исполнение итераций с решением одной или нескольких СЛАУ.

Продемонстрировано, что система моделирования на базе ОСМ (расширений ЭГРРМТ и СРМТ) PGEN++, позволяет компактно описывать/генерировать, обучать, анализировать и трансформировать (упрощать) ИНС прямого распространения.

Предложена технология планирования и реализации эксперимента по построению комбинированных моделей данных (по принципу МГУА с каскадным частичным пересчетом при выборе наилучшего интерполятора), включающих нейронные сети, полиномы и произвольные функции. Предложен набор классов для системы моделирования PGEN++, реализующий данную технологию. Проиллюстрировано использование данных классов для построения моделей коэффициента преломления воды и скорости выполнения фрагмента программы.

## Глава 2. Нейросетевое моделирование турбулентной вязкости в задачах, связанных с получением общей картины воздушных потоков

Целью данной главы является формулировка основных положений, связанных с упрощенным моделированием турбулентной вязкости в численных экспериментах. Для реализации данной цели поставим следующие задачи:

а) провести обзор работ, посвященных обычному (на традиционных одноядерных компьютерах) и параллельному расчету картины воздушных потоков в трехмерной области сложной геометрической конфигурации, представляющей городскую застройку в пределах строений в районе одной или нескольких улиц;

б) выбрать подход к моделированию картины воздушных потоков;

в) рассмотреть различные методики моделированию турбулентности в рамках выбранного подхода с точки зрения точности и трудозатратности;

г) выбрать методику с учетом возможности применения приближенных выражений;

д) сформулировать упрощенные модели турбулентности, основанные на выражениях, приближающих результаты более точного расчета по выбранной методике (в следующих главах сформулированные модели будут подвергнуты дополнительному приближенно-алгебраическому упрощению);

е) поставить несколько типовых модельных задач, в рамках которых предполагаются базовые численные эксперименты в данной работе;

ж) определить методику контроля погрешности расчета в численных экспериментах (это особенно важно, поскольку ряд первичных выражений модели будет заменен приближенными выражениями, что может отрицательно сказаться на сходимости при слишком крупном шаге интегрирования по времени).

### 2.1. Обзор подходов к моделированию картины воздушных потоков

Необходимо сразу заметить, что в данной работе основное внимание уделяется именно проблеме количественного расчета *картины воздушных потоков*, по-

скольку такой подход является наиболее точным, наиболее приближенным к реальности. Это наиболее очевидно для задач расчета ветровой нагрузки. Тем не менее, в самом начале нашего обзора нельзя не упомянуть о существовании подходов, не предусматривающих прямого, точного (количественного) расчета воздушных потоков, которые более характерны для задач определения концентраций загрязнителей, выделяющихся на улицах города. Фактор турбулентности в таких задачах обычно рассматривается либо косвенно (как один из множества иных факторов, влияющих на моделируемый случай), либо не рассматривается вообще.

Назовем, в первую очередь, *модели системной динамики* [67], обобщающие и приближающие взаимосвязи между входными параметрами моделируемого процесса, некоторыми его внутренними характеристиками и выходными данными. Существенные сомнения вызывает универсальность таких моделей при расчете множества случаев, существенно различающихся, например, конфигурацией расчетной области. Все же данный класс моделей не предназначен для получения точных значений полей скорости ветра или концентраций загрязнителей, хотя не исключено, что схожие подходы, использующие достаточно простые алгебраические модели, могут быть развиты для моделирования отдельных характеристик течений, например, турбулентной вязкости.

Далее отметим подходы на базе *имитационных моделей*, которые в некоторых случаях способны дать даже количественную картину распределения некоторых требуемых физических величин. Например, в работе [7] построена математическая модель переноса и рассеяния выхлопных газов на автомобильных трассах, по которой можно вычислить математические ожидания концентраций газов в заданных точках. Еще одним примером является работа [93], в которой с помощью имитационной модели рассчитываются распределения сыпучих веществ — для этого вводится сетка, в узлах которой вычисляются вероятности нахождения веществ в данном узле. Такой подход может использоваться, например, для вычисления отдельных величин (турбулентной вязкости, концентраций примесей), характеризующих процесс возникновения и распределения воздушных потоков, переноса примесей. Однако следует отметить, что в таком случае вычислительная

сложность процесса решения не ниже чем при численном решении системы дифференциальных уравнений в частных производных. В самом деле, как в том, так и в другом случаях приходится, фактически, вводить некую расчетную сетку, в узлах которой вычисляются искомые физические величины, независимо от того, представляются ли они вероятностями или имеют прямой смысл. В то же время точность имитационного подхода с очевидностью меньше, поскольку в его рамках точно представить перераспределение физических величин, зависящих от целого ряда иных величин, достаточно сложно. Поэтому применение имитационного подхода для рассматриваемого нами класса задач представляется сомнительным.

Еще одним подходом, не предусматривающим прямого вычисления картины воздушных потоков, является применение приближенных моделей, в основе которых лежат простые аналитические решения. Например, для описания конечных распределений загрязнителей широко используются модели на основе распределения Гаусса (см., например, обзор в [84], также отметим простые модели ADAM, ADMS-3, CAL3QHC [98] и ISC-3 [99]). Это двумерные, «плоские» модели, для которых задаются метеорологические условия (направления, скорость и стабильность ветра), координаты источников загрязнителей и дополнительные параметры (для CAL3QHC, рассчитывающей загрязнение на улице города, — параметры транспортного потока, а для ISC-3, считающей загрязнение в окрестности предприятия, — наличие строений). Вычисляются концентрации загрязнений в заданных точках. Модели такого рода дают достаточно неплохое приближение к точному решению лишь в весьма простых случаях. В случаях, например, достаточно сложной городской застройки, простые аналитические решения чаще невозможны. Это справедливо как для расчетов распределений концентраций загрязнителей, так и при расчете ветровой нагрузки. В частности, классический расчет ветровой нагрузки [58] подразумевает применение простых аналитических соотношений, в которых фигурирует максимальная скорость ветра. Более точный расчет все же требует расчета распределения значений скорости ветра вблизи стены сооружения, уже по которым вычисляются динамическое давление ветра и полная ветровая нагрузка (интеграл от давления по поверхности стены).

Итак, в результате данного краткого обзора методов, не предусматривающих точного количественного расчета картины воздушных потоков, мы пришли к выводу об их явной недостаточности. Перейдем к рассмотрению методов, предусматривающих такой расчет.

Задача точного количественного расчета картины воздушных потоков достаточно хорошо известна. Здесь используются подходы, основанные или на решении дифференциальных уравнений газовой динамики в частных производных (Больцмана, Эйлера, Навье-Стокса, Рейнольдса, Фавра [2, 33, 56, 79, 82, 103]) для сплошной среды или на моделировании среды как совокупности крупных частиц [21], для которых записываются обыкновенные дифференциальные уравнения (ОДУ). Для *интегрирования ОДУ* используются достаточно хорошо известные и разработанные методы (Рунге-Кутты, Адамса, Гира [4, 49, 53, 94] и многие комбинированные методы, например, из числа тех, которые реализованы в системе MatLAB [22, 37]). Для интегрирования уравнений газовой динамики (преимущественно параболических и/или эллиптических) также используются различные численные подходы.

Для *параболических уравнений* используются методы Монте-Карло [31], различные варианты методов расщепления [10, 17, 34, 35, 62, 73] и метода переменных направлений [56, 79]. При этом для аппроксимации производных используются явные и/или неявные схемы различных порядков [56, 59, 79]: явные схемы типа предиктор-корректор Лакса-Вендрофа [94, с.844], явные и неявные схемы Кранка-Николсона [35, с.290-293], явная попеременно-треугольная схема [34, с.309; 62, с.622], явная схема Дюфорта-Франкела (для диффузионных членов, см., например, [56, с.96]), неявная схема Годунова и ее модификации (например, схема ENO (Essentially Non-Oscillatory) [24; 79, с.163-165; 103; 106]).

Для *эллиптических уравнений* используются два основных подхода [4, 13, 26, 56, 79, 94]: сведение к уравнению по типу теплопроводности и счет до установления с применением соответствующих численных методов (см., например, [3, 17, 26, 56, 94]), запись разностной схемы непосредственно для эллиптического уравнения и решение полученной системы линейных алгебраических уравнений

(СЛАУ). Для решения СЛАУ в задачах малой и средней размерности используются прямые матричные методы (Гаусса [79], блочные методы [4], сопряженных градиентов [4, 25, 94], неполных сопряженных градиентов Холецкого и другие [4, с.581-582; 56]), в остальных случаях применяются или достаточно экономичные по памяти, хотя иногда и имеющие проблемы со сходимостью итерационные методы (Ричардсона (Якоби), Либмана (Гаусса-Зейделя) [3, 56], различные варианты метода релаксации [56, 94]), или быстрые методы, основанные на быстром преобразовании Фурье (FFT — Fast Fourier Transformation) [3, 4, 25, 94], методе циклической редукции и комбинации этих методов (FACR — Fourier Analysis and Cyclic Reduction [3, 13, 25, 94]). Дополнительно заметим, что быстрые методы также требуют существенных дополнительных затрат памяти [56].

В различных пакетах моделирования воздушных потоков и связанных с ними процессов (FlowVision [28, 72], FLUENT, GAS DYNAMICS TOOL, PHOENICS, Star-CD, PANACHE, AirEcology-P и других, см., например, [74]) наиболее часто применяется подход на базе дифференциальных уравнений в частных производных. Фактор турбулентности при этом или моделируется напрямую (подход DNS — используется расчетная сетка с вихреразрешающей способностью, на которой решаются уравнения Навье-Стокса) или вводятся дополнительные, замыкающие модель соотношения. Дадим краткий обзор наиболее типичных таких пакетов.

Пакет GAS DYNAMICS TOOL<sup>1</sup> позволяет рассчитывать многофазные течения (с учетом тепла, химической кинетики, межфазных переходов) в областях сложной формы путем численного решения трехмерных уравнений Эйлера или Навье-Стокса. В нем используется метод крупных частиц, причем фактор турбулентности не учитывается, в связи с чем данный пакет не вполне универсален.

Пакет Star-CD (информацию можно найти, например, в [19]) позволяет рассчитывать многофазные потоки путем численного решения трехмерных уравнений Навье-Стокса или Рейнольдса. Пакет предоставляет широкий выбор моделей турбулентности, преимущественно на основе одного или нескольких дифференциальных уравнений, и претендует на универсальность.

---

<sup>1</sup> Данные получены с сайта [www.cfd.ru](http://www.cfd.ru)

Пакет FLUENT (см., например, [76, 83]) весьма популярен для расчетов многофазных реагирующих потоков с учетом межфазных переходов и многих других факторов. Имеются различные модели турбулентности и прочих физических процессов. Как и Star-CD, данный пакет претендует на универсальность.

Пакет AirEcology-P [48, 49] предназначен преимущественно для расчетов экологических задач, характеризующихся наличием многофазной многокомпонентной реагирующей среды. В нем также имеются различные модели турбулентности, как на основе простых алгебраических соотношений, так и на базе одного или нескольких дифференциальных уравнений. Заметим, что данный пакет, несмотря на определенную специализацию, вполне может быть отнесен к универсальным за счет его гибкости и открытости: базовая математическая модель проектируется визуально, в виде специальной объектной диаграммы (состав классов объектов — пополняемый, то есть в систему могут оперативно вводиться новые классы уравнений и их решателей), по которой специальный генератор создает настроечно-расчетный модуль, к которому подключаются прочие необходимые блоки, после чего компилируется общая программа, предназначенная для расчета задачи, модель которой описана в виде диаграммы.

В качестве еще двух пакетов, предназначенных преимущественно для расчета экологических задач и, тем не менее, вполне успешно рассчитывающих картину воздушных потоков, назовем PANACHE [84] и ECOSIM [90], в которых также учитывается фактор турбулентности.

Подводя некоторые итоги, необходимо сказать, что в рассмотренных выше *универсальных* пакетах моделирования используются различные модели турбулентности, от простых алгебраических соотношений до сложных систем дифференциальных уравнений. Выбор конкретной модели во многом зависит от специфики моделируемого случая. Очевидно, что далее нам необходимо провести достаточно детальный обзор подходов к моделированию турбулентности, выбрав те из них, которые обладают достаточной точностью и потенциально наиболее пригодны для расчета ветровой нагрузки на одиночные здания и сооружения, а также концентраций загрязнителей в сложной области из одной или нескольких улиц.

Более того, необходимо *определить перспективность того или иного подхода с точки зрения возможности снижения вычислительных затрат путем замены тех или иных соотношений на приближительные и упрощением полученных выражений*. Заметим, что ни в одной из упомянутых нами выше работ о такого рода возможности снижения вычислительных затрат *не упоминалось*. Хотя, при этом, по крайней мере в пакетах моделирования турбулентных воздушных потоков, которые можно напрямую использовать для расчета ветровой нагрузки, упоминалось о снижении вычислительных затрат путем распараллеливания расчета (StarCD, AirEcology-P). Иная ситуация с программами моделирования распространения загрязнений. Что касается зарубежных программ (ADAM, ADMS-3, CAL3QHC, ISC-3, PANACHE, REMSAD, RPM-IV, UAM-IV, UAM-V, WYNDVALLEY и других), то все они рассчитаны на однопроцессорные системы. Что же касается отечественных разработок, то большая их часть (см., например, [7, 14, 18, 20, 29, 34, 54]) не затрагивает вопроса распараллеливания<sup>1</sup>. Поэтому задача снижения вычислительных затрат, обусловленного модификацией самой модели, достаточно актуальна, поскольку ее решение повышает скорость расчета как для параллельных, так и одноядерных/однопроцессорных программ.

## 2.2. Обзор моделей турбулентности

Турбулентность является настолько сложным процессом, что существует весьма большое количество различных подходов к ее моделированию. Соответственно, множество моделей данного явления можно классифицировать по различным признакам [30, 33, 64, 65, 79, 82, 83, 103]. За основу возьмем классификацию подходов к моделированию турбулентности на базе работ [30, 49, 79]. Сразу заметим, что существуют и иные перспективные направления к моделированию турбулентности, в которых она рассматривается как совокупность вихрей различных масштабов [5, 65]. При всей интересности такого подхода нельзя не отметить его

---

<sup>1</sup> Там же, где данный вопрос рассматривается (см., например, [69]), обычно используются сравнительно несложные модели распространения загрязнений, учитывающие не все значимые факторы.

достаточно высокую вычислительную сложность, что противоречит изначальной установке данной работы — упростить по возможности расчет турбулентности, обеспечив его приемлемую точность.

Итак, выделим четыре основных направления моделирования турбулентности:

а) решение уравнений Рейнольдса, Фавра или Навье-Стокса, замкнутых с помощью полуэмпирических моделей турбулентности;

б) прямое решение (DNS — Direct Numerical Simulation [79, 82, 103]) нестационарных уравнений газовой динамики без каких-либо замыкающих соотношений на очень подробной сетке с вихреразрешающей способностью — позволяющей учесть самые мелкие вихри;

в) решение нестационарных уравнений газовой динамики для крупных вихрей (LES — Large Eddy Simulation [79, 82, 103]) с моделированием мелких вихрей;

г) направление DES (Detached Eddy Simulation [30]), где в зоне «гладкого» течения решаются уравнения Навье-Стокса или Рейнольдса (с полуэмпирическими моделями), а в зоне отрыва потока с крупными вихрями используется LES.

При использовании первого направления в стандартные (или схожие с ними) уравнения Рейнольдса, Фавра или Навье-Стокса вводятся вспомогательные соотношения (полуэмпирические модели) для расчета, соответственно, тензора турбулентных напряжений или турбулентной вязкости. Подходы, соответствующие данному направлению, применяются наиболее часто, поскольку позволяют получить результаты с приемлемой точностью для широкого круга задач при невысоких вычислительных затратах.

Второй направление (DNS) позволяет получить весьма точные результаты, но требует чрезвычайно высоких вычислительных затрат, поскольку обычные уравнения аэрогидродинамики решаются на очень подробной расчетной сетке [79]. Такой подход наиболее целесообразен, например, для случая высокоскоростных потоков, который выходит за рамки нашей работы.

Подход, соответствующий третьему направлению (LES, см., например, [79, 103]), позволяет получить достаточно хорошие результаты. Данное направление

является компромиссом между применением классических полуэмпирических моделей и DNS. Однако заметим, что вычислительная сложность данного подхода, даже будучи на один-два порядка меньше чем в DNS, все же остается очень высокой. В качестве еще одного проблемного момента для применения в данной работе отметим, что подход требует большого количества замыкающих соотношений, что рождает достаточно сложные вопросы о том, какие из них могут быть упрощены и насколько корректны такие упрощения. Это, вообще говоря, является предметом рассмотрения теоретической гидродинамики и выходит за рамки данной работы.

Четвертое направление (DES, см. [30]) является компромиссом между первым и третьим направлениями. Возможно получение результатов достаточной точности с меньшими вычислительными затратами, чем в LES, однако они также превышают затраты, характерные для первого подхода. Кроме того, здесь, видимо остается справедливым сказанное выше о проблемах упрощения LES-моделей.

Исходя из вышеизложенного, остановим наш выбор на *полуэмпирических моделях*, которые, как уже было сказано выше, достаточно универсальны, обладают приемлемой точностью при сравнительно невысокой вычислительной сложности. Как будет показано далее, в ряде случаев задача моделирования турбулентности сводится к задаче моделирования одной-двух дополнительных величин, для которых вполне могут быть введены обоснованные упрощающие/приближающие соотношения. Именно такой случай является для нас предпочтительным, особенно привлекательной выглядит идея сведения задачи моделирования турбулентности к задаче моделирования единственной величины — турбулентной вязкости, что соответствует гипотезе Буссинеска [33]. Это значительно более простой, быстро рассчитываемый и легко упрощаемый случай в сравнении с подходами, основанными, например, на прямом использовании замыкающих соотношений для тензора рейнольдсовых напряжений.

Полуэмпирические модели обычно делят на *две группы*: дифференциальные (алгебраические) и интегральные (на базе одного или нескольких дифференциальных уравнений). Введем *третью группу* — интерполяционные модели (алгебраи-

ческие соотношения, интерполирующие результаты, полученные с помощью какой-либо из интегральных моделей), которые могут существовать в двух основных формах:

а) *локально-интерполяционной* (интерполяционные соотношения используются непосредственно как дифференциальные модели);

б) *интегро-локальной* (интерполяционные соотношения включаются в дифференциальные уравнения).

*Алгебраические модели* предполагают, что турбулентное перемешивание в точке определяется локальными характеристиками потока, а именно — плотностью, вязкостью и значениями производных осредненной скорости. Назовем модели Прандтля, Кармана, Болдуина-Ломакса, существуют и иные [79, 103]. Достоинство подхода состоит в его простоте. Недостаток — не учитывает сложные интегральные эффекты, наблюдающиеся в турбулентных потоках. Ниже даны два примера алгебраических моделей для турбулентной вязкости  $\nu_{\text{турб}}$ :

1. Модель Прандтля [63, с.67-68]:

$$\nu_{\text{турб}} = l^2 \left| \frac{\partial U}{\partial y} \right|, \quad (1.1)$$

$$l = cx, \quad (1.2)$$

где  $U$  — вектор скорости течения,  $y$  — нормаль к вектору  $U$ ,  $c = 0,021$ ,  $x$  — расстояние до твердой стенки.

2. Модель Кармана [33, с.556], включающая уравнение

$$l = 0,4 \left| \frac{\partial U}{\partial y} / \frac{\partial^2 U}{\partial y^2} \right| \quad (1.3)$$

и уравнение (1.2).

*Интегральные модели* связывают турбулентное перемешивание с процессами, происходящими как в данной точке, так и в потоке в целом, что позволяет смоделировать, например, эффект памяти турбулентности. Такой подход характерен для моделей К-Е, К- $W$ , К- $W^2$  [103], Абрамовича-Секундова [3], Риварда, Спаларта-Аллмараса [30, 82] и многих других [79, 82, 103]. Достоинство подхода — достаточно хорошая точность описания турбулентных процессов. Недостаток —

более высокая вычислительная сложность. Ниже даны несколько примеров однопараметрических и двухпараметрических моделей турбулентной вязкости, записанных для трехмерного случая в эйлеровых координатах  $(t, x_1, x_2, x_3)$  в среде с молекулярной вязкостью  $\nu_{\text{мол}}$ .

1. *Модель Абрамовича – Секундова* [3, с.24]:

$$\frac{\partial v_{\text{турб}}}{\partial t} + \sum_{i=1}^3 U_i \frac{\partial v_{\text{турб}}}{\partial x_i} = \sum_{i=1}^3 \frac{\partial}{\partial x_i} \left( (\nu_{\text{мол}} + \kappa v_{\text{турб}}) \frac{\partial v_{\text{турб}}}{\partial x_i} \right) + v_{\text{турб}} f \left( \frac{v_{\text{турб}}}{8\nu_{\text{мол}}} \right) D - \gamma S;$$

$$S = \frac{v_{\text{турб}} (\nu_{\text{мол}} + \beta v_{\text{турб}})}{L_{\text{min}}^2}; \quad D = \sqrt{\sum_{i=1}^3 \sum_{j=1}^3 \frac{\partial U_i}{\partial x_j} \left( \frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right)}; \quad f(z) = 0,2 \frac{z^2 + 1,47z + 0,2}{z^2 - 1,47z + 1},$$

где  $\kappa = 2,0$ ,  $\gamma = 50,0$ ,  $\beta = 0,06$ ,  $L_{\text{min}}$  — кратчайшее расстояние до твердой стенки.

2. *Модель Спаларта-Аллараса* [30, 82, 103]:

$$\frac{\partial \bar{v}}{\partial t} + \sum_{i=1}^3 U_i \frac{\partial \bar{v}}{\partial x_i} = \sum_{i=1}^3 \frac{\partial}{\partial x_i} \left( \frac{1}{\sigma_v} \cdot (\nu_{\text{мол}} + \bar{v}) \frac{\partial \bar{v}}{\partial x_i} \right) + c_{b1} \bar{S} \bar{v} + \frac{c_{b2}}{\sigma_v} \sum_{i=1}^3 \left[ \frac{\partial \bar{v}}{\partial x_i} \right]^2 - c_{w1} \cdot f_w \cdot \left( \frac{\bar{v}}{L_{\text{min}}} \right)^2;$$

$$v_{\text{турб}} = \bar{v} \cdot f_{v1};$$

$$\bar{S} = \frac{\bar{v}}{\kappa^2 L_{\text{min}}^2} f_{v2} + \sqrt{\sum_{i=1}^3 \sum_{j=1}^3 \frac{1}{2} \left( \frac{\partial U_i}{\partial x_j} - \frac{\partial U_j}{\partial x_i} \right)^2};$$

$$f_w = g \cdot \left( \frac{1 + c_{w3}^6}{g^6 + c_{w3}^6} \right)^{1/6}; \quad f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}; \quad f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}};$$

$$\chi = \frac{\bar{v}}{\nu_{\text{мол}}}; \quad g = r + c_{w2} (r^6 - r); \quad r = \frac{\bar{v}}{\bar{S} \kappa^2 L_{\text{min}}^2};$$

$$c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1 + c_{b2}}{\sigma_v},$$

где  $c_{b1} = 0,1355$ ,  $c_{b2} = 0,622$ ,  $c_{v1} = 7,1$ ,  $\sigma_v = 2/3$ ,  $c_{w2} = 0,3$ ,  $c_{w3} = 2$ ,  $\kappa = 0,41$ .

3. *Модель К-Е.*

3.1. *Стандартный* (наиболее распространенный, но имеющий существенные допущения) вариант (см., например, [80; 82, с.295-296]):

$$\frac{\partial K}{\partial t} + \sum_{i=1}^3 U_i \frac{\partial K}{\partial x_i} = \sum_{i=1}^3 \frac{\partial}{\partial x_i} \left( \frac{\nu_{\text{эфф}}}{\sigma_K} \frac{\partial K}{\partial x_i} \right) + G_K - E;$$

$$\frac{\partial E}{\partial t} + \sum_{i=1}^3 U_i \frac{\partial E}{\partial x_i} = \sum_{i=1}^3 \frac{\partial}{\partial x_i} \left( \frac{v_{\text{эфф}}}{\sigma_E} \frac{\partial E}{\partial x_i} \right) + \frac{E}{K} (c_1 G_K - c_2 E);$$

$$v_{\text{турб}} = c_\mu \frac{K^2}{E}; \quad G_K = v_{\text{эфф}} \left\{ \sum_{i=1}^3 \sum_{j=1}^3 \frac{1}{2} \left( \frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right)^2 \right\};$$

$$v_{\text{эфф}} = v_{\text{мол}} + v_{\text{турб}},$$

где  $\sigma_K = 1,0$ ,  $\sigma_E = 1,3$ ,  $c_1 = 1,44$ ,  $c_2 = 1,92$ ,  $c_\mu = 0,09$ .

3.2. *Модификация RNG* [82, 92], имеющая весьма хорошие характеристики и успешно работающая как с крупномасштабными, так и с мелкомасштабными вихрями:

$$\frac{\partial K}{\partial t} + \sum_{i=1}^3 U_i \frac{\partial K}{\partial x_i} = \sum_{i=1}^3 \frac{\partial}{\partial x_i} \left( \frac{v_{\text{эфф}}}{\sigma_K} \frac{\partial K}{\partial x_i} \right) + G_K - E;$$

$$\frac{\partial E}{\partial t} + \sum_{i=1}^3 U_i \frac{\partial E}{\partial x_i} = \sum_{i=1}^3 \frac{\partial}{\partial x_i} \left( \frac{v_{\text{эфф}}}{\sigma_E} \frac{\partial E}{\partial x_i} \right) + \frac{E}{K} (c_1 G_K - c_2 E);$$

$$v_{\text{турб}} = c_\mu \frac{K^2}{E}; \quad G_K = v_{\text{эфф}} \left\{ \sum_{i=1}^3 \sum_{j=1}^3 \frac{1}{2} \left( \frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right)^2 \right\};$$

$$c_2 = c_{21} + c_\mu \eta^3 \frac{1 - \eta/\eta_0}{1 + \beta \eta^3};$$

$$\eta = \frac{G_K K}{v_{\text{эфф}} \cdot E},$$

где  $\sigma_K = 0,7194$ ,  $\sigma_E = 0,7194$ ,  $c_1 = 1,42$ ,  $c_{21} = 1,68$ ,  $\eta_0 = 4,38$ ,  $\beta = 0,012$ ,  $c_\mu = 0,0845$ .

3.3. *Модификация Yар* [105], исправляющая ряд погрешностей стандартной модели, позволяющая более корректно обрабатывать случаи некоторых течений:

$$\frac{\partial K}{\partial t} + \sum_{i=1}^3 U_i \frac{\partial K}{\partial x_i} = \sum_{i=1}^3 \frac{\partial}{\partial x_i} \left( \frac{v_{\text{эфф}}}{\sigma_K} \frac{\partial K}{\partial x_i} \right) + G_K - E;$$

$$\frac{\partial E}{\partial t} + \sum_{i=1}^3 U_i \frac{\partial E}{\partial x_i} = \sum_{i=1}^3 \frac{\partial}{\partial x_i} \left( \frac{v_{\text{эфф}}}{\sigma_E} \frac{\partial E}{\partial x_i} \right) + \frac{E}{K} (c_1 G_K - c_2 E) + S_{EP};$$

$$v_{\text{турб}} = c_{\mu} \frac{K^2}{E}; \quad G_K = v_{\text{эфф}} \left\{ \sum_{i=1}^3 \sum_{j=1}^3 \frac{1}{2} \left( \frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right)^2 \right\};$$

$$S_{EP} = \max \left( 0; 0,83 \frac{E^2}{K} \left( \frac{L}{L_E} - 1 \right) \left( \frac{L}{L_E} \right)^2 \right);$$

$$L = \frac{K^{1,5}}{E}; \quad L_E = c_{\mu}^{-0,75} \kappa L_{\min},$$

где  $\sigma_K = 1,0$ ,  $\sigma_E = 1,3$ ,  $c_1 = 1,44$ ,  $c_2 = 1,92$ ,  $c_{\mu} = 0,09$ ,  $\kappa = 0,41$ .

*Интерполяционные модели*, как уже было сказано выше, представляют собой алгебраические соотношения, приближающие результаты, полученные по интегральным моделям. Эти соотношения либо используются непосредственно, либо включаются в какие-либо дополнительные дифференциальные уравнения. Данный вид моделей, по крайней мере с теоретической точки зрения, сочетает достаточную вычислительную простоту с неплохой вычислительной точностью, является компромиссным подходом между двумя предыдущими. Его недостатками являются: а) значительное время, которое может быть потрачено на поиск адекватного интерполятора, б) потенциально недостаточная универсальность (найденный интерполятор может быть валиден лишь в случаях, близких к тем, на которых он обучался).

Подводя итоги вышесказанному, остановим выбор на компромиссных интерполяционных моделях турбулентной вязкости [46], потенциально обладающих достаточной точностью и оставляющих значительное пространство для поиска и упрощения различных интерполяторов. При этом в качестве «опорной» для интерполяции модели турбулентности целесообразно выбрать К-Е (RNG) модель, которая, как показано в работе [49], является одной из наиболее точных (среди общеупотребимых полуэмпирических) как при моделировании обтекания одиночного препятствия — здания (задача класса CEDVAL A-1 [88], данные Метеорологического Института Гамбургского университета), так и (как показано далее в той же работе [49]) для задач моделирования распространения загрязнений на улице большого города.

Наиболее интересным классом интерполяторов представляются нейросетевые [66], применимость которых к анализируемой нами задаче, по крайней мере при использовании нейронных сетей прямого распространения, показана в работе [46]. После обучения соответствующей нейронной сети модель турбулентной вязкости приобретает вид:

$$v_{\text{турб}} = \text{NET}(\bar{x}),$$

где  $\text{NET}(\bar{x})$  — нейросетевая функция от некоторого вектора входных параметров  $\bar{x}$ .

Нейросетевые интерполяторы отличаются высокой универсальностью (это освобождает нас от необходимости тщательного подбора вида и структуры интерполятора для конкретных случаев) при потенциально возможной избыточности: во-первых, количество нейронов может быть завышено (достаточно трудно дать четкую оценку такому количеству по исходным данным), во-вторых, многие нейроны «работают» только на определенных участках активационных функций. Первое говорит нам о том, что вполне возможно упрощение нейросетевого интерполятора: возможны как прямые изъятия отдельных весов и/или нейронов, так и косвенные их удаления путем упрощения нейросетевой функции, в результате которого отдельные ее члены сокращаются. Такому упрощению как раз и будет способствовать второй из упомянутых выше фактов избыточности: если нейрон работает лишь на определенном участке графика активационной функции, то на этом участке активационную функцию вполне можно заменить сравнительно простой полиномиальной или иной функцией. Наиболее целесообразным подходом здесь является, видимо, выработка некоего множества вариантов упрощенных соотношений (основанных на переборе вариантов замен активационных функций) для сети в целом, сочетающих приемлемую точность с вычислительной простотой варианта. Следовательно, необходима разработка соответствующих схем и стратегий, что и будет одной из основных тем главы 3 настоящей работы.

## 2.3. Модельные задачи

Для экспериментов, проводимых в данной работе, были выбраны два основных случая:

а) моделирование распространения инертного загрязнителя на участке сложной формы, включающем несколько улиц большого города, — вполне типичный случай при анализе проектного решения по фактору экологической безопасности;

б) двумерное обтекание одиночного небольшого здания, находящегося между двумя многоэтажными сооружениями, — один из типичных случаев для расчета ветровой нагрузки.

Эксперименты проводились в системе AirEcology-P.

### 2.3.1. Распространение загрязнителя на улицах города

Эксперимент ставился в трехмерной расчетной области сложной формы, представляющей участок, включающий улицу Göttinger Straße города Ганновер (Hannover, Германия,  $52^{\circ}20'$  северной широты,  $9^{\circ}40'$  восточной долготы) и несколько примыкающих улиц. На рис. 2.1 представлен соответствующий фрагмент карты Ганновера. Данные о конфигурации расчетной области и о различных условиях моделирования были получены из работ [86, 87, 96].

Количество узлов неравномерной расчетной сетки составляло  $72 \times 86 \times 45 = 278640$ .

Приведем фрагмент математической модели [48]. Опустим часть уравнений и членов уравнений, преимущественно связанных с наличием химических реакций (загрязнитель инертен), излучения, тепла, тяжелой твердой (пылевой) и жидкой (капельной) фаз, межфазных переходов. Кроме того, опустим данные о постановке соответствующих граничных условий, которые можно получить в работе [49].

Введем в расчетной области правую систему координат  $(x_1, x_2, x_3)$  с вертикальной осью  $Ox_3$ .

$$\frac{\partial U_j}{\partial t} + U \cdot \nabla U_j = \nabla \cdot \left( (v_{\text{мол}} + v_{\text{турб}}) \cdot \nabla U_j \right) - \frac{1}{\rho} \cdot \nabla P; \quad j = 1, 2, 3; \quad (2.1)$$

$$\widehat{D} = \nabla \cdot U;$$

$$\nabla^2 P = \rho \left[ -\frac{\partial \widehat{D}}{\partial t} + \nabla^2 \left( (v_{\text{мол}} + v_{\text{турб}}) \widehat{D} \right) - \sum_{i=1}^3 \sum_{j=1}^3 \left( \frac{\partial U_i}{\partial x_j} \frac{\partial U_j}{\partial x_i} \right) \right]; \quad (2.2)$$

где  $U$  — вектор скорости основной фазы,  $P$  — давление,  $v_{\text{мол}}$  и  $v_{\text{турб}}$  — молекулярная и турбулентная вязкости,  $\rho$  — плотность воздуха. *Модель турбулентности K-E (RNG)*:

$$\frac{\partial K}{\partial t} + U \cdot \nabla K = \nabla \cdot \left( \frac{v_{\text{мол}} + v_{\text{турб}}}{\sigma_K} \cdot \nabla K \right) + G_K - E; \quad (2.3)$$

$$\frac{\partial E}{\partial t} + U \cdot \nabla E = \nabla \cdot \left( \frac{v_{\text{мол}} + v_{\text{турб}}}{\sigma_E} \cdot \nabla E \right) + \frac{E}{K} (c_1 G_K - c_2 E); \quad (2.4)$$

$$v_{\text{турб}} = c_\mu \frac{K^2}{E}; \quad G_K = \left( v_{\text{мол}} + v_{\text{турб}} \right) \left\{ \sum_{i=1}^3 \sum_{j=1}^3 \frac{1}{2} \left( \frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right)^2 \right\};$$

$$c_2 = c_{21} + c_\mu \eta^3 \frac{1 - \eta/\eta_0}{1 + \beta \eta^3}; \quad \eta = \frac{G_K K}{(v_{\text{мол}} + v_{\text{турб}}) \cdot E},$$

где  $\sigma_K = 0,7194$ ,  $\sigma_E = 0,7194$ ,  $c_1 = 1,42$ ,  $c_{21} = 1,68$ ,  $\eta_0 = 4,38$ ,  $\beta = 0,012$ ,  $c_\mu = 0,0845$ .

Распространение инертного загрязнителя описывается уравнением:

$$\frac{\partial C}{\partial t} + (U + W) \cdot \nabla C = \nabla \cdot \left( (D + \alpha v_{\text{турб}}) \cdot \nabla C \right), \quad (2.5)$$

где  $C$  — концентрация загрязнителя,  $W$ ,  $D$  — скорость витания и коэффициент диффузии загрязнителя,  $\alpha$  — коэффициент.

Вышеприведенная модель с сопутствующими граничными условиями сохранилась в одном из стандартных демонстрационных экспериментов системы AirEcology-P, была описана в виде диаграммы, включающей стандартные блоки для основной динамической части (уравнения Навье-Стокса в системе «скорость-давление») и для подмодели турбулентности K-E (RNG). Модель была обработана встроенным генератором программ, скомпилирована и выполнена на 128-ядерном кластере ИГЭУ на процессорах Opteron 2 ГГц, сеть передачи Gigabit Ethernet. При

этом программа осуществляла интегрирование системы дифференциальных уравнений модели по неявным разностным схемам с применением метода локально-одномерного расщепления [49].

На рис. 2.2 показано полученное в результате численного эксперимента распределение воздушных потоков во фрагменте расчетной области.

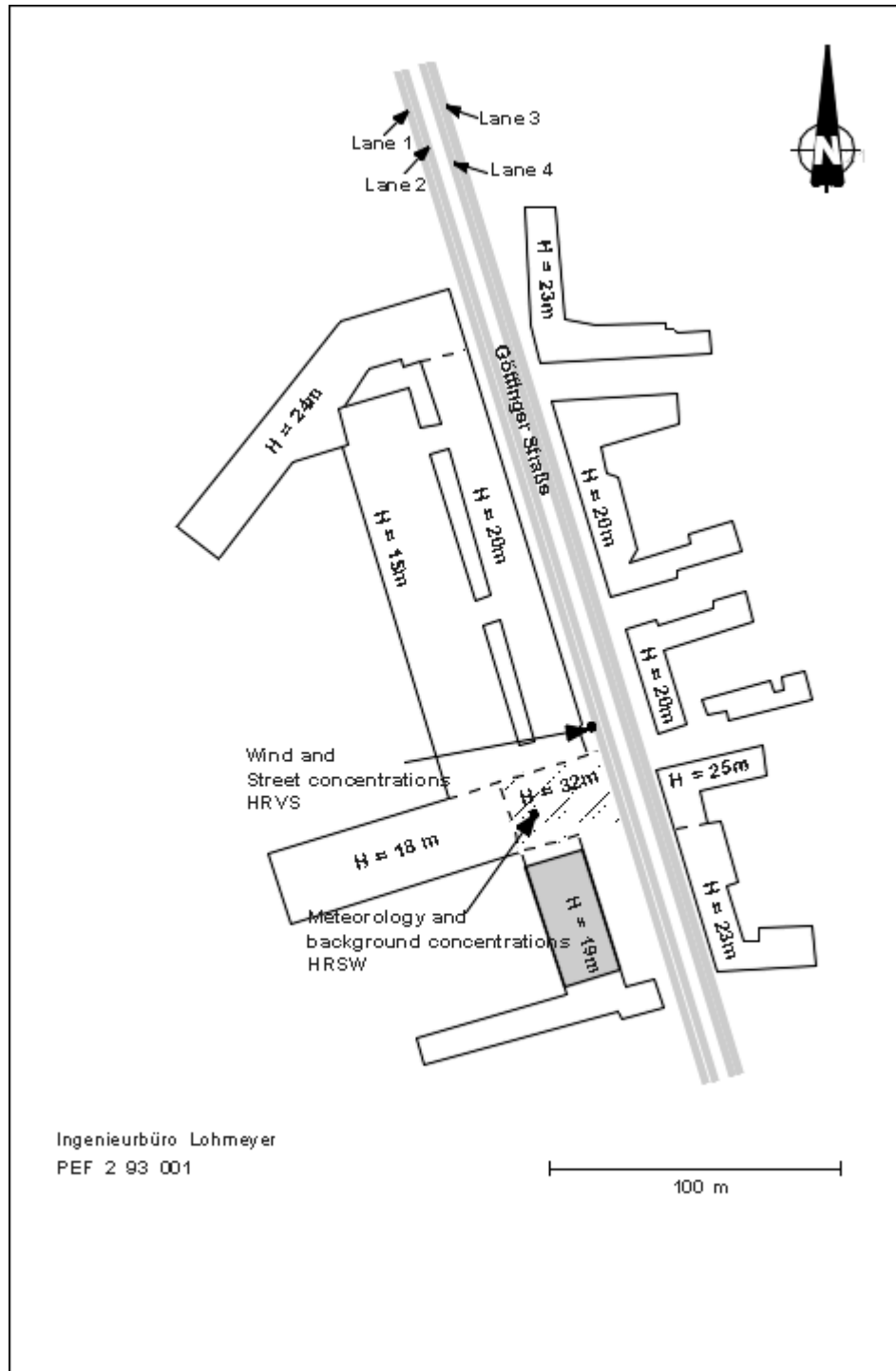


Рис. 2.1. Фрагмент карты Ганновера

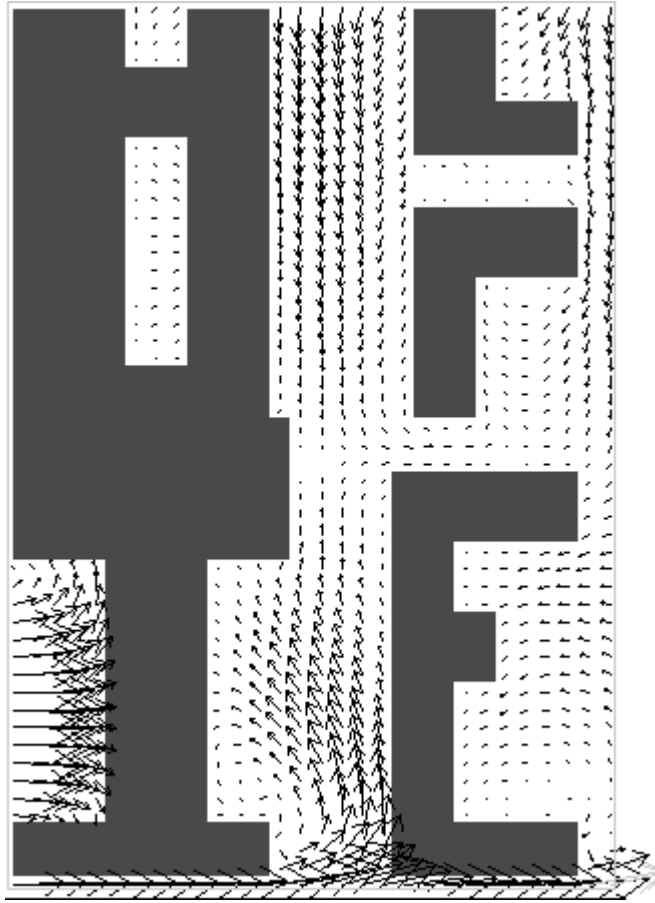


Рис. 2.2. Смоделированное распределение воздушных потоков на высоте 10 метров над участком моделируемой области

### 2.3.2. Обтекание одиночного здания

Двумерная расчетная область (размером  $50 \times 37,5$  м, сетка  $20 \times 15$  узлов, шаг равномерной сетки 2,5 м) была описана каверной с движущейся верхней крышкой (скорость крышки равнялась скорости ветра  $U = 5$  м/с), на нижней границе каверны присутствовало препятствие — обтекаемое здание. На рис. 2.3 показаны схематическая постановка задачи (а) и полученное в численном эксперименте распределение турбулентной вязкости (б). Приведем двумерную математическую модель [3, 56], заметив, что здесь ось  $Ox_1$  является горизонтальной,  $Ox_2$  — вертикальной:

$$\frac{\partial \omega}{\partial t} + U \cdot \nabla \omega = \nabla \cdot \left( (v_{\text{мол}} + v_{\text{турб}}) \cdot \nabla \omega \right); \quad (2.6)$$

$$\nabla^2 \psi = -\omega; \quad (2.7)$$

$$\omega = [\nabla \times \mathbf{U}]_3 = \frac{\partial U_1}{\partial x_2} - \frac{\partial U_2}{\partial x_1};$$

$$U_1 = \frac{\partial \psi}{\partial x_2};$$

$$U_2 = -\frac{\partial \psi}{\partial x_1};$$

$$U_3 = 0,$$

где  $\omega$  — завихренность,  $\psi$  — потенциал (функция тока). Это достаточно классические уравнения аэродинамики для несжимаемой вязкой среды, получаемые перекрестным дифференцированием уравнений Навье-Стокса, осредненных по Рейнольдсу (RANS — Reynolds-Averaged Navier-Stokes) и их вычитанием друг из друга с учетом равенства дивергенции нулю:

$$\nabla \cdot \mathbf{U} = 0.$$

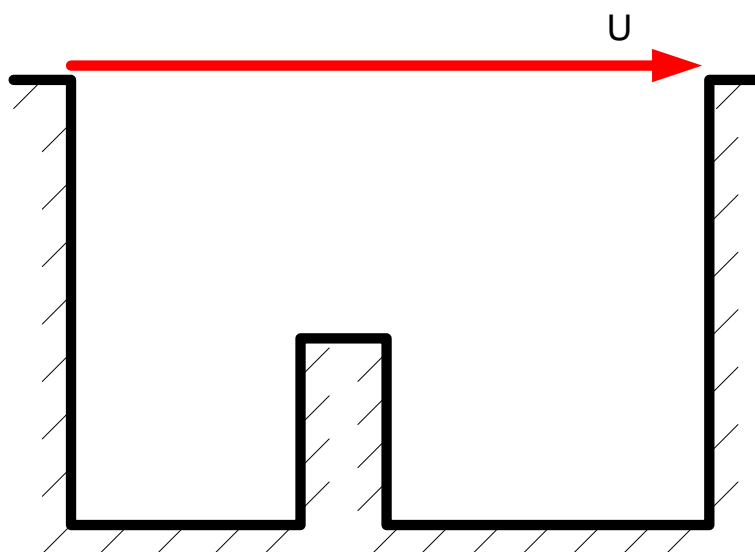
К уравнениям (2.6) и (2.7) присоединялись уравнения модели К-Е (RNG) (2.3) и (2.4), переписанные для двумерного случая (выражение для  $G_K$  считалось с суммированием только по двум пространственным измерениям):

$$G_K = (v_{\text{мол}} + v_{\text{турб}}) \left\{ \sum_{i=1}^2 \sum_{j=1}^2 \frac{1}{2} \left( \frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right)^2 \right\}.$$

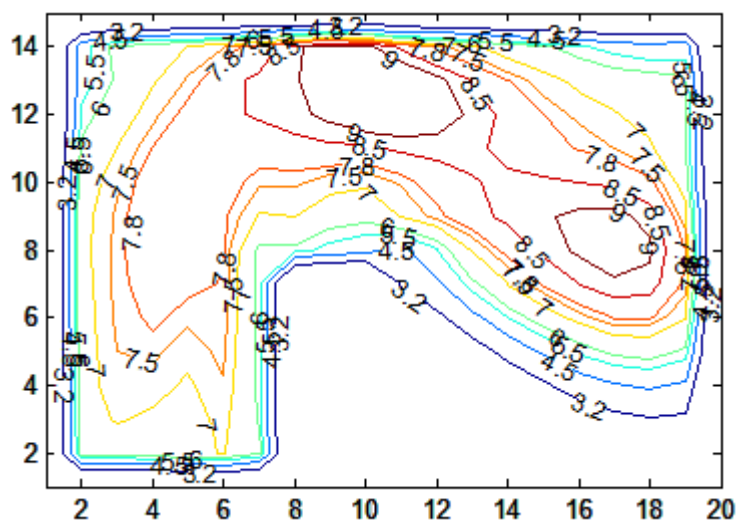
Для завихренности и функции тока граничные условия ставились согласно [56]. Граничные условия для К и Е ставились согласно [49]. На верхней стенке (движущейся крышке) было поставлено условие наличия константной скорости 5 м/с. Для всех этих случаев граничных условий в AirEcology-P имелись стандартные диаграммные блоки.

Математическая модель (2.3), (2.4), (2.6), (2.7) с сопутствующими граничными условиями была описана в виде диаграммы, включающей стандартные блоки для основной динамической части (уравнения Навье-Стокса в системе «вихрь-потенциал») и для подмодели турбулентности К-Е (RNG). Программа, построен-

ная генератором системы AirEcology-P, компилировалась и запускалась на 16-ядерной машине платформы Google's Compute Engine.



а)



б)

Рис. 2.3. Схема расчетной области (а) и распределение турбулентной вязкости (б), полученное с применением К-Е (RNG) модели

#### 2.4. Локальная и интегро-локальная нейросетевые модели турбулентной вязкости

Как уже было сказано выше, необходим поиск компромиссных интерполяционных моделей турбулентной вязкости, базирующихся на математическом ап-

парате нейронных сетей [66]. Уже в локально-интерполяционной (локальной) постановке такие модели, что очевидно, дадут вполне правдоподобные конечные результаты, но будут не вполне корректны в описании промежуточных состояний турбулентных потоков. Соответственно, также представляет интерес рассмотрение локальных нейросетевых моделей, включенных в дифференциальное уравнение переноса турбулентной вязкости, что позволит учесть, например, эффект «памяти потока» и даст некоторый промежуточный класс моделей — интегро-локальных. Необходимо будет проанализировать погрешность и скорость выхода на стационарное решение как локальных (локально-интерполяционных) так и интегро-локальных нейросетевых моделей. Целью здесь будет являться наиболее быстрое решение при достаточно хорошей точности.

Начнем с локально-интерполяционной (локальной) постановки, при которой моделью турбулентной вязкости является непосредственно нейронная сеть:

$$v_{\text{турб}} = \text{NET}(\bar{x}).$$

Для экспериментов была выбрана *трехслойная нейронная сеть* прямого распространения с 7 нейронами в первом слое, с 5 нейронами во втором и с одним нейроном в третьем. Передаточные функции первых двух слоев — *гиперболический тангенс* (в некоторых экспериментах — *экспоненциальная сигмоида*), последний слой — *линейный*. Выход сети интерпретировался как значение турбулентной вязкости. Количество входов варьировалось от одного до четырех,  $\bar{x} = (\overline{x_1, x_2, x_3, x_4})$ . На входы подавались, соответственно, значения:

а) квадрата минимального расстояния  $x_1 = L_{\min}^2$  до ближайшей твердой стенки — *геометрический фактор*;

б) оценки турбулентной вязкости (*оценочный фактор*), рассчитываемой как  $x_2 = L_{\min} |\overline{U}|$ , где  $\overline{U} = (\overline{U_1, U_2, U_3})$  — вектор скорости течения;

в) функции деформации  $x_3 = \sqrt{\sum_{i=1}^3 \sum_{j=1}^3 \frac{\partial U_i}{\partial x_j} \left( \frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right)}$ ;

$$г) \text{ функции вращения } x_4 = \sqrt{\sum_{i=1}^3 \sum_{j=1}^3 \left( \frac{\partial U_i}{\partial x_j} - \frac{\partial U_j}{\partial x_i} \right)^2}.$$

При обучении сети использовались данные, полученные в эксперименте по моделированию распространения инертного загрязнителя на улицах города. Имеющиеся данные представляли результаты моделирования с установившимися значениями полей скорости и турбулентной вязкости, полученные для 18 различных направлений ветра (от 20 до 360 градусов с шагом в 20°). Для обучения брались данные для направления ветра в 20° относительно направления на север. Использовался метод обратного распространения ошибки. Объем обучающей выборки равнялся количеству узлов расчетной сетки (72×86×45). Для проверки адекватности получаемых нейросетевых моделей использовался дисперсионный анализ с критерием Фишера при уровне значимости 0,01.

В таблице 2.1 приведены результаты экспериментов.

Таблица 2.1. Результаты обучения нейронной сети

| N | F <sub>CALC</sub> | F <sub>TABL</sub> | Абсолютные величины |       |      |
|---|-------------------|-------------------|---------------------|-------|------|
|   |                   |                   | D                   | σ     | δ    |
| 1 | 0,233             | 1,0092            | 1,002               | 1,001 | 8,28 |
| 2 | 0,147             | 1,0092            | 0,668               | 0,817 | 8,9  |
| 3 | 0,08              | 1,0092            | 0,346               | 0,588 | 6,14 |
| 4 | 0,061             | 1,0092            | 0,265               | 0,515 | 9,41 |

Здесь N — количество входов сети, F<sub>CALC</sub> — расчетное значение критерия Фишера, F<sub>TABL</sub> — табличное значение критерия Фишера, D — дисперсия, σ — стандартное отклонение, δ — максимальное по модулю отклонение. Как показал дисперсионный анализ, все модели являются *адекватными*, наиболее же точна модель с 4 входами, то есть, учитывающая все вышеперечисленные входные функции. Это подтверждает нашу гипотезу о применимости нейронных сетей в

качестве локально-интерполяционных моделей турбулентной вязкости и создает предпосылки для их применения в интегро-локальных моделях.

Интересной представлялась проверка модели в той же расчетной области для иных направлений ветра, нежели при обучении. Соответствующие эксперименты были проведены, результаты сравнения даны на графиках (рис. 2.4 и рис. 2.5).

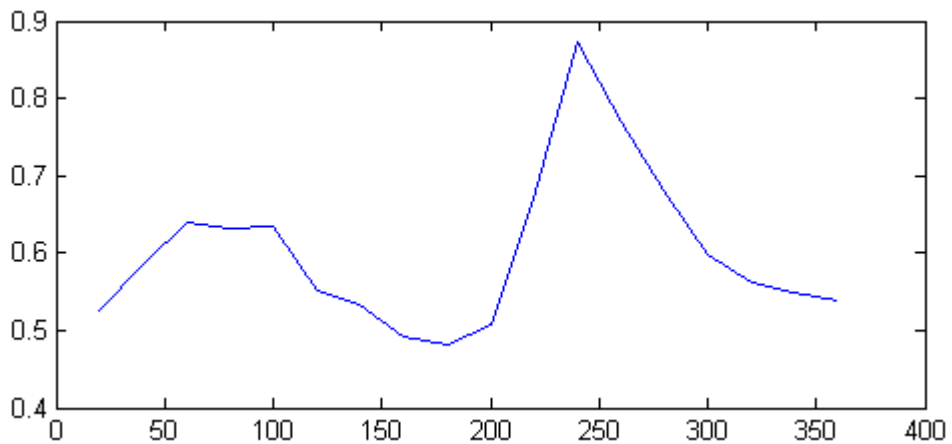


Рис. 2.4. Стандартное относительное отклонение в зависимости от направления ветра (в градусах)

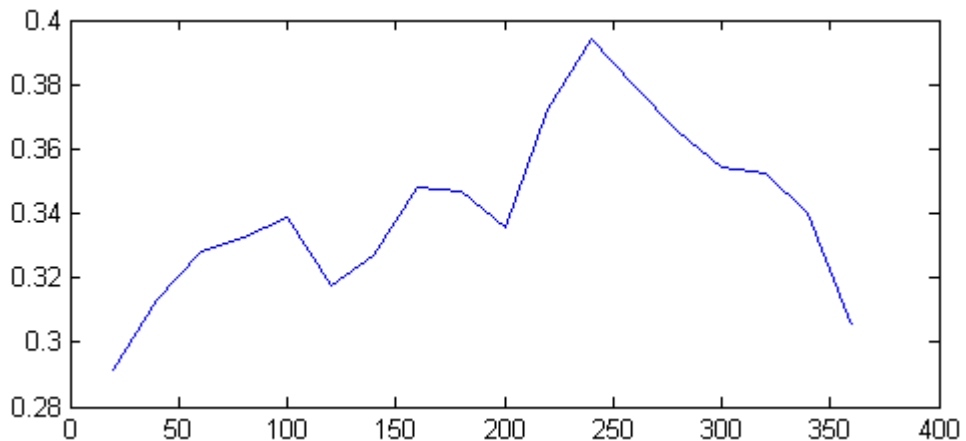


Рис. 2.5. Среднее относительное отклонение в зависимости от направления ветра (в градусах)

Очевидно, что относительные отклонения вполне приемлемы. Пик погрешности отмечается вблизи направлений ветра около  $220\div 240^\circ$ . Такой результат не случаен, поскольку нейронная сеть, обученная по другому направлению ветра

(220°) дала очень схожие графики отклонений с тем же характерным пиком. Косвенно этот факт подтверждает объективность, выражающуюся в наличии определенных закономерностей, построенной модели. Вероятно, вышеуказанный пик объясняется отсутствием учета в текущей модели каких-либо важных факторов генерации или диссипации турбулентности.

Дальнейшие эксперименты проводились для более простого плоского случая моделирования обтекания одиночного препятствия. В качестве образцовой использовалась та же модель турбулентности К-Е (RNG), дополнительно применялась модель Абрамовича-Секундова [3]. Использовались нейронные сети с тремя входами, как показавшие хорошую точность и более простые (это будет особенно важно в дальнейшем, при поиске эквивалентных упрощенных соотношений для сетей).

Построим еще одну (интегро-локальную) модель, также основанную на нейросетевом моделировании поля турбулентной вязкости, однако (возможно при некотором допустимом снижении конечной точности) учитывающую эффект памяти потока и, следовательно, более гладко и правдоподобно приближающую *промежуточные стадии* развития турбулентного течения. Такое требование привело к необходимости ввода дифференциального уравнения-«обертки» для локальной нейросетевой модели.

Нейронная сеть может быть введена в источниково-стоковый член дифференциального уравнения в форме, обеспечивающей плавную минимизацию отклонения турбулентной вязкости от значения, возвращаемого сетью:

$$\alpha(\text{NET}(\bar{x}) - \nu_{\text{турб}}),$$

где  $\alpha$  — некоторая константа,  $\text{NET}(\bar{x})$  — значение, возвращаемое нейронной сетью,  $\nu_{\text{турб}}$  — текущее значение турбулентной вязкости. Для повышения устойчивости завязим внутреннее трение<sup>1</sup> при переносе турбулентной вязкости в 3÷4 раза

---

<sup>1</sup> Это допущение, несомненно, несколько снизит конечную точность расчета, но с учетом повышения его стабильности (а также правдоподобности промежуточных результатов) — вполне оправданное.

и разобьем соответствующий оператор Лапласа на две части — дифференциальную

$$\nabla \cdot (\mathbf{v}_{\text{мол}} + \kappa \nabla v_{\text{турб}}),$$

где  $\kappa$  — некоторая константа, и приближенную алгебраическую, содержащую

$$\frac{\max(v_{\text{турб}})}{L_{\min}^2(x, y)} = \frac{\max(\text{NET}(\bar{x}))}{L_{\min}^2(x, y)}.$$

Внесем соответствующие соотношения в источниково-стоковый член, что и должно обеспечить требуемый эффект повышения устойчивости, и запишем искомое уравнение для турбулентной вязкости:

$$\begin{aligned} \frac{\partial v_{\text{турб}}}{\partial t} + \bar{U} \cdot \nabla v_{\text{турб}} = \nabla \cdot (\mathbf{v}_{\text{мол}} + \kappa \nabla v_{\text{турб}}) + \\ + \alpha \left( \left[ 1 + \frac{\beta}{L_{\min}^2} \right] \cdot \text{NET}(\bar{x}) - v_{\text{турб}} \right); \quad \beta = \gamma \frac{v_{\text{мол}} + \sigma \cdot \max(\text{NET}(\bar{x}))}{\alpha}, \end{aligned} \quad (2.8)$$

где  $v_{\text{мол}}$  — молекулярная вязкость,  $\gamma$ ,  $\sigma$  — некоторые константы. В наших экспериментах использовались следующие значения:

$$\alpha = 0,08; \quad \gamma = 1,1; \quad \sigma = 2; \quad \kappa = 2.$$

Проанализируем погрешности и вычислительные трудозатраты предложенных моделей. При определении трудозатрат интересны не только чистые, но и скомпенсированные повышенной скоростью выхода на стационарное состояние трудозатраты. Для определения скомпенсированных трудозатрат будем использовать контроль шага интегрирования по времени по авторской методике (будет рассмотрена в следующем пункте), использующей правило Рунге в сочетании с регулированием порядка точности разностной схемы в отдельных узлах для предотвращения тенденций к дальнейшему локальному росту погрешности (порядок точности прямо пропорционален количеству маркерных частиц в ячейке, притягиваемых в области с повышенной погрешностью).

Результаты экспериментов по обтеканию одиночного здания приведены в таблице 2.2. Полученные для различных моделей распределения турбулентной вязкости даны на рис. 2.6.



Таблица 2.2. Результаты экспериментов с разными моделями турбулентности

| Модель                                | Время счета (для модельного времени 10 с) |                      | Относительные отклонения от К-Е (RNG) |             |
|---------------------------------------|---|----------------------|---------------------------------------|-------------|
|                                       | Без контроля шага (с)                     | С контролем шага (с) | Среднее                               | Стандартное |
| К-Е (RNG)                             | 242,89                                    | 52,08                | -                                     | -           |
| Абрамович-Секундов                    | 232,80                                    | 50                   | 0,998                                 | 0,002       |
| Локальная (локально-интерполяционная) | 226,48                                    | 28,97                | 0,2                                   | 0,264       |
| Интегро-локальная                     | 259,34                                    | 43,29                | 0,544                                 | 0,459       |

Очевидно, что наибольшей чистой трудоемкостью обладает интегро-локальная модель, что вполне понятно, поскольку в таком случае помимо интегрирования дифференциального уравнения для каждого узла определяется отклик нейронной сети. Наименьшей чистой трудоемкостью обладает локальная нейросетевая модель, не предусматривающая решения дифференциального уравнения. Однако при всей важности этих наблюдений нельзя не отметить, что значительно более интересной характеристикой является скомпенсированная скоростью сходимости трудоемкость и здесь лучшие результаты показали обе нейросетевые модели. Это объясняется тем, что нейронная сеть сразу же выдает конечное (близкое к результатам модели К-Е (RNG), на которых проходило обучение сети) значение турбулентной вязкости для заданных распределения скорости и удаления от твердой стенки, что в случае локальной модели приводит к наиболее быстрой перестройке течения, а в случае интегро-локальной модели дает несколько более медленную, но, возможно, более правдоподобную по динамике перестройку.

Предложенные нейросетевые модели (рис. 2.6, б, г) показывают точность, несколько худшую по отношению к двухпараметрической модели К-Е (рис. 2.6,

а), на которой были обучены сети, но *лучшую по отношению к модели Абрамовича-Секундова* (рис. 2.6, в) с меньшим количеством параметров, что, вероятно, является общей закономерностью. Не исключено, что локально-интерполяционные модели являются аппроксимациями аналитического решения некоего дифференциального уравнения для турбулентной вязкости или даже системы уравнений, о чем косвенно свидетельствует наличие в соответствующей нейронной сети двух нелинейных слоев.

Локально-интерполяционная модель может применяться для быстрых приближительных расчетов, в которых наибольший интерес представляет именно конечное стационарное состояние. Интегро-локальная модель при менее точных конечных результатах может давать более правдоподобные промежуточные, повышая гладкость и, возможно, устойчивость решения. Не исключено использование предложенных интерполяционных моделей для инициализации полей турбулентной вязкости при использовании однопараметрических моделей турбулентности. Интересной также представляется идея ускорения расчета путем совместного использования интегральных и нейросетевых моделей, где нейронная сеть могла бы использоваться в тех блоках расчетной области, где ее погрешность потенциально невелика, например, в зонах с простой геометрией без возвратных течений.

Для дальнейшего рассмотрения выберем *локально-интерполяционную (локальную) нейросетевую модель*, поскольку из предложенных она показала лучшие результаты по точности и скорости, а промежуточные состояния полей течения в данной работе нас не интересуют.

## 2.5. Контроль погрешности вычислений

Решение задач газовой динамики, рассматриваемых в данной работе, часто требует параллельного расчета в связи с большим объемом вычислений. При этом обычно используются явно- неявные и неявные схемы численного интегрирования первого-второго порядка точности как по времени так и по пространству. Это связано с требованиями снижения количества обменов данными на стыках блоков

расчетной области при использовании геометрического параллелизма (распараллеливания по пространству). Данный вопрос подробно рассмотрен в работе [49].

При этом важное значение имеет контроль точности, регулирующий не только величину шага интегрирования, но и порядок метода, исходя из необходимости сокращения вычислений путем применения более простых и менее точных схем в тех областях, где это не влияет критически на общую точность расчета. Существенным моментом является локальное регулирование точности интегрирования по пространству [12, 89], но в данной работе основное внимание уделяется локальному регулированию точности интегрирования по времени. Отметим, что в известных автору работах (см., например, [100]) прибегают к глобальному изменению порядка метода интегрирования по времени по всей расчетной области. Очевидно, что такой подход неоптимален.

Заметим, что с учетом широкого распространения многоядерных процессоров и потенциальных кластерных вычислителей – компьютерных сетей (требующих лишь установки специального программного обеспечения, такого как различные реализации MPI [11], или DVM [27]), актуальна задача разработки новых малозатратных и эффективных методов локального контроля точности интегрирования по времени для указанного выше класса разностных схем. Для достижения этой цели рассмотрим и проанализируем вычислительные трудозатраты нескольких таких схем, определим кандидатов на использование, предложим алгоритм переключения и/или настройки схем с целью достижения оптимального контроля точности расчета. Весьма интересной является идея разработки новых алгоритмов, в основе которых лежит перспективный тезис о применении математических моделей физических процессов к регулировке процесса вычисления путем построения глубоких аналогий между физическим и вычислительным процессами (в качестве примера назовем  $r$ -методы в терминологии Т.Т.Чунг [79] и методы вычисляющей среды [38] в терминологии профессора Ф.Н.Ясинского).

В данном пункте все численные эксперименты будут проводиться в постановке модельной задачи об обтекании одиночного препятствия (см. ранее), которая дополнительно будет рассчитана (для схожего случая) в трехмерном варианте

(математическая модель, включающая уравнения Навье-Стокса в переменных «скорость-давление», взята из работы автора [49], но вместо модели турбулентности К-Е (RNG) использовалась модель Абрамовича-Секундова) на равномерной расчетной сетке  $40 \times 29 \times 35$  узлов, размер ячейки  $3 \times 3 \times 5$  м.

Для интегрирования эллиптического уравнения функции тока или давления использовался метод верхней релаксации. Разностные схемы для динамических уравнений были записаны с противоточными производными первого порядка точности. В качестве схем-кандидатов интегрирования динамических уравнений были взяты: схема переменных направлений второго порядка точности по времени (метод Дугласа, только для двумерной постановки [3]), локально-одномерное расщепление (неявная схема) первого порядка [49], улучшенная явно-неявная схема Эйлера первого-второго порядка (зависит от параметра). В улучшенной схеме неявный компонент рассчитывался по схеме локально-одномерного расщепления, а явный по вполне заурядному шеститочечному шаблону.

### 2.5.1. Анализ разностных схем

Вычислительные трудозатраты схем-кандидатов оценивались в численном эксперименте как с контролем точности (погрешность определялась по правилу Рунге [94], выбранному исходя из меньшей затратности контроля по сравнению с широко применяемым подходом Ричардсона [77, 78]) так и без такового. В данном пункте эксперименты проводились на компьютере с двухъядерным процессором Intel Atom 1,86 ГГц (Windows).

В двумерном эксперименте контроль погрешности проводился один раз в 200 итераций, система интегрировалась до  $T = 10$  секунд модельного времени, начальный шаг по времени составлял  $\tau = 0,0001$ , допустимая погрешность счета  $\varepsilon_{\text{tol}} = 0,001$ . При этом были получены следующие результаты (таблица 2.3).

Таблица 2.3. Результаты по вычислительным трудозатратам схем в двумерном эксперименте

| Способ счета/<br>Численный метод | Время счета, с   |             |   |
|----------------------------------|------------------|-------------|---|
|                                  | Схема<br>Дугласа | Расщепление | Явно-неявная схема Эйлера с параметром $\alpha = 0,5$ |
| Без контроля точности            | 260              | 204         | 282   |
| С контролем точности             | 73               | 129         | 60  |

Проанализируем полученные данные по двумерному эксперименту. Очевидно, что в чистом виде наименее затратна схема локально-одномерного расщепления первого порядка точности по времени. Следовательно, в некритических по погрешности областях применение данной схемы вполне объективно. В случае же наличия контроля точности наиболее выгодна улучшенная схема Эйлера второго порядка точности по времени, которая, видимо, оказалась менее чувствительна к увеличению шага интегрирования по времени, чем схема переменных направлений Дугласа и схема расщепления. Учитывая, что в нашей реализации схема расщепления встроена в улучшенную схему Эйлера, напрашивается вывод о том, что данного тандема вполне достаточно: в критических областях и по умолчанию будет использоваться улучшенная схема ( $\alpha = 0,5$ ), а в некритических имеет смысл увеличивать параметр  $\alpha$ , доводя его до единицы (неявная схема локально-одномерного расщепления), что при приемлемой погрешности дает меньшие трудозатраты.

Дополнительно был проведен схожий эксперимент по обтеканию здания, стоящего на открытой местности, в трехмерной постановке (применялся параллельный расчет в двух режимах: OpenMP или MPI с применением вспомогательных разностных схем Головичева для предвычислений на стыках блоков расчетной области, обрабатываемых различными процессорами): контроль погрешности проводился один раз в 50 итераций, система интегрировалась до  $T = 5$  секунд модельного времени, начальный шаг по времени составлял  $\tau = 0,0025$ , допустимая погрешность счета  $\varepsilon_{\text{tol}} = 0,005$ .

Были получены следующие результаты (таблица 2.4).

Таблица 2.4. Результаты по вычислительным трудозатратам схем в трехмерном эксперименте с контролем точности

| Режим распараллеливания/<br>Численный метод | Время счета, с |   |
|---|----------------|---|
|   | Расщепление    | Явно-неявная схема Эйлера с параметром $\alpha = 0,5$ |
| OpenMP                                      | 568            | 952   |
| MPI   | 855            | 807   |

При расчете с MPI были получены результаты, дающие то же соотношение величин, что и в двумерной постановке. Интересно, что в OpenMP-варианте (без вспомогательных стыковых разностных схем) явно-неявная схема дала худшие результаты по времени, чем схема расщепления: видимо здесь явно-неявная схема не смогла создать условий для такого увеличения шага интегрирования (возможно из-за быстрой утраты устойчивости счета), при котором дополнительные вычислительные затраты на ее реализацию были бы меньше эффекта от увеличения шага. Тем не менее, мы оставим в силе наши сделанные выше предварительные выводы о потенциале различных схем в критических и некритических областях в надежде, что при локальном (а не глобальном), адресном применении явно-неявная схема Эйлера внесет малозначительные дополнительные трудозатраты и ее применение все же окупится уменьшением времени счета.

### 2.5.2. Схема контроля погрешности счета

Заметим, что при контроле точности целесообразно вокруг наиболее критичных точек «организовать» зону безопасности с повышенной «критичностью», падающей по мере удаления от таких точек. Также достаточно хорошей идеей является ввод некоторой инерционности «критичности» областей во избежание резких скачков погрешности. Первое приводит нас к мысли о том, что поле «критичности» (то есть поле значений, величины которых *прямо пропорциональны тре-*

буемому порядку точности разностной схемы) целесообразно описать уравнением Пуассона с источником членом — величиной погрешности. Второе может быть реализовано вводом виртуальных частиц-индикаторов, постепенно перемещающихся в направлении градиента поля «критичности», на которые среда оказывает некий тормозящий эффект во избежание как чрезмерного разгона так и мелких колебаний положения. Тогда, в зависимости от числа частиц-индикаторов вблизи некоторой точки можно сделать вывод о *текущей потребности в применении в ней разностной схемы того или иного порядка точности*. Приходим к выводу о возможности применения метода по типу частиц в ячейках [68]. Математически же, мы вводим в модель вышеупомянутое уравнение Пуассона (2.10) для «критичности» и уравнения динамики частиц-индикаторов (2.11)-(2.12). В разностной улучшенной схеме (2.14) появляется особое выражение (2.13) для расчета параметра  $\alpha$ .

$$\alpha \nabla^2 I = \begin{cases} -k_1 \frac{\text{err}(x, y)}{\varepsilon_{\text{tol}}}, & \text{при } \text{err}(x, y) < \varepsilon_{\text{tol}}, \\ -k_1, & \text{при } \text{err}(x, y) \geq \varepsilon_{\text{tol}}, \end{cases} \quad (2.10)$$

$$m \frac{dV_k}{dt} = k_2 \text{grad}(I(\overline{X}_k)) - k_4 \overline{V}_k |\overline{V}_k|; \quad k = \overline{1, N_p}, \quad (2.11)$$

$$\frac{d\overline{X}_k}{dt} = \overline{V}_k, \quad (2.12)$$

$$\alpha(x, y) = 1 - 0,5 \begin{cases} 0, & \text{при } q(x, y) \leq 1, \\ \frac{q(x, y)}{M}, & \text{при } 1 < q(x, y) < M, \\ 1, & \text{при } q(x, y) \geq M, \end{cases} \quad (2.13)$$

$$q(x, y) = \sum_k \text{in}(k, \overline{(x, y)}),$$

$$H_{i,j}^{s+1} = \alpha(x, y) \widehat{H}_{i,j}^{s+1} + (1 - \alpha(x, y)) \widetilde{H}_{i,j}^{s+1}, \quad (2.14)$$

где  $I$  — поле «критичности»,  $k_1, k_2, k_4$  — некоторые эмпирические константы,  $\text{err}(x, y)$  — погрешность, определенная в ячейке с указанными координатами по правилу Рунге,  $m$  — условная «масса» частицы-индикатора, определяемая эмпи-

рически,  $\overline{V}_k$  — вектор скорости  $k$ -й частицы,  $N_p$  — количество частиц,  $\overline{X}_k$  — вектор координат  $k$ -й частицы,  $M$  — количество частиц в ячейке, при которой она объявляется максимально критичной (в наших экспериментах  $M = 5$ ),  $\text{in}(k, \overline{(x, y)})$  — логическая функция, дающая единицу, если  $k$ -я частица находится в ячейке с координатами  $\overline{(x, y)}$ ,  $N$ ,  $\hat{N}$ ,  $\tilde{N}$  — новые (в момент времени  $s+1$ ) значения интегрируемой функции, соответственно, первое — конечное, второе — рассчитанное локально-одномерным расщеплением, третье — рассчитанное явной схемой Эйлера первого порядка с противоточными производными и пятиточечным шаблоном для оператора Лапласа. Для ускорения счета при  $\alpha = 1$  член  $\tilde{N}$  не рассчитывается. Контроль шага по времени выполняется по стандартной схеме [94] с применением правила Рунге.

Отметим, что в начале счета частицы случайным образом распределяются по расчетной области. Их количество целесообразно выбрать, например, равным четверти ячеек области. Уравнение Пуассона (2.10) для «критичности» интегрируется на каждой стадии контроля до установления. При этом используются граничные условия равенства «критичности» нулю на всех границах.

Уравнения динамики частиц (2.11)-(2.12) интегрируются методом Эйлера.

Был проведен численный эксперимент с применением предложенной схемы контроля точности для решения той же задачи, что и ранее, как в двумерной, так и в трехмерной постановках. В двумерной постановке было получено время решения *50 секунд*, что на *16%* *лучше* показателей контроля на базе «чистых» схем (см. таблицу 2.3). В трехмерной постановке были получены значения *366 секунд* (OpenMP) и *631 секунда* (MPI), что на *35%* и *21%* соответственно *лучше* показателей контроля по «чистым» схемам (см. таблицу 2.4).

На рис. 2.7 показаны распределения частиц-индикаторов на различных этапах (ближе к середине (а) и ближе к концу (б)) двумерного эксперимента. Достаточно отчетливо видны различия в расположении наиболее критичных зон, характеризующихся максимальными количествами частиц. Эти различия, видимо, объясняются, преимущественно, постепенным развитием вихря в зоне за зданием

(вихрь возникает у верхней кромки здания на первых секундах моделирования и последовательно опускается и расширяется вниз с течением времени).

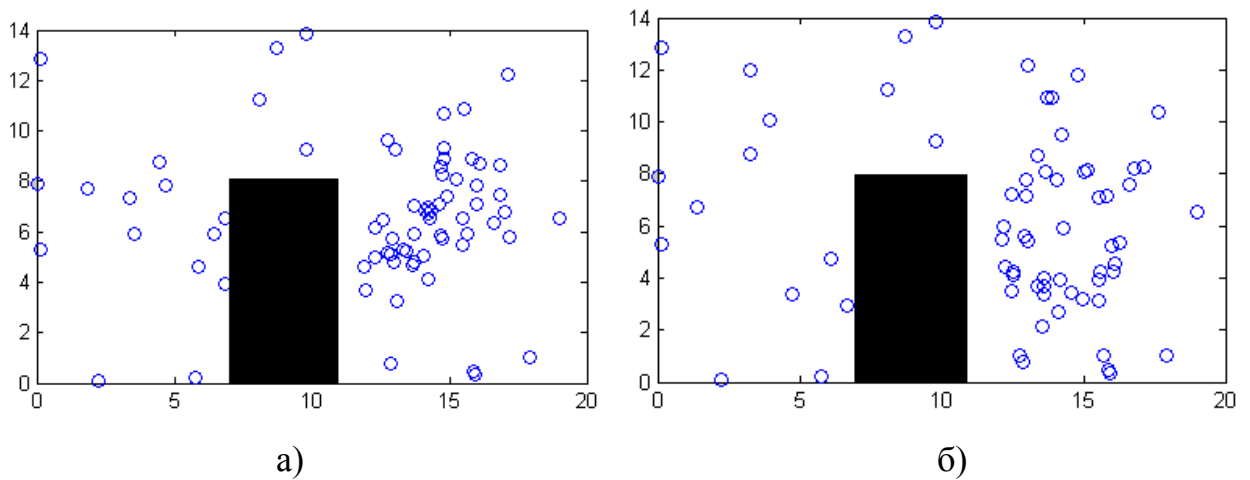


Рис. 2.7. Распределения частиц-индикаторов при  $t = 6$  с (а) и  $t = 9$  с (б)

Итак, предложенный подход показал лучшие результаты среди всех рассмотренных схем контроля точности.

### Выводы ко второй главе

В данной главе рассмотрены работы, посвященные проблеме расчета картины воздушных потоков. Сделан вывод, что наиболее совершенными являются методики моделирования на основе систем дифференциальных уравнений газовой динамики, в которых важнейшую роль имеет учет фактора турбулентности. Кратко рассматриваются основные виды моделей турбулентности и делается вывод о том, что наиболее перспективным направлением снижения вычислительных затрат на ее моделирование, является интерполяционный подход на базе нейронных сетей. Предложены соответствующие нейросетевые локально-интерполяционная и интегро-локальная модели турбулентности. Сформулированы типовые модельные задачи по обтеканию здания в уличном каньоне и по распространению инертного загрязнителя на сложном участке улиц большого города. Проведены численные эксперименты, в которых показано, что предложенные модели имеют достаточно

высокую скорость сходимости к решению и сохраняют приемлемую точность, превышающую точность некоторых однопараметрических интегральных моделей турбулентности.

Для дальнейшего рассмотрения выбрана локально-интерполяционная модель, как более точная по сравнению с интегро-локальной и наименее затратная с вычислительной точки зрения из всех рассмотренных моделей. Декларируется, что задача дальнейшего снижения трудозатрат расчета фактора турбулентности сводится к построению эквивалентных упрощенных соотношений (путем приближенного замещения активационных функций нейронов на более простые) для нейронной сети. При этом наиболее целесообразным подходом здесь является выработка некоего множества вариантов упрощенных соотношений, сочетающих приемлемую точность с вычислительной простотой варианта.

Предложен новый подход к контролю погрешности численного расчета поставленных аэродинамических задач с применением перемещающихся частиц-индикаторов в поле «критичности», определяющих порядок точности и степень «неявности» разностной схемы. Подход показал лучшие результаты по скорости схождения к стационарному решению (что связано с возможностью адекватного увеличения шага интегрирования) и, соответственно, по времени решения в сравнении со всеми рассмотренными фиксированными разностными схемами в условиях контроля погрешности.

## Глава 3. Получение приближенных выражений для ИНС

Целью данной главы является рассмотрение проблемы получения первичных приближенных выражений для нейронной сети, представляющей локально-интерполяционную модель турбулентной вязкости. Для достижения данной цели поставим следующие задачи: проанализировать общую постановку задачи нейросетевой интерполяции, определить наиболее перспективную структуру нейронной сети для рассматриваемой задачи (в том числе вид активационных функций), определить замещающие (более простые, чем активационная) функции, предложить базовые стратегии упрощения нейронной сети.

### 3.1. Общая постановка задачи

Итак, как было показано в главе 2, в данной работе мы будем иметь дело с нейросетевыми моделями турбулентной вязкости, для которых будет необходимо получить замещающие упрощенные аналитические выражения. Вопрос о символическом упрощении первичных полученных выражений является предметом рассмотрения главы 4.

Рассмотрим для начала саму постановку задачи о нейронном интерполяторе [55, 66]. Известно, что такие интерполяторы являются, по сути, упрощенными моделями как отдельных нервных клеток, так и их ансамблей. На рис. 3.1 показана нервная клетка<sup>1</sup>. Дендриты идут от тела нервной клетки к другим нейронам, где они принимают сигналы в точках соединения, называемых синапсами.

---

<sup>1</sup> Рисунок взят из книги [66].

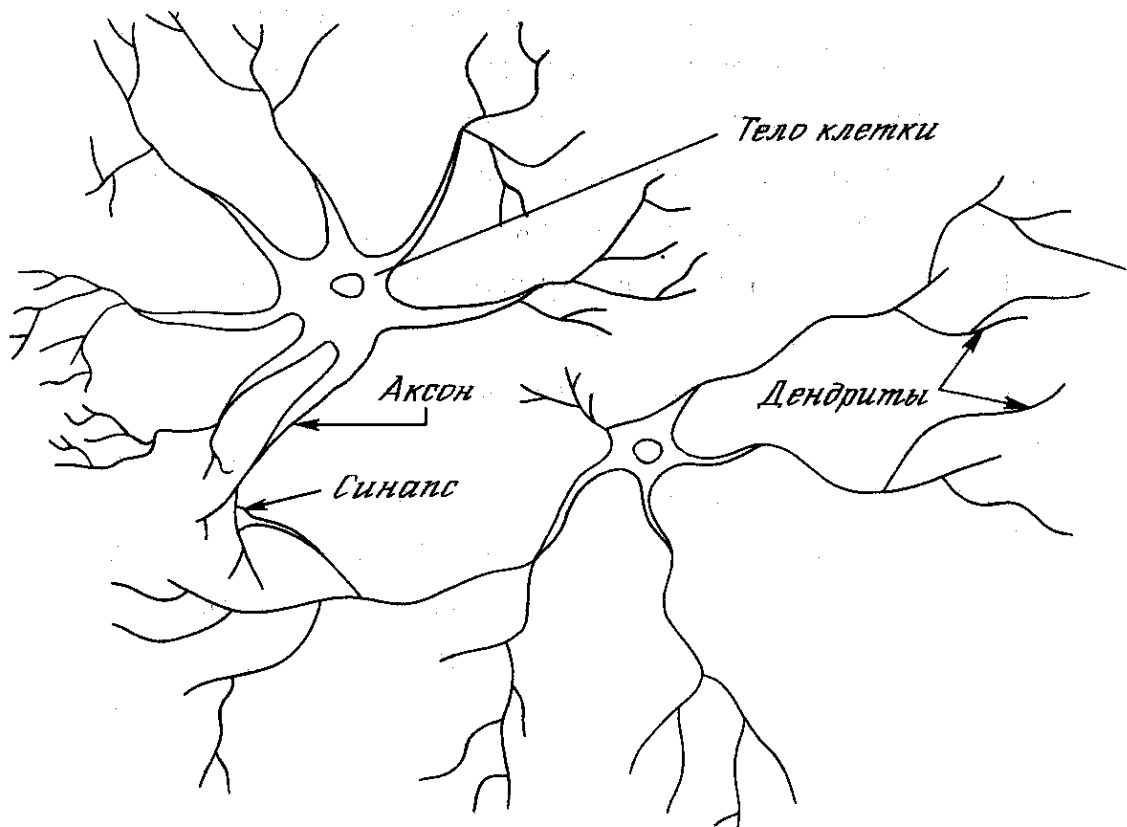


Рис. 3.1. Биологические нервные клетки со связями

Входами и выходами нервной клетки можно считать синапсы, на которых извне/изнутри устанавливаются специфические уровни электрического потенциала, соответственно, входные  $X_i$ ,  $i = \overline{1, N_X}$ , и выходные  $Y_j$ ,  $j = \overline{1, N_Y}$ . Уровни входных потенциалов, соответственно, усиливаются или ослабляются, что эквивалентно умножению на некоторые веса  $w_i$  ( $w_i < 1$  для ослабления,  $w_i > 1$  для усиления), после чего интегрируются (складываются в теле клетки) с неким результатом, который проходит дополнительную обработку: к нему добавляется смещение ( $b_j$  — фоновый уровень) и он пропускается через активационные (или, что то же самое, передаточные) функции  $y_j(s)$ , после чего подается на выходные синапсы. Такая модель нейрона математически описывается следующим образом:

$$\forall j: Y_j = y_j \left( \sum_i X_i w_i + b_j \right).$$

Обычно в теории интерполяции принято считать, что один нейрон имеет лишь один выход, поэтому модель биологического нейрона упрощается до

$$Y = y\left(\sum_i X_i w_i + b\right),$$

где  $y(s)$  — активационная функция,  $b$  — смещение.

При этом схематично нейрон изображается, как показано на рис. 3.2, а, а его модель — как показано на рис. 3.2, б.

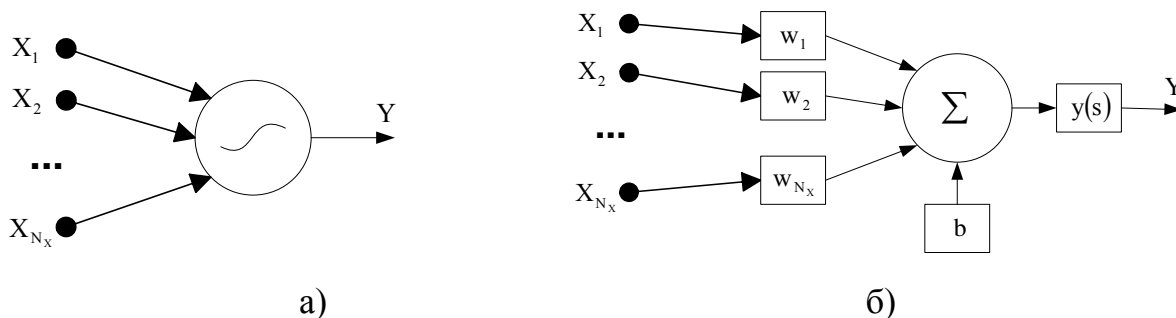


Рис. 3.2. Схематическое изображение нейрона (а) и его модель (б)

Описательные возможности одного такого нейрона, очевидно, не очень высоки и сильно зависят от вида активационной функции. Например, если она представляет собой «ступеньку» вида

$$y(s) = \begin{cases} a, & \text{если } s \geq a/2, \\ 0, & \text{иначе,} \end{cases}$$

то задача интерполяции одним таким нейроном трансформируется в задачу классификации, заключающейся в проведении в пространстве  $X_1 X_2 \dots X_{N_x}$  разделяющей гиперплоскости, которая делит множество исходных точек на некие два класса. Например, при  $N_x = 2$  задача сводится к проведению линии (см. рис. 3.3).

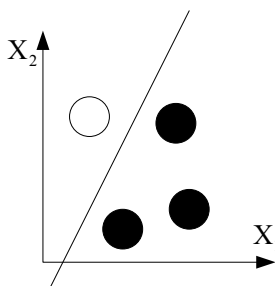


Рис. 3.3. Разделение точек нейроном на два класса при использовании ступенчатой активационной функции

Как было показано еще Минским, такой вариант нейрона не может, например, решить задачу «исключающего или», которая схематично показана на рис. 3.4.

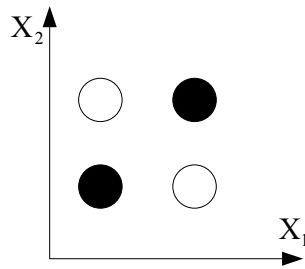


Рис. 3.4. Объекты двух классов, иллюстрирующие задачу «исключающего или»

Поэтому на практике чаще используются нелинейные активационные функции, чаще всего сигмоидальные: «гиперболический тангенс» (см. рис. 3.5) и «экспоненциальная сигмоида» (см. рис. 3.6).

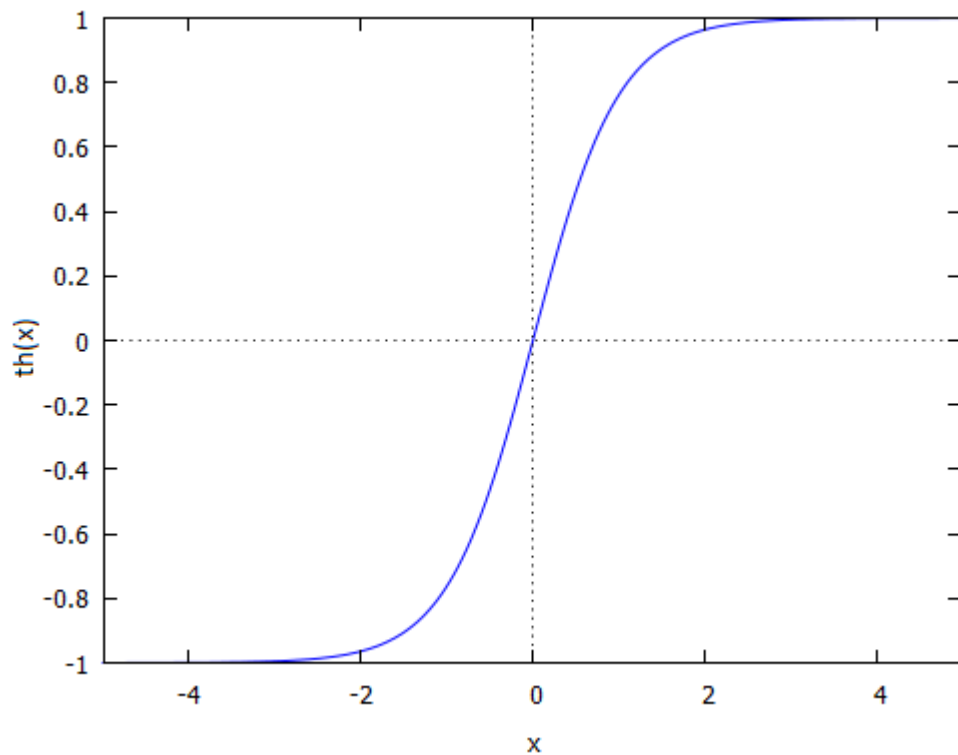


Рис. 3.5. Функция «гиперболический тангенс»

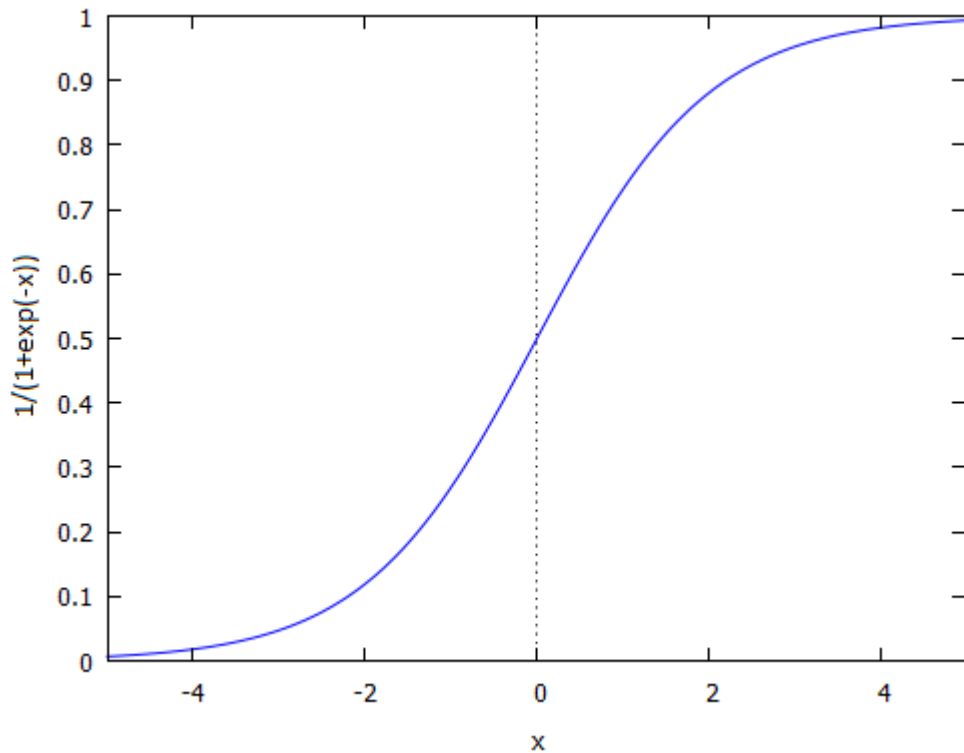


Рис. 3.6. Функция «экспоненциальная сигмоида»

Математически вышеуказанные активационные функции записываются следующим образом:

$$\text{th}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}; \quad (3.1)$$

$$\text{expsig}(x) = \frac{1}{1 + e^{-x}}. \quad (3.2)$$

Заметим далее, что с точки зрения возможностей интерполяции наибольший интерес представляют многослойные ансамбли искусственных нейронов (по аналогии с таковыми для естественных нейронов). Именно такой подход использовался нами выше при построении нейросетевых моделей турбулентной вязкости: использовалась *трехслойная сеть прямого распространения* (все выходы одного слоя соединяются со всеми входами следующего слоя, см. рис. 3.7), в первых двух слоях которой используются сигмоидальные активационные функции, а в третьем слое — линейная активационная функция  $f(x) = x$ . Это достаточно классическая схема — наиболее часто последний слой линеен.

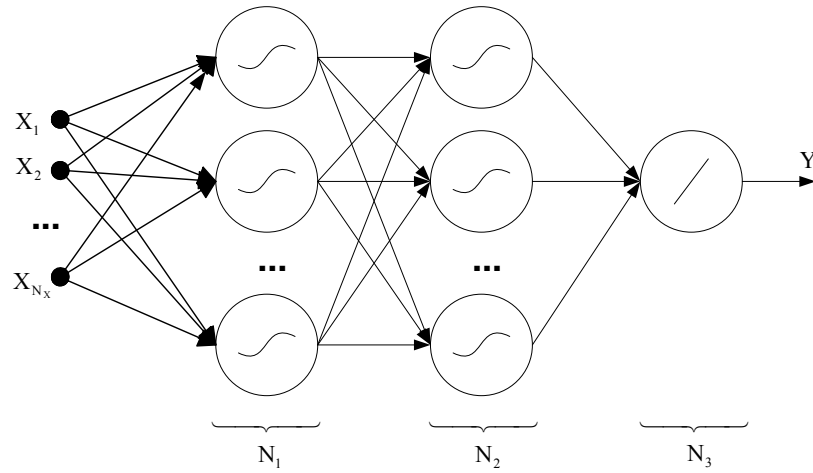


Рис. 3.7. Используемая в настоящей работе сеть прямого распространения

Математически используемая нами сеть может быть описана в виде:

$$IN_{1jk} = X_j; \quad j = \overline{1, N_X}; \quad k = \overline{1, N_1};$$

$$OUT_{1k} = f_1 \left( \sum_j IN_{1jk} w_{1jk} + b_{1k} \right); \quad j = \overline{1, N_X}; \quad k = \overline{1, N_1};$$

$$IN_{2jk} = OUT_{1j}; \quad j = \overline{1, N_1}; \quad k = \overline{1, N_2};$$

$$OUT_{2k} = f_2 \left( \sum_j IN_{2jk} w_{2jk} + b_{2k} \right); \quad j = \overline{1, N_1}; \quad k = \overline{1, N_2};$$

$$IN_{3jk} = OUT_{2j}; \quad j = \overline{1, N_2}; \quad k = \overline{1, N_3};$$

$$OUT_{3k} = \sum_j IN_{3jk} w_{3jk} + b_{3k}; \quad j = \overline{1, N_2}; \quad k = \overline{1, N_3};$$

$$N_3 = 1;$$

$$Y = OUT_{31}; \quad v_{\text{турб}} = Y,$$

где  $f_1(s)$  и  $f_2(s)$  — соответственно, передаточные функции первого и второго слоя;  $IN_{ajk}$  —  $j$ -й вход  $k$ -го нейрона  $a$ -го слоя;  $OUT_{ak}$  — выход  $k$ -го нейрона  $a$ -го слоя;  $N_1, N_2, N_3$  — количество нейронов в первом, втором и третьем слоях соответственно;  $w_{ajk}$  — вес  $j$ -го входа  $k$ -го нейрона  $a$ -го слоя;  $b_{ak}$  — смещение  $k$ -го нейрона  $a$ -го слоя. Отметим, что в нашей задаче  $N_3 = 1$ , поскольку ставится задача получения упрощенной нейросетевой локально-интерполяционной модели только для одной величины — турбулентной вязкости.

О виде функций  $f_1$  и  $f_2$  будет сказано далее, пока же следует оговориться, что в данной работе нами не будет детально анализироваться вопрос об алгоритмах обучения нейронной сети. Было принято решение воспользоваться стандартным и хорошо известным алгоритмом обратного распространения ошибки, при этом наблюдалась достаточно хорошая сходимость при вполне удовлетворительных по качеству интерполяции результатах. Дополнительно отметим лишь, что метод обратного распространения относится к градиентным, поэтому имеет тенденцию к «скатыванию» в локальные минимумы, что при необходимости можно скомпенсировать вводом элементов случайного поиска, а в наиболее критических случаях перейти непосредственно к полному алгоритму случайного поиска. Более подробное рассмотрение данного аспекта выходит за рамки настоящей работы.

### 3.2. Первичное упрощение сети

Предлагается простейшее первичное упрощение сети, позволяющее осуществить небольшое снижение объема вычислений. Для этого достаточно последовательно исключать из сети связи и нейроны, пока погрешность не достигнет некоего заданного исследователем уровня  $\varepsilon_{\text{перв}}$ . Целесообразно установить этот уровень, например, превышающим на 5% изначальную погрешность обученной сети  $\varepsilon_{\text{нач}}$ , то есть

$$\varepsilon_{\text{перв}} = 1,05 \cdot \varepsilon_{\text{нач}}.$$

Организуется итерационная процедура, на каждом этапе для каждой связи определяется ее значимость как модуль разности глобальных ошибок приближения исходных данных после и до изъятия связи (изъятие осуществляется приравнением соответствующего веса связи нулю). Выбирается связь с наименьшей значимостью и соответствующий коэффициент связи уже постоянно приравнивается нулю. Если обнаруживается нейрон, у которого все входные связи исключены, то он изымается из сети вместе со всеми своими связями. Процедура продолжается до тех пор, пока конечная ошибка приближения исходных данных (после

изъятия очередной связи с наименьшей значимостью) не превысит  $\varepsilon_{\text{перв}}$ . При необходимости получения наиболее высокой точности данный этап вполне может быть опущен, в таком случае выполняется лишь основное упрощение сети.

На рис. 3.8 представлена блок-схема предложенного алгоритма.

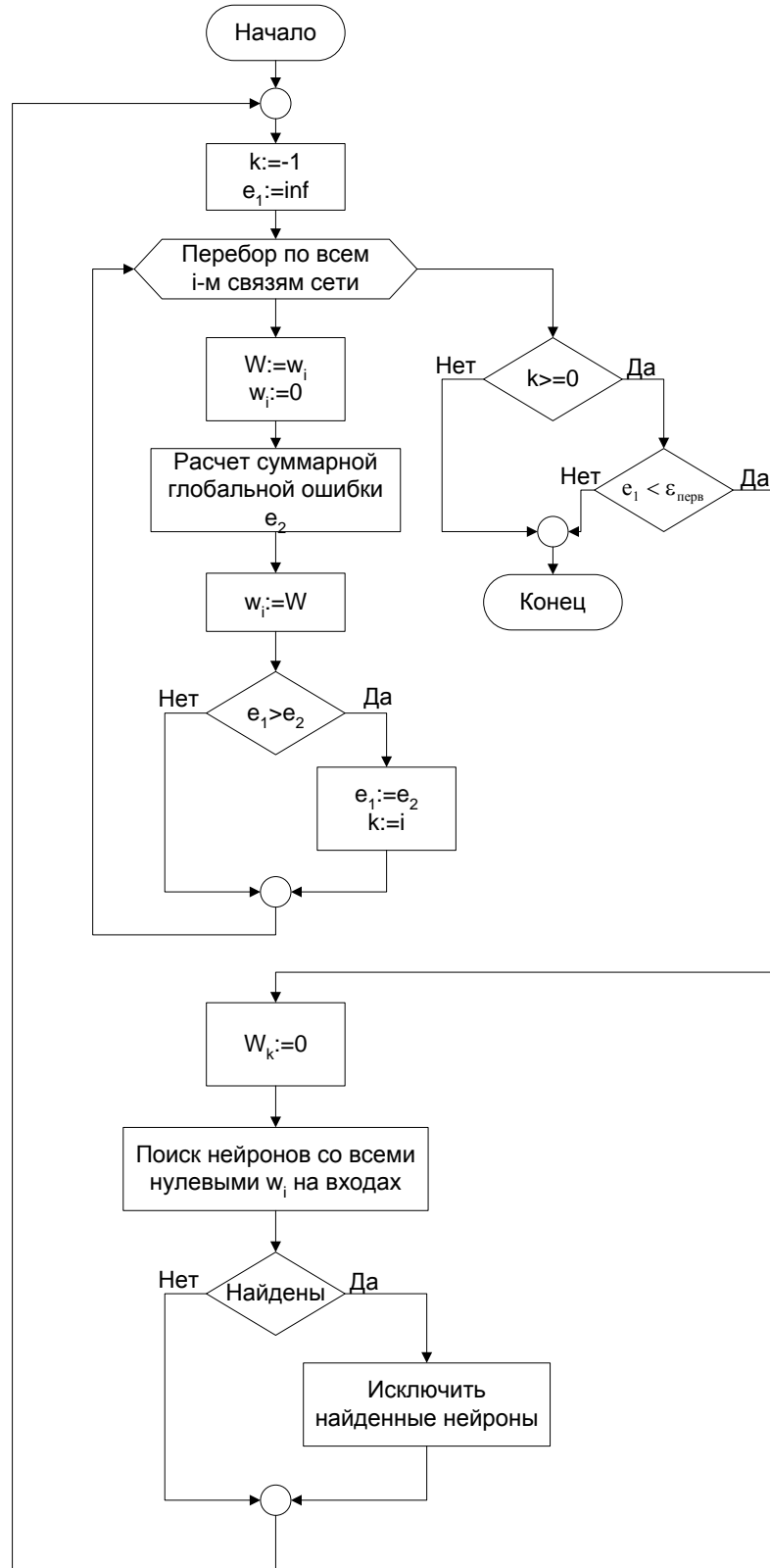


Рис. 3.8. Алгоритм первичного упрощения сети

### 3.3. Основное упрощение сети

Дальнейшее упрощение сети (замена активационных функций на более простые для анализа) является обычной задачей регрессии, решение которой может быть осуществлено *методом наименьших квадратов*. Целесообразно ограничиться такими функциями, для которых данный метод сводится к однократному решению системы линейных алгебраических уравнений, причем сами функции должны быть способны эффективно приближать хотя бы отдельные «рабочие» участки активационных функций.

Рассмотрим вопрос о выборе активационной функции для двух первых слоев. Прибегнем к разложению функции (3.1) в ряд Тейлора, выписав первые несколько членов:

$$\text{th}(x) = x - \frac{x^3}{3} + \frac{2x^5}{15} + \dots$$

Очевидно, что гиперболический тангенс (3.1) является не самым лучшим выбором, несмотря на то, что его прямое вычисление требует двукратного расчета экспоненты и, следовательно, потенциально упрощение его нахождения может давать существенный выигрыш в скорости расчета. В процессорах Intel, например, экспонента вычисляется по формуле  $e^x = 2^{x \cdot \log_2(e)}$  не менее чем за три команды: помещение константы  $\log_2(e)$  в стек, умножение этой константы на  $x$ , возведение 2 в степень. На практике же операций еще больше, учитывая различного рода проверки исходных значений и результата на допустимость.

Однако замещающая функция, подобная вышеуказанному разложению в ряд Тейлора, даже при небольшом количестве членов содержит чрезмерно высокие степени аргумента  $x$ , более того, комбинация таких замещающих функций из обоих первых слоев, вероятно, тоже будет в большинстве вариантов содержать  $x$  в различных и достаточно высоких степенях. Чрезмерное завышение степеней  $x$  может привести к тому, что конечная комбинация замещающих функций будет иметь дополнительные «паразитные» экстремумы. Это может привести к выдаче *упрощенной* сетью не вполне корректных результатов на таких комбинациях

входных данных, которые не содержались в обучающей выборке. Кроме того, не исключено, что в результирующем скомбинированном выражении мы получим завышенное количество членов, а значит и не очень существенный прирост скорости расчета.

Проанализируем теперь экспоненциальную сигмоиду (3.2). Рассмотрим разложение в ряд Тейлора экспоненты:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots,$$

учитывая, что при отрицательном аргументе можно непосредственно записать

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \dots = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots, \quad (3.3)$$

и, кроме того, предположить, что в определенных диапазонах значения  $x$  возможно следующее приближение

$$e^{-x} = \frac{1}{e^x} = \frac{1}{1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots} \approx a_0 + \frac{a_1}{x + b_1} + \frac{a_2}{x^2 + b_2} + \frac{a_3}{x^3 + b_3} + \dots, \quad (3.4)$$

где в ряде случаев коэффициенты  $b_i$  для простоты могут считаться равными нулю.

Очевидно, что замещающие функции в случае экспоненциальной сигмоиды заметно более просты и безопасны для вычисления, поскольку не требуют таких высоких степеней как гиперболический тангенс. Поэтому остановим наш выбор для первых двух слоев нейронной сети именно на активационной функции «экспоненциальная сигмоида». При этом мы, вероятно, не получим потерь качества интерполяции, поскольку описательные возможности сигмоид различного вида весьма схожи.

Исходя из приведенных выше соотношений (3.3) и (3.4), а также учитывая, что выходы одного слоя подаются на входы следующего слоя, причем необходимо обеспечить конечные минимальные степени при переменных (исходя из цели получения максимально простых выражений), были выбраны следующие четыре вида заместительных функций:

$$- f_1(s) = \sum_{i=0}^P a_i s^i, \quad (3.5)$$

$$- f_2(s) = \sum_{i=0}^P a_i s^{-i}, \quad (3.6)$$

$$- f_3(s) = \sum_{i=0}^P a_i s^{0,5i}, \quad (3.7)$$

$$- f_4(s) = \sum_{i=0}^P a_i s^{-0,5i}, \quad (3.8)$$

где параметр  $P$  выбирается минимально возможным при условии достижения требуемой точности приближения активационных функций.

Использовался метод наименьших квадратов (МНК) с весами в целевой функции. Рабочая область значений аргумента  $s$  заместительной функции  $f(s)$  определялась динамически при вычислении значений отклика сети на заданной обучающей выборке  $(\bar{x}_i; y_i)$ ,  $i = \overline{1, N}$ , где  $\bar{x}_i$  — вектор значений входов,  $y_i$  — значение выхода,  $N$  — объем выборки. Рабочая область делилась на некоторое количество равновеликих участков, для каждого из которых подсчитывалось количество элементов обучающей выборки, дающих значение аргумента активационной функции, попадающее в этот участок. Эти частоты попадания и выступали в роли весов элементов  $w_i$  обучающей выборки в целевой функции МНК:

$$\sum_{i=1}^N w_i (y_i - f(s(\bar{x}_i)))^2 \rightarrow \min .$$

Рассмотрим первый вариант (3.5) заместительной функции. Обозначим

$$E_1 = \sum_{i=1}^N w_i (y_i - f_1(s(\bar{x}_i)))^2;$$

$$E_1(a_0, a_1, \dots, a_P) \rightarrow \min .$$

Это квадратичная функция с единственным экстремумом — минимумом. Решим данную оптимизационную задачу аналитически.

$$\begin{cases} \frac{\partial E_1}{\partial a_0} = 2 \sum_{i=1}^N w_i y_i - 2 \sum_{i=1}^N w_i (a_0 + a_1 s_i + \dots + a_p s_i^p) = 0 \\ \frac{\partial E_1}{\partial a_1} = 2 \sum_{i=1}^N w_i y_i s_i - 2 \sum_{i=1}^N w_i s_i (a_0 + a_1 s_i + \dots + a_p s_i^p) = 0 \\ \dots \\ \frac{\partial E_1}{\partial a_p} = 2 \sum_{i=1}^N w_i y_i s_i^p - 2 \sum_{i=1}^N w_i s_i^p (a_0 + a_1 s_i + \dots + a_p s_i^p) = 0 \end{cases}$$

Получаем

$$\begin{cases} a_0 \sum_{i=1}^N w_i + a_1 \sum_{i=1}^N w_i s_i + \dots + a_p \sum_{i=1}^N w_i s_i^p = \sum_{i=1}^N w_i y_i \\ a_0 \sum_{i=1}^N w_i s_i + a_1 \sum_{i=1}^N w_i s_i^2 + \dots + a_p \sum_{i=1}^N w_i s_i^{p+1} = \sum_{i=1}^N w_i y_i s_i \\ \dots \\ a_0 \sum_{i=1}^N w_i s_i^p + a_1 \sum_{i=1}^N w_i s_i^{p+1} + \dots + a_p \sum_{i=1}^N w_i s_i^{2p} = \sum_{i=1}^N w_i y_i s_i^p \end{cases}$$

В матричной форме

$$\begin{bmatrix} \sum_{i=1}^N w_i & \sum_{i=1}^N w_i s_i & \dots & \sum_{i=1}^N w_i s_i^p \\ \sum_{i=1}^N w_i s_i & \sum_{i=1}^N w_i s_i^2 & \dots & \sum_{i=1}^N w_i s_i^{p+1} \\ \dots & \dots & \dots & \dots \\ \sum_{i=1}^N w_i s_i^p & \sum_{i=1}^N w_i s_i^{p+1} & \dots & \sum_{i=1}^N w_i s_i^{2p} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_p \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^N w_i y_i \\ \sum_{i=1}^N w_i y_i s_i \\ \dots \\ \sum_{i=1}^N w_i y_i s_i^p \end{bmatrix}$$

или

$$A \cdot M = B; \quad (3.9)$$

$$A_{jk} = \sum_{i=1}^N w_i s_i^{j+k-2}; \quad j = \overline{1, p+1}; \quad k = \overline{1, p+1};$$

$$B_j = \sum_{i=1}^N w_i y_i s_i^{j-1};$$

$$M = [a_0 \quad a_1 \quad \dots \quad a_p]^T.$$

Решая систему линейных алгебраических уравнений (3.9) относительно M получаем все неизвестные коэффициенты в (3.5).

Рассмотрим вариант (3.6). Легко видеть, что, произведя замену

$$z = s^{-1},$$

мы получаем следующий вариант заместительной функции

$$f_2(z) = \sum_{i=0}^P a_i z^i,$$

который эквивалентен функции (3.5), соответственно задача сводится к предыдущей, которая может быть решена по уравнению (3.9).

Рассмотрим третий вариант (3.7) заместительной функции. Обозначим

$$E_3 = \sum_{i=1}^N w_i \left( y_i - f_3(s(\bar{x}_i)) \right)^2;$$

$$E_3(a_0, a_1, \dots, a_P) \rightarrow \min.$$

Это квадратичная функция с единственным экстремумом (минимумом). Решим данную оптимизационную задачу аналитически.

$$\begin{cases} \frac{\partial E_3}{\partial a_0} = 2 \sum_{i=1}^N w_i y_i - 2 \sum_{i=1}^N w_i (a_0 + a_1 s_i^{0,5} + \dots + a_P s_i^{0,5P}) = 0 \\ \frac{\partial E_3}{\partial a_1} = 2 \sum_{i=1}^N w_i y_i s_i^{0,5} - 2 \sum_{i=1}^N w_i s_i^{0,5} (a_0 + a_1 s_i^{0,5} + \dots + a_P s_i^{0,5P}) = 0 \\ \dots \\ \frac{\partial E_3}{\partial a_P} = 2 \sum_{i=1}^N w_i y_i s_i^{0,5P} - 2 \sum_{i=1}^N w_i s_i^{0,5P} (a_0 + a_1 s_i^{0,5} + \dots + a_P s_i^{0,5P}) = 0 \end{cases}$$

Получаем

$$\begin{cases} a_0 \sum_{i=1}^N w_i + a_1 \sum_{i=1}^N w_i s_i^{0,5} + \dots + a_P \sum_{i=1}^N w_i s_i^{0,5P} = \sum_{i=1}^N w_i y_i \\ a_0 \sum_{i=1}^N w_i s_i^{0,5} + a_1 \sum_{i=1}^N w_i s_i + \dots + a_P \sum_{i=1}^N w_i s_i^{0,5(P+1)} = \sum_{i=1}^N w_i y_i s_i^{0,5} \\ \dots \\ a_0 \sum_{i=1}^N w_i s_i^{0,5P} + a_1 \sum_{i=1}^N w_i s_i^{0,5(P+1)} + \dots + a_P \sum_{i=1}^N w_i s_i^P = \sum_{i=1}^N w_i y_i s_i^{0,5P} \end{cases}$$

В матричной форме

$$\begin{bmatrix} \sum_{i=1}^N w_i & \sum_{i=1}^N w_i s_i^{0,5} & \dots & \sum_{i=1}^N w_i s_i^{0,5P} \\ \sum_{i=1}^N w_i s_i^{0,5} & \sum_{i=1}^N w_i s_i^1 & \dots & \sum_{i=1}^N w_i s_i^{0,5(P+1)} \\ \dots & \dots & \dots & \dots \\ \sum_{i=1}^N w_i s_i^{0,5P} & \sum_{i=1}^N w_i s_i^{0,5(P+1)} & \dots & \sum_{i=1}^N w_i s_i^P \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_P \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^N w_i y_i \\ \sum_{i=1}^N w_i y_i s_i^{0,5} \\ \dots \\ \sum_{i=1}^N w_i y_i s_i^{0,5P} \end{bmatrix}$$

или

$$A \cdot M = B; \quad (3.10)$$

$$A_{jk} = \sum_{i=1}^N w_i s_i^{0,5(j+k-2)}; \quad j = \overline{1, p+1}; \quad k = \overline{1, p+1};$$

$$B_j = \sum_{i=1}^N w_i y_i s_i^{0,5(j-1)};$$

$$M = [a_0 \quad a_1 \quad \dots \quad a_P]^T.$$

Решая систему линейных алгебраических уравнений (3.10) относительно  $M$  получаем все неизвестные коэффициенты в (3.7).

Легко видеть, что для случая (3.8) можно воспользоваться тем же приемом, что и для (3.6) — произвести соответствующую замену переменной.

В общем случае, с учетом того, что заместительные функции являются лишь промежуточными, а конечной будет их комбинация, для которой достаточно сложно изначально определить, заместительные функции какого вида предпочтительнее для отбора, для каждого нейрона целесообразен параллельный (с применением OpenMP) *отбор сразу нескольких вариантов* (в дальнейшем будут перебираться все комбинации таких найденных для различных нейронов вариантов) функции, руководствуясь не только критерием наилучшего приближения рабочего участка активационной функции, но и какими-либо простыми стратегиями получения возможно более простой, а следовательно и оптимальной конечной комбинации. Данный вопрос будет рассмотрен в следующем пункте.

### 3.4. Новая стратегия получения оптимальной комбинации заместительных функций

Предлагается новая стратегия, в основе которой лежит идея невозрастания свыше определенного предела максимальной степени входных переменных в конечном выражении. Нами рассматриваются трехслойные сети с линейным третьим слоем, в которых данная стратегия может быть реализована путем определения большей толерантности выбора упрощенных функций (типа *REV*) с отрицательными степенями во втором слое, если в первом преобладают функции (типа *DIR*) с положительными степенями, и наоборот, повышения толерантности выбора функций с положительными степенями во втором слое, если в первом преобладают функции с отрицательными степенями.

Под толерантностью подразумевается величина допустимого относительно отклонения анализируемого варианта функции от наилучшего варианта функции. Величину толерантности  $\varepsilon_{\text{тол}}$  для анализируемой во втором слое функции типа  $\text{TYPE} \in \{\text{DIR}, \text{REV}\}$  можно представить следующим образом:

$$\varepsilon_{\text{тол}} = \varepsilon_{\text{мин}} + \alpha \left( 1 - \frac{N_{\text{TYPE}}}{N_{\text{DIR}} + N_{\text{REV}}} \right),$$

где  $\varepsilon_{\text{мин}}$  — минимально возможная толерантность,  $\alpha$  — параметр,  $N_{\text{DIR}}$  — количество отобранных в первом слое функций с положительными степенями,  $N_{\text{REV}}$  — количество отобранных в первом слое функций с отрицательными степенями,  $N_{\text{TYPE}}$  — количество отобранных в первом слое функций типа  $\text{TYPE}$ .

Легко видеть, что, например, при  $N_{\text{TYPE}} = N_{\text{DIR}}$  выражение в скобках равняется  $\frac{N_{\text{REV}}}{N_{\text{DIR}} + N_{\text{REV}}}$ , при этом, если  $N_{\text{DIR}} > N_{\text{REV}}$ , то величина толерантности выбора функций типа *DIR* во втором слое меньше толерантности выбора функций типа *REV*, а при  $N_{\text{DIR}} < N_{\text{REV}}$  — наоборот. Тем самым обеспечивается баланс между количеством функций типов *DIR* и *REV* в первом и втором слоях соответственно.

Адекватность идей, заложенных в данную стратегию, будет показана нами далее, в главе 5, при анализе выражений, полученных при использовании различных вариантов комбинаций заместительных функций.

### **Выводы к третьей главе**

Итак, в данной главе рассмотрена общая постановка задачи нейросетевой интерполяции данных. Обсуждается первичное упрощение сети путем исключения из нее малозначащих связей и нейронов. Анализируется структура большинства используемых в настоящее время универсальных активационных функций, мотивируется выбор активационной функции «экспоненциальная сигмоида».

Предлагаются возможные заместительные по отношению к «экспоненциальной сигмоиде» простые функции. Предлагается процедура отбора таких функций по критерию метода наименьших квадратов, которая дополняется стратегией получения наиболее оптимальной конечной комбинации заместительных функций.

## Глава 4. Символическое комбинирование приближенных выражений. Распараллеливание

Как уже было отмечено в предыдущих главах, первичные выражения для локально-интерполяционной нейросетевой модели турбулентной вязкости, полученные в результате замены активационных функций приближенными, подлежат комбинированию в единое выражение и окончательному, уже аналитическому, упрощению. Таким образом, необходимо воспользоваться методами символьной математики. Здесь основным вопросом является то, будет ли достаточно и оправданно применить возможности какого-либо промышленного пакета символьной математики или же необходима разработка собственного небольшого ядра, обладающего лишь необходимым для наших манипуляций набором возможностей. Следующим станет вопрос о том, насколько велики трудозатраты на поиск оптимального конечного аналитического выражения и возможно ли их уменьшение за счет применения распределенной или параллельной обработки данных.

Итак, целью данной главы является рассмотрение вопросов, связанных с возможными разработкой и оптимизацией производительности ядра символьческой математики, которое будет заниматься символьским комбинированием заместительных по отношению к активационным функций (вопрос об их генерации рассматривался в главе 3) и упрощением полученного результата.

Оценим время вычислений при упрощении нейросетевой модели в варианте работы одного ядра одного процессора. Как уже было показано ранее, процесс выработки оптимальных вариантов скомбинированной упрощенной функции, замещающей собой общую нейросетевую функцию, является, преимущественно, процессом перебора вариантов комбинаций заместительных функций. При этом каждый нейрон может иметь до четырех вариантов заместительной функции, реально —  $2 \div 3$  (для определенности примем среднее значение 2,5). Если сеть имеет  $7 \div 10$  нейронов, то количество вариантов составляет от  $2,5^7$  до  $2,5^{10}$ , то есть, приблизительно, от  $6 \cdot 10^2$  до  $10^4$ . Даже если предположить, что время на комбинирование и упрощение варианта составляет (на современном процессоре с тактовой

частотой не менее 2 ГГц при работе *с одним потоком вычислений*) от 0,5 до 3 секунд, в среднем 1,5 секунды, то общее время анализа составит от  $9 \cdot 10^2$  до  $1,5 \cdot 10^4$  секунд, то есть от 15 минут до 4÷5 часов.

Таким образом, становится очевидной необходимость применения параллельной обработки. Заметим, что время, необходимое для анализа и проработки любого из вариантов, достаточно велико. Это позволяет сделать предположение о том, что возможно построение такого алгоритма, при котором распараллеливание будет оправданно, в частности, доля времени на собственно вычисления будет существенно превышать долю времени на обмены/синхронизации данных.

Подводя итоги этого краткого вступления, поставим следующие задачи: провести обзор современных пакетов символьного упрощения выражений, позволяющих упростить получаемые первичные приближенные выражения; рассмотреть современные подходы к распараллеливанию вычислений для задач такого рода (упрощения вариантов приближения и перебора этих вариантов); разработать, при необходимости, новое, простое ядро символических преобразований с различными оптимизациями, позволяющими ускорить упрощение результата; рассмотреть проблему распараллеливания вычислений, предложить соответствующие методики и алгоритмы; проанализировать полученные результаты, оценить эффект предложенных методик и алгоритмов.

#### 4.1. Обзор пакетов символьной математики

В настоящее время существуют достаточно элегантные и совершенные пакеты, предоставляющие возможность проведения вычислений в символьной форме. Назовем Maple [1, 15], MathCAD [43, 44], Scientific WorkPlace, Derive, MatLAB [15, 22], Mathematica [1]. Данные системы (за исключением системы начального уровня Derive) предоставляют достаточно полноценные возможности для символьного расчета и анализа, в том числе потенциально необходимые нам функции преобразования и упрощения символьных выражений. Заметим, что MathCAD, Scientific WorkPlace и MatLAB используют символическое ядро Maple, поэтому,

видимо, речь идет преимущественно о ядрах от Maple и Mathematica. Первым существенным недостатком решений на базе таких ядер является неотчуждаемость самого ядра, а иногда и непрозрачность (MathCAD, MatLAB) его низкоуровневого интерфейса. Поэтому для работы с ядром приходится использовать пакет в целом, прибегая к генерации и исполнению специализированных скриптов/программ (работа с системой может вестись, например, через СОМ-интерфейсы). Это достаточно громоздкое и не вполне эффективное решение.

Кроме того, перечисленные выше коммерческие пакеты, обладающие встроенными возможностями символьной математики, имеют достаточно высокую стоимость (от одной, как в случае MathCAD, до нескольких, как в случае MATLAB, тысяч долларов). Исходя из идеи, что интеграция предлагаемых нами решений в какую-либо САПР должна быть экономически выгодна/оправданна и заметив, что нецелесообразна интеграция большого математического пакета лишь для реализации функций символьной математики, поскольку в таком случае слишком мало отношение полезности решения к его стоимости, приходим к необходимости рассмотреть существующие бесплатные пакеты символьной математики, в том числе и проекты с открытым кодом.

Отметим системы Maxima [75], YaCaS<sup>1</sup> и Genius<sup>2</sup>. Наиболее простой является, вероятно, Genius, который, к тому же, существует лишь в версии для Linux. Значительно шире возможности системы Maxima, которая действительно умеет упрощать достаточно сложные выражения, однако написана она на языке Lisp, в связи с чем ее эффективность и возможности потенциальной интеграции с кодом, написанным на иных языках программирования вызывают существенные вопросы. Также достаточно широки возможности по упрощению выражений в системе YaCaS, написанной на языке C++, представляющей дополнительно возможность программирования на внутреннем языке. Однако эта система (впрочем, как и все иные упомянутые выше системы), насколько известно автору, не оптимизирована для исполнения на многоядерных процессорах.

---

<sup>1</sup> <http://yacas.readthedocs.org/en/latest/#>

<sup>2</sup> <http://www.jirka.org/genius-reference.pdf>

Подводя краткие итоги обзора бесплатных пакетов, приходится констатировать, что многие из них имеют достаточно сомнительную базовую производительность, при этом ни один из них не обладает распараллеленным ядром, то есть не предусматривает параллельного исполнения операций символьной математики. Это очень существенный недостаток, который не позволяет ядрам таких пакетов достичь наиболее высокой производительности. Действительно, в настоящее время уже достаточно трудно встретить вычислительную систему на одноядерном процессоре, где такое решение было бы естественным. Поэтому одной из наших основных целей является, несомненно, полное использование вычислительных ресурсов современных систем на многоядерных процессорах.

Все вышесказанное приводит нас к идее разработки собственного небольшого ядра символьной математики, которое будет выполнять операции комбинирования и элементарного упрощения символьных арифметических выражений. Такое ядро, несомненно, может быть наиболее быстродействующим решением, поскольку: а) будет избавлено от лишних, не являющихся необходимыми и, видимо, нередко присутствующих в ядрах промышленных пакетов операций преобразования и проверки на совместимость элементов выражений и б) будет оптимизировано (распараллелено) для работы с современными многоядерными процессорами для получения наиболее высокой производительности.

Таким образом, нам осталось рассмотреть различные технологии распараллеливания и выбрать наиболее подходящую.

## **4.2. Обзор технологий распараллеливания вычислений**

Сразу же отметим, что в нашей работе не предполагается применение специализированных языков параллельного программирования, например, Ада, SR, CSP, ZPL, C\*, Оккам и многих других [71]. Основная проблема таких языков состоит в том, что соответствующие трансляторы существуют лишь для определенных и достаточно узких классов многопроцессорных машин и операционных систем. Таким образом, область применения подобных языков в настоящее время

достаточно ограничена. Автору известны два исключения: язык Java [71] и язык Planning C++<sup>1</sup>. Универсальность Java обеспечивается поддержкой виртуальных Java-машин. Однако такая универсальность сопровождается существенным недостатком — сниженной производительностью вычислений, что весьма критично для решаемых в настоящей работе задач. Универсальность Planning C++ обеспечивается трансляцией его дополнительных конструкций на стандартный C++, реализации которого существуют для подавляющего большинства платформ. Параллельные конструкции в настоящее время транслируются с применением технологии OpenMP 2.0, которая также реализована для многих вычислительных систем. К сожалению, этот факт говорит о том, что в существующей реализации Planning C++ дает ту же (в лучшем случае) или даже несколько меньшую эффективность распараллеливания, что и OpenMP. Это, повторимся, может быть весьма критично для решаемых в настоящей работе задач.

Наиболее широким делением прочих технологий распараллеливания вычислений является, видимо, разделение *по способу обмена данными*, который в свою очередь обычно определяется архитектурой параллельной системы [11, 16] — с общей или с разделенной памятью. В случае *общей памяти* используется передача данных через ее общие для всех ядер/процессоров регионы, а в случае *разделенной памяти* применяется передача данных через сообщения в некоей среде передачи данных (см., например, [32]).

Следующим важным признаком классификации технологий распараллеливания является их *уровень*. *Высокоуровневые* средства в значительной степени «скрывают» от программиста особенности конкретной вычислительной среды и операционной системы. Достигается это путем того, что высокоуровневые функции на самом деле являются «обертками» над реальными низкоуровневыми функциями обмена данными, которые проводят все необходимые подготовительные и

---

<sup>1</sup> Авторская разработка. В настоящее время стандарт этого языка находится на завершающей стадии создания и, соответственно, в окончательном виде еще не опубликован. Тем не менее его основные конструкции поддерживаются транслятором, реализующим работу с процедурами с планированием повторного входа [52] — рабочая версия транслятора доступна на сайте автора: <http://pekunov.chat.ru/Progs.htm#Reenterable>

заключительные проверки и прочие действия. Это, несомненно, повышает универсальность (в частности, кроссплатформенность), но несколько снижает эффективность реализации.

*Низкоуровневые* средства существенно менее удобны и универсальны, но обычно именно они позволяют осуществлять тонкую настройку и оптимизацию под конкретную архитектуру. Их целесообразно применять в случае, когда необходима пусть и не универсальное, но наиболее эффективное решение.

В следующую таблицу сведены [27, 39, 57, 70, 71, 81, 91] наиболее популярные и развитые в настоящее время технологии:

Таблица 4.1. Технологии распараллеливания

| Доминирующая архитектура | Уровень                                       |                              |
|--------------------------|---|------------------------------|
|                          | Низкий  | Высокий                      |
| С общей памятью          | Pthreads, Windows API                         | DVM, OpenMP, OpenCL, OpenACC |
| С разделенной памятью    | Winsock API,<br>Linux Sockets,<br>DCOM, Corba | MPI, PVM                     |

Особо рассмотрим случай, когда многопроцессорная система имеет смешанную архитектуру (например, широко встречающиеся в настоящее время кластерные системы с многоядерными узлами). Здесь могут использоваться две основные стратегии:

а) комбинированное распараллеливание расчета, когда в пределах узла кластера используется программирование с общей памятью, а для обменов данными между узлами применяется передача сообщений (такой подход применен, например, в работе [49]);

б) унитарное распараллеливание с «виртуализацией» архитектуры, когда или ядра узла кластера используют для обмена данными друг с другом передачу сообщений, или организуется виртуальная общая память, единая для всех узлов

кластера. Примерами таких технологий являются, соответственно, MPI (рассматривает ядра всех имеющихся процессоров на общих основаниях — как отдельные самостоятельные процессоры, независимо от того, находятся они на одном узле кластера или на нескольких) и DVM (использует разделение данных по процессорам, маскируя факт их распределения путем передачи сообщений или напрямую через общую память). В качестве еще одного любопытного примера «виртуализации» архитектуры можно назвать технологию OpenCL, в которой, фактически предполагается работа с расширителем, в роли которого может выступать любое вычислительное устройство, реализующее соответствующий интерфейс виртуального (реализованного центральным процессором) или реального (обычно это многоядерная видеокарта NVIDIA или Radeon) расширителя.

Решение с комбинированным распараллеливанием может быть более сложным, но и более оптимальным, поскольку способно наиболее тонко и точно учесть особенности архитектуры. Решение с «виртуализацией» архитектуры отличается большей простотой и, в ряде случаев, универсальностью, что, однако, сопровождается несколько меньшей эффективностью реализации.

Таким образом, для выбора технологии распараллеливания нам необходимо определиться с архитектурой вычислительной системы и уровнем требуемых средств. Для этого необходимо в полной мере уяснить основные особенности решаемой задачи.

Отметим, что в решаемой задаче, несомненно будут присутствовать как крупные (варианты приближения нейронной сети), так и мелкие (внутренние фрагменты циклов различных операций упрощения) гранулы параллелизма. В таком случае можно сразу исключить вариант применения лишь средств для систем с разделенной памятью, поскольку их работа с мелкими гранулами параллелизма неэффективна: возникает большое количество пересылок, время на которые может превышать эффект от параллельного счета микрогранул. Далее отметим, что вряд ли оправданным будет и применение комбинированного распараллеливания, поскольку ожидаемое количество крупных гранул, которые можно было бы распределить по узлам кластера, не превышает нескольких сотен. В таком случае более

оправдано применение все же не кластера, а единой вычислительной системы с несколькими многоядерными процессорами<sup>1</sup>. Такое решение представляется достаточным и экономически более выгодным, поскольку стоимость кластера из нескольких машин все же выше стоимости одной машины, например, с двумя процессорами, каждый из которых содержит от восьми (Intel Core i7) до шестнадцати ядер (AMD Opteron 6300, Intel Xeon v2).

Таким образом, в данной работе целесообразно воспользоваться одной машиной с общей памятью. Что же касается уровня используемых средств, то по критерию универсальности (в том числе кроссплатформенности) необходимо выбрать высокоуровневые средства. Задача оптимизации под конкретную архитектуру/операционную систему в настоящее время не ставится.

Рассматривая далее вопрос о выборе конкретной технологии, необходимо сказать, что для нашей задачи вряд ли целесообразно применение вычислительных расширителей на графических видеокартах, поскольку предполагаемые гранулы параллелизма, вероятнее всего, будут содержать внутри себя достаточное количество ветвлений (а не простые векторные или матричные расчеты), что может очень сильно снизить эффективность расчета в микроархитектуре SIMT (Single Instruction — Multiple Threads), характерной для графических расширителей. По этой же причине вряд ли будет эффективен перевод расчетов на «виртуальные» расширители (драйверы для основного процессора, разработанные, например, AMD или Intel), задействующие векторные расширения SSE системы команд основного процессора. Этот факт, видимо, позволяет нам исключить из рассмотрения OpenCL и OpenAcc. Вариант DVM нам тоже не подходит, поскольку данный интерфейс, насколько нам известно, не поддерживает порождения подзадач.

Таким образом, наш выбор останавливается на OpenMP — хорошо развитом и универсальном интерфейсе для программирования систем с общей памятью, представляющем простые и достаточно выразительные средства распараллелива-

---

<sup>1</sup> Это справедливо для текущих решаемых в настоящей работе задач. Для задач большей размерности количество крупных гранул может достигнуть нескольких тысяч и при этом применение кластерной системы, возможно, будет оправданным.

ния, достаточно успешно работающим с гранулами параллелизма большого, среднего, и (за счет объединения гранул в специальном расписании распараллеливания циклов — *guided*) даже достаточно мелкого размера, успешно реализующим порождение подзадач, начиная с OpenMP 3.0.

### 4.3. Общее описание иерархии классов символического ядра и их основных элементов

На рис. 4.1 приведена упрощенная иерархия классов разработанного символического ядра, построенного по технологии объектно-ориентированного программирования (на языке C++). Показаны только те поля и методы классов, которые имеют существенное значение для понимания ключевых моментов специфики работы ядра.

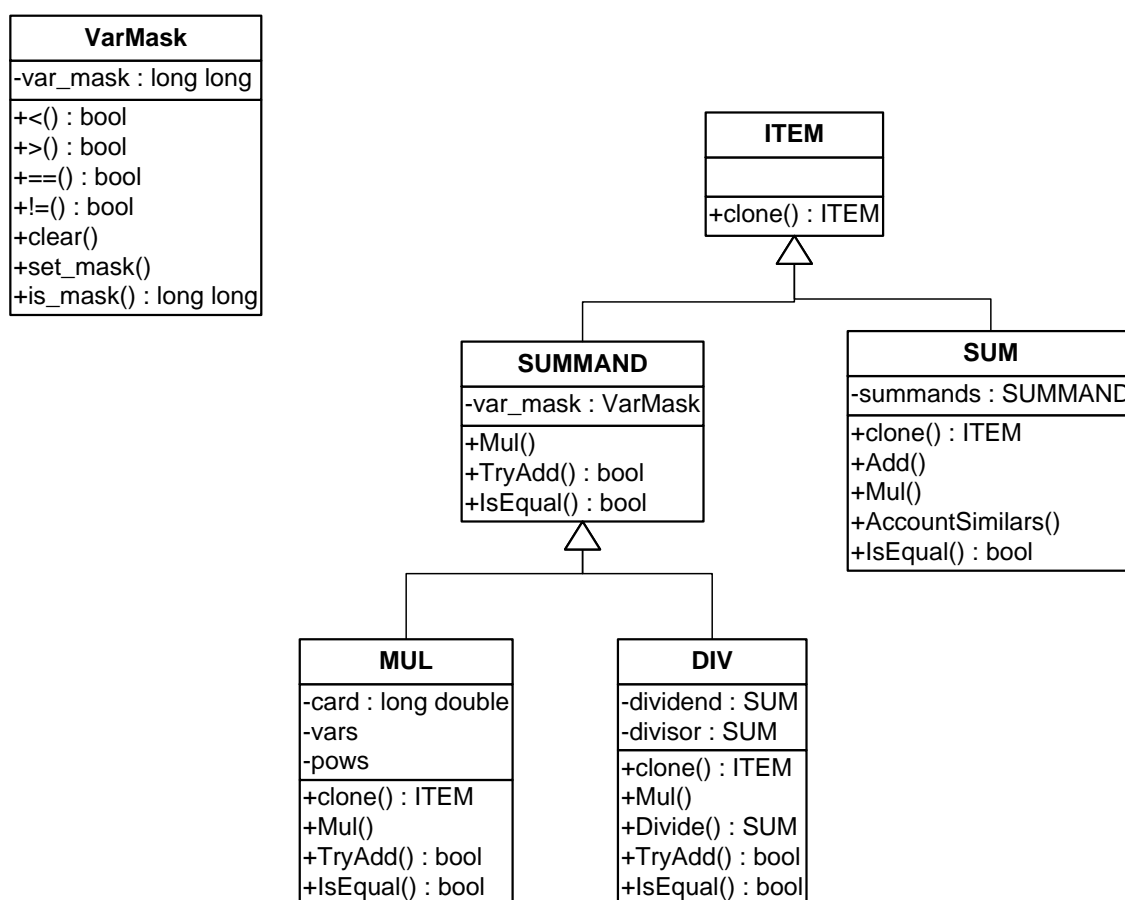


Рис. 4.1. Упрощенная иерархия классов символического ядра

Дадим краткие пояснения к иерархии классов. Класс `VarMask` реализует специальную сущность (*битовую маску*), которая в сжатой форме описывает структуру мультипликативного операнда, представляющего собой произведение различных переменных задачи (в различных степенях), взятое с некоторым коэффициентом. Для подобных выражений значения маски совпадают, что позволяет осуществить разнообразные оптимизации процесса комбинирования и упрощения выражений — более подробно об этом будет сказано в специальном подразделе. В частности, мультипликативные операнды, входящие в некоторую сумму, в таком случае выгодно хранить в классической STL-структуре `map` (элемент стандартной библиотеки шаблонов обобщенного программирования STL [42]), а для этого требуется переопределение ряда операторов сравнения, что и показано в схеме класса `VarMask`. Прочие три функции-элемента (методы) данного класса отражают различные стандартные операции с данными этого класса: ввод в маску и удаление из маски информации о новом члене мультипликативного операнда (соответственно, `set_mask` и `clear`), проверка вхождения переменной в операнд (`is_mask`).

Обратим внимание на символическое обозначение величин, которые будут входить в конечное упрощенное выражение. Это, прежде всего, входы сети (формально обозначаются строчными латинскими буквами, начиная с «а»), и, кроме того, квадратные корни выходов  $s_k$  каждого из  $k$ -х нейронов (до входа в активационную функцию) — для этого нейроны нумеруются в порядке от первого слоя к последнему и корень соответствующего выхода  $s_k$  обозначается как  $z_{\langle \text{номер\_нейрона } k \rangle}$ . В ряде случаев величины, обозначающие корни выходов нейронов, в конечное выражение не попадают — они присутствуют там лишь в том случае, если выражение, представленное  $z$ -величиной, невозможно (с точки зрения логики ядра) раскрыть и подставить в другие выражения (это возможно в случае, если  $z$ -выражение присутствует в первой степени, то есть является квадратным корнем, который затруднительно раскрыть).

Все переменные, участвующие в генерируемых выражениях, дополнительно нумеруются по порядку (сначала входы сети, затем корни выходов  $s_k$  нейронов до входа в активационную функцию).

Рассмотрим теперь *базовый абстрактный класс* ITEM. Наиболее ценным методом в нем является функция clone, используемая для создания копии текущего объекта. Дело в том, что в разработанном ядре для экономии места в памяти используется концепция «разрушаемых» операндов: в любой операции, включающей два операнда, один из них (первый) становится результатом, а второй «разрушается» — элементы его данных непосредственно переносятся в первый операнд-результат. Поэтому, для случаев, когда одно и то же выражение является операндом в нескольких иных выражениях, возникает потребность в создании его копий, которые и передаются в соответствующие операции. Этим и объясняется появление виртуальной функции clone, переопределяемой в классах-потомках.

Класс SUMMAND представляет собой *абстрактное слагаемое*, содержащее маску var\_mask мультипликативного операнда (если данное слагаемое может быть представлено в такой форме) и три базовые метода, реализуемые любым видом слагаемого:

- Mul — умножение на элемент выражения,
- TryAdd — попытка прибавления к текущему слагаемому иного слагаемого (если в результате получается одно слагаемое, то данная функция возвращает true, иначе false),
- IsEqual — проверка на равенство текущего слагаемого иному слагаемому, передаваемому в качестве аргумента.

Класс MUL представляет *мультипликативное слагаемое*, являющееся произведением переменных в различных степенях, домноженным на коэффициент. Поле card представляет собой сумму степеней при переменных, входящих в слагаемое, и используется для быстрой проверки объектов класса MUL на неравенство. Поле vars является структурой класса vector, хранящей номера переменных, входящих в слагаемое. Поле rows также относится к классу vector и хранит показатели степеней соответствующих переменных, номера которых указаны в vars. Особо отметим, что коэффициент при слагаемом также хранится в этих двух последних полях: для этого введена псевдопеременная с номером -1, причем значение коэффициента помещается в rows.

Класс `DIV` представляет дробь, в числителе и знаменателе которой находятся суммы (объекты класса `SUM`). Соответственно, класс содержит поля `dividend` (числитель) и `divisor` (знаменатель) класса `SUM`. Дополнительно класс содержит метод `Divide`, который пытается (настолько, насколько это возможно) разделить числитель дроби на знаменатель. Результатом является сумма, которая, обычно состоит из частного и остатка от деления в виде дроби.

Операция `Divide` очень трудозатратна. Это объясняется спецификой работы символического ядра. По мере комбинирования выражений каких-либо специальных упрощений (за исключением приведения подобных слагаемых) не проводится. В результате последовательных открытия скобок и преобразования дробей генерируется единственная дробь, которая может иметь несколько сотен элементов как в числителе, так и в знаменателе. Отсюда и высокая трудоемкость операции `Divide`, а, следовательно, и необходимость в ее распараллеливании с контролем времени счета, позволяющим гибко регулировать соотношение времени упрощения и качества упрощения. Более подробно эти моменты будут рассмотрены в следующих подразделах.

#### 4.4. Общее ускорение алгоритмов

С целью общего ускорения процесса символического комбинирования конечного выражения, являющегося упрощенной версией нейросетевой функции, применены:

а) бинарный поиск подобных слагаемых по равенству специальных битовых масок (класса `VarMask`), в которых кодируются степени входящих в слагаемое переменных. Маски выступают в роли ключа стандартной `map`-структуры, а в роли значений выступают сами слагаемые;

б) контроль времени поиска очередного члена делимого при делении полинома на полином.

Далее эти вопросы рассматриваются более подробно.

#### 4.4.1. Применение маски

Как уже указывалось выше, класс `VarMask` хранит битовую маску, хранящую в упакованной форме показатели степеней переменных, входящих в мультипликативное слагаемое. Было поставлено (непринципиальное) *ограничение* на сложность решаемой задачи (которое при необходимости может быть снято путем расширения маски) — использование в совокупности не более чем 20 переменных. Это объясняется тем, что по соображениям быстродействия объем маски был выбран в 128 бит [два элемента данных типа `long long` (64-битовое целое со знаком)], причем под обозначение целочисленной степени каждой переменных отводится 6 бит, из них один бит занимает знак, пять бит — собственно значение. Таким образом, в мультипликативном операнде допускаются степени при переменных от  $-(2^5 - 1)$  до  $+(2^5 - 1)$ . Учитывая, что в каждом слое активационная функция замещается более простой со степенями при переменных не выше четвертой, причем такое замещение имеет место только в первых двух слоях, максимальная степень любой переменной не превышает 16. Соответственно, пяти бит на представление таких степеней достаточно. Максимальная величина в 20 общих переменных пригодна, например, для описания сети с четырьмя входами и нейронной структурой  $7 \times 5 \times 1$ , что в совокупности дает 17 переменных. Как показано в [46], для построения адекватных локально-интерполяционных нейросетевых моделей турбулентной вязкости такой размерности сети вполне достаточно (как показывают наши исследования, возможно и успешное применение сетей даже меньшей размерности, например, со структурой  $4 \times 3 \times 1$  и с тремя входами).

Ввиду особой важности данного класса, приведем листинги его основных методов в версии для Microsoft Visual C++:

```
#define FULL64 0x3Fi64
#define ZERO64 0x0i64
#define MINUS64 0x20i64
inline void VarMask::set_mask(long double pow, int var) {
```

```

long long p = (long long) (pow >= 0.0 ? 0.5*pow : -
(0.5-pow));
long long sign = p < 0 ? MINUS64 : ZERO64;
long long mask = ((abs(p) | sign)<<(6*(var%10)));
if (var >= 10) {
    var_mask[1] |= mask;
} else {
    var_mask[0] |= mask;
}
};

inline long long VarMask::is_mask(int var) {
    return var_mask[var/10] & (FULL64<<(6*(var%10)));
};

inline void VarMask::clear(int var) {
    var_mask[var/10] &= ~(FULL64<<(6*(var%10)));
};

```

Особенно подчеркнем, что текущая версия символического ядра работает только с целыми показателями степеней. При этом, заметим, упрощающие выражения (3.7) и (3.8) содержат и половинные степени. Данная проблема решается достаточно простым способом: половинная степень может появиться только в том случае, если выход  $S$  нейрона пропускается через соответствующую упрощенную функцию (3.7) или (3.8), замещающую активационную. При этом появляются как целые степени выражения  $S$ , которые могут быть незамедлительно раскрыты, так и дробные степени  $S^{A+0,5}$ , где  $A$  — целое число. Тогда последние выражения преобразуются к виду:

$$S^{A+0,5} = S^A \sqrt{S} = P \cdot Z,$$

где  $P$  — раскрытое выражение, представляющее  $S$  в степени  $A$ , а  $Z$  — соответствующая  $z$ -переменная (см. ранее), представляющая собой квадратный корень из выхода нейрона  $S$  до применения активационной функции. В ходе даль-

нейших упрощений эта  $z$ -переменная может быть возведена в четную степень и раскрыта по обычным алгебраическим формулам. Если же этого не происходит, то она попадает в конечное выражение без каких-либо изменений.

Необходимо пояснить еще один момент. В любую сумму помимо простых мультипликативных слагаемых могут входить дроби, которые тоже имеют маску `VarMask` (унаследованную от `SUMMAND`). Очевидно, что в таком случае в маске необходимо и достаточно хранить некое специальное значение, означающее, что текущее слагаемое представляет собой дробь. В качестве такого значения выбраны единицы, взятые с отрицательным знаком:

```
var_mask[0] = -1;
var_mask[1] = -1;
```

Именно с такой маской дроби и хранятся в поле `summands` класса `SUM`. Очевидно, что поскольку данное поле является ассоциативным массивом класса `map`, в сумму может входить единственная дробь с указанной маской. Если производится попытка добавить в сумму еще одну дробь, то производится ее сложение с дробью, которая уже присутствует в сумме, по элементарным алгебраическим правилам.

В качестве конкретных применений маски в разработанном ядре рассмотрим реализацию некоторых операций, относящихся к классу `SUM`, а также функции приведения подобных слагаемых.

Наиболее просто реализуется операция `SUM::Add(SUMMAND)`. Прежде всего, определяется, является ли операнд дробью (объектом класса `DIV`). Если это так, то данная дробь или складывается с уже присутствующей в ассоциативном массиве `summands` дробью по ключу `(-1, -1)` или просто помещается в него с указанным ключом. Если же операнд является мультипликативным (объект класса `MUL`), то производится проверка наличия в `summands` слагаемого с такой же маской (`VarMask`): если оно присутствует, то к его коэффициенту просто добавляется

коэффициент при добавляемом операнде; если его там нет, то он добавляется в *summands* с ключом, равным своей маске.

Необходимо заметить, что вышеприведенные операции поиска в ассоциативном массиве *summands* элемента по ключу-маске являются очень быстродействующими — используется бинарный поиск. Поэтому такая реализация хранения операндов по сравнению, например, с их хранением в обычном (не ассоциативном) массиве существенно повышает скорость поиска — максимально в

$$O\left(\frac{R}{\log_2 R}\right)$$

раз, где *R* — среднее количество операндов в суммах в упрощаемых выражениях.

Поговорим теперь о реализации операции *SUM::Add(SUM)*. Легко видеть, что данная операция сводится к поэлементному добавлению в текущую сумму слагаемых из суммы-операнда, причем приведение подобных членов происходит автоматически (по маске слагаемых), точно так же, как это реализовано в *SUM::Add(SUMMAND)*.

Функция, реализующая приведение подобных членов, достаточно проста по реализации. Это сложно рекурсивная функция<sup>1</sup>, поскольку рекурсия происходит через упрощающее приведение членов в различных составных элементах выражения (дробь или сумма, содержащая дробь). При этом все выражения, содержащие дробь, преобразуются в дроби, при необходимости эти дроби делятся или домножаются на прочие входящие в выражение дроби (по правилам элементарной алгебры), поэтому в результате упрощающего приведения подобных членов в сложном выражении часто получается *единственная дробь* с числителем и знаменателем, содержащими большое количество слагаемых. Фактически, слагаемые в получаемых суммах всегда уже приведены (алгоритм с применением маски, о котором уже говорилось ранее). Поэтому заключительным этапом здесь является обнаружение и исключение нулевых слагаемых по тривиальным и не рассматриваемым здесь алгоритмам.

---

<sup>1</sup> Также сложно рекурсивной является, например, функция *IsEqual*, проверяющая два объекта одного и того же класса на равенство.

Интерес представляет еще, вероятно, многоэтапная ускоренная проверка на равенство двух мультипликативных операндов `MUL::IsEqual(MUL)`. Перечислим эти этапы последовательной, построенной по принципу повышения «строгости» проверки:

- а) проверка равенства количеств переменных в операндах;
- б) проверка коэффициентов при операндах;
- б) проверка равенства полей `card` (см. выше);
- в) проверка на равенство масок `var_mask`.

#### 4.4.2. Новая схема контроля времени при делении полинома на полином

Рассмотрим операцию поиска очередного члена частного при делении полинома  $A = \sum_i A_i$  на полином  $B = \sum_j B_j$ . Точнее, рассмотрим один виток цикла по слагаемым делителя  $B_j$ , в котором определяется очередной член делимого  $A_i$ , такой, что в остатке от деления  $A - (A_i/B_j) \cdot B$  будет наименьшая сумма степеней переменных. При этом вводится эвристическое *предположение*, что она достигается при достаточно больших суммах степеней (с вариациями как в большую, так и в меньшую стороны) переменных в частном  $A_i/B_j$ , далее это предположение проверяется путем прямого вычисления вышеупомянутого остатка. Как уже упоминалось, деление полинома на полином, вероятно, наиболее трудозатратная операция из всех, реализуемых символическим ядром. Поэтому, помимо векторного распараллеливания вычислений (по виткам вышеупомянутого цикла), целесообразно применить дополнительные приемы контроля времени, позволяющие снизить собственно количество выполняемых операций. Этот контроль ведется индивидуально для каждого процессора, по окончании цикла результаты контроля обобщаются по принципу «выигрывает наиболее жесткая оценка».

В соответствии с вышеизложенным, для контроля времени поиска очередного члена делимого  $A_i$  при заданном  $B_j$  предлагается (чтобы не вычислять остаток  $A - (A_i/B_j) \cdot B$  для всех  $A_i$ ) сперва вычислить множество перебираемых ин-

дексов  $i \in V$ , где  $V \subseteq \{k \mid k = \overline{1, M}\}$ ,  $M$  — количество членов  $A_i$ , а затем (в отдельном цикле) вычислить остатки Величина множества  $V$  регулируется динамически — увеличивается, если время поиска меньше допустимого, или уменьшается в противном случае (ниже эта схема будет дана в более формальном виде). Предлагается следующая стратегия вычисления этого множества для каждого конкретного  $B_j$ :

1. Пусть  $V := \emptyset$ ,  $C := -\infty$ .

2. Цикл по  $i = \overline{1, M}$ .

2.1. Находим  $D := A_i / B_j = k \prod_{r=1}^{g_r} v_r^{g_r}$ , где  $k$  — коэффициент,  $v_r$  — переменная.

2.2. Находим  $F := \sum_{r=1}^{g_r} g_r$ .

2.3. Если  $F \in [C - d; C + d]$ , то  $V := V \cup \{i\}$ .

2.4. Иначе если  $F > C + d$ , то  $V := \{i\}$ ,  $C := F$ .

3. Конец цикла.

В описании стратегии (ее основного алгоритма) присутствует  $d$  — допустимое отклонение значения от лучшего из вариантов. В начале выполнения операции деления полинома на полином целесообразно установить величину  $d$  равной максимально возможной (чтобы не вводить априорных, возможно, излишне жестких ограничений) сумме степеней при переменных в членах делимого/делителя, которую мы обозначим как  $d_{\max}$ . Далее для каждого очередного  $B_j$  (на очередном витке цикла по слагаемым делителя) замеряется время  $t$  поиска требуемого  $A_i$ , которое сравнивается с допустимым  $t_{\text{доп}}$ . Если  $t > t_{\text{доп}}$ , то  $d := \max(0; d - 1)$ , если  $t < t_{\text{доп}}$ , то  $d := \min(d_{\max}; d + 1)$ . Новое значение  $d$  используется для следующего  $B_j$ . Заметим, что, если цикл по слагаемым делителя  $B_j$  распараллелен, то каждый процессор будет иметь *собственное значение*  $d$ . В таком случае, по окончании распараллеленного цикла используется *редукция*: в качестве нового значения  $d$  (для следующей стадии деления) выбирается минимальное из его локальных значений.

Такая схема позволяет получить оптимальное качество анализа при делении полинома на полином при приемлемом времени, затрачиваемом на данную операцию. Это подтверждается достаточно хорошим качеством получаемых упрощенных замещающих выражений при вполне допустимом времени решения.

#### 4.5. Новый алгоритм распараллеливания вычислений

Новый подход к распараллеливанию вычислений в разработанном символическом ядре состоит в комбинированном применении параллелизма «портфель задач» в сочетании с векторным параллелизмом. При этом порождается множество задач (вернее, подзадач), каждая из которых заключается в упрощении одного из вариантов комбинации заместительных (по отношению к классической нейросетевой) функций. В пределах каждой подзадачи дополнительно (векторно) распараллеливается (с контролем времени) поиск очередного члена частного при делении полинома  $A = \sum_i A_i$  на полином  $B = \sum_j B_j$  (распараллеливается цикл по слагаемому делителю  $B_j$ , см. выше). Основная идея состоит в том, чтобы выделить под каждую гранулу параллелизма в «портфеле задач» не одно ядро, а целую группу ядер (на которых и реализуется векторное распараллеливание). Делается предположение о том, что в условиях контроля времени такая схема распараллеливания даст лучшие временные показатели по сравнению с «чистыми» схемами «портфель задач» и «векторный параллелизм». Далее это предположение будет доказано как экспериментально так и аналитически.

##### 4.5.1. О порождении подзадач

Опишем в укрупненном виде алгоритм подзадачи:

1. Генерация первичного (не упрощенного) общего выражения и частных выражений для выходов нейронов (до активационных функций) путем разворачивания структуры сети с применением заместительных функций (3.5)-(3.8). Част-

ные выражения генерируются для тех нейронов, выходы которых в общей функции возводятся в *дробные степени*, и представляют собой квадратные корни из выходов нейронов (*z*-переменные).

2. Производятся последовательные подстановки (с упрощением) одних *z*-переменных в частные выражения, соответствующие другим *z*-переменным. При этом *z*-переменные в четных степенях сначала раскрываются, затем упрощаются и, только после этого подставляются. Далее производится промежуточное упрощение полученных выражений. Этот этап заканчивается сразу, как только заканчиваются возможности подстановок.

3. Производятся последовательные подстановки частных выражений, соответствующих *z*-переменным, в общее выражение. После каждой подстановки производится упрощение. Этот этап также завершается, когда исчерпываются возможности подстановок.

4. Помечаются *z*-переменные, которые остались в общем выражении.

5. Генерируется конечное символическое выражение (упрощенная нейросетевая функция), которое сопровождаются символические выражения для *z*-переменных, оставшихся в нем.

Порождение подзадач производится в рекурсивной функции BUILD\_FUNC, перебирающей варианты комбинаций заместительных функций. В связи с наличием рекурсии наиболее правильным вариантом порождения гранулы параллелизма для подзадачи является директива порождения подзадачи (если присутствует OpenMP версии не ниже 3.0, в противном случае подзадачи считаются последовательно, по мере порождения, а распараллеливание ограничится применением векторного параллелизма, упомянутого ранее):

```
#pragma omp task if((int)(NPP*TASK_PART) > 1) untied
```

Здесь NPP — доступное количество ядер, TASK\_PART — коэффициент «задачности», который равен нулю при чисто векторном распараллеливании и единице при исключительном распараллеливании по принципу «портфель задач».

Далее приводится упрощенный схематический проект кода, который запускает процесс перебора вариантов.

```

#pragma omp parallel if((int)(NPP*TASK_PART) > 1)
num_threads((int)(NPP*TASK_PART))
{
    #pragma omp single
    {
        BUILD_FUNC( аргументы );
        if ((int)(NPP*TASK_PART) > 1) {
            #pragma omp taskwait
        }
    }
}

```

#### 4.5.2. Векторное распараллеливания при делении полинома на полином

Как уже было указано выше, при поиске очередного члена частного использовано векторное распараллеливание по виткам цикла с контролем времени. Такой комбинированный вариант распараллеливания потребовал гибкого разделения множества процессоров на несколько подмножеств, каждому из которых соответствует одна подзадача, а уже в пределах подмножества применен векторный параллелизм. Разделением управляет упомянутая выше переменная `TASK_PART`, значение которой задается исследователем. При векторном распараллеливании цикла использовалась следующая директива:

```

#pragma omp parallel for schedule(guided) if(NPP-(int)(NPP*TASK_PART) >
1) num_threads(NPP-(int)(NPP*TASK_PART))

```

Особенное внимание необходимо обратить на явно заданный тип расписания распараллеливания цикла — `guided`. Согласно нашим данным, полученным с помощью простых экспериментальных программ, данное расписание является наиболее эффективным, поскольку:

а) является динамическим, соответственно, позволяет более равномерно распределить нагрузку по ядрам в отличие от статических расписаний;

б) гибко регулирует размер чанка итераций на процессор, начиная с достаточно больших размеров и заканчивая маленькими размерами. Это позволяет существенно снизить непроизводительные затраты на распределение итераций по ядрам, характерное, например, для расписания `dynamic`, снизив количество таких чанков. При этом, за счет постепенного уменьшения размеров чанков, сохраняются все преимущества динамического распределения нагрузки.

#### **4.5.3. Экспериментальное исследование схемы распараллеливания**

Описанная выше схема распараллеливания позволила достаточно равномерно загрузить процессорные ядра (при наличии контроля времени решения) и добиться более высокой степени ускорения. В частности, для задач маленькой размерности, если принять за единицу время расчета при использовании только векторного параллелизма, то время счета (на стандартной 16-ядерной машины платформы Google's Compute Engine) при использовании только «портфеля задач» составляет уже 0,96, а при комбинированном варианте распараллеливания — от 0,78 до 0,81.

Однако нам более интересны результаты для задач относительно большой размерности. Была проведена (на той же машине) серия экспериментов для сети с тремя входами и структурой  $4 \times 3 \times 1$ , варьировалось значение `TASK_PART`. Обработывались 32 варианта комбинаций заместительных функций. Результаты экспериментов помещены в таблицу 4.2.

Таблица 4.2. Время счета для эксперимента большой размерности при различном TASK\_PART

| Значение<br>TASK_PART                    | Время счета,<br>с |
|--|-------------------|
| 0 (векторное<br>распараллеливание)       | 310,43            |
| 0,2                                      | 227,33            |
| 0,4                                      | 156,64            |
| 0,5                                      | 186,42            |
| 0,6                                      | 165,44            |
| 0,8                                      | 210,16            |
| 1 (распараллеливание<br>только подзадач) | 468,39            |

Здесь, если принять за единицу время расчета при использовании только векторного параллелизма, то время счета при использовании только «портфеля задач» составляет уже 1,5, а при комбинированном варианте распараллеливания — от 0,5 до 0,73. Из приведенных данных очевидно, что предложенная комбинированная схема распараллеливания (при промежуточных значениях TASK\_PART) дает лучшие временные показатели по сравнению с «чистыми» схемами «портфель задач» и «векторный параллелизм». При этом полученная временная зависимость от TASK\_PART весьма интересна и требует дополнительного аналитического рассмотрения/обоснования<sup>1</sup>.

---

<sup>1</sup> Автор выражает признательность доценту Е.И.Ляпунову, обратившему внимание автора на нестандартный характер полученных данных и посоветовавшему дать им аналитическое объяснение.

#### 4.5.4. Аналитическое обоснование временной зависимости от доли параллелизма «портфеля задач»

Введем предположение о том, что можно построить модель временной зависимости, считая, что все гранулы имеют некоторые одинаковые средние объемные и временные характеристики. Тогда время счета  $T(\alpha)$  на  $N_{\text{я}}$  ядрах с долей  $\alpha$  параллелизма «портфеля задач» для  $N_{\text{гр}}$  гранул составит

$$T(\alpha) = t_0 + \left\lceil \frac{N_{\text{гр}}}{\max(1, \lfloor \alpha N_{\text{я}} \rfloor)} \right\rceil t_{\text{гр}},$$

где  $t_0$  — некоторые постоянные временные издержки на запуск «портфеля задач», а время исполнения  $t_{\text{гр}}$  одной векторно распараллеленной гранулы в «портфеле задач» включает три основных члена:

$$t_{\text{гр}} = t_{\text{посл}} + t_{\text{изд}} + t_{\text{пар}},$$

где  $t_{\text{посл}}$  — постоянное время исполнения последовательной части гранулы,  $t_{\text{изд}}$  — временные издержки на векторное распараллеливание,  $t_{\text{пар}}$  — время исполнения распараллеленной части гранулы. Предположим, что  $t_{\text{изд}}$  описывается линейной функцией

$$t_{\text{изд}} = t_{\text{вс}} + \beta \cdot \max(1, N_{\text{я}} - \lfloor \alpha N_{\text{я}} \rfloor),$$

где  $t_{\text{вс}}$  — постоянные издержки на запуск векторного распараллеливания,  $\beta$  — некоторая константа. Время исполнения распараллеленной части гранулы  $t_{\text{пар}}$  включает затраты на многократное исполнение распараллеленного цикла по слагаемым делителя, каждое исполнение включает  $K$  итераций на ядро,

$$K = \left\lceil \frac{N_{\text{делитель}}}{\max(1, N_{\text{я}} - \lfloor \alpha N_{\text{я}} \rfloor)} \right\rceil$$

Соответственно, неоднократно, по  $K$  раз исполняется внутренний цикл поиска требуемого члена делимого  $A_i$ , причем каждое такое исполнение имеет время, последовательное приближающееся (за счет контроля) к образцовой величине  $t_{\text{доп}}$ . Из детального анализа этого алгоритма следует, что

$$t_{\text{пар}} \propto t_1 \sum_{p=\omega}^{f(K)} p + \omega t_{\text{доп}},$$

$$\omega = \delta - \phi K,$$

где  $t_1$  — константа,  $\delta$  и  $\phi$  — константы,  $f(K)$  — некоторая линейная функция. Отсюда следует, что

$$t_{\text{пар}} = c_1 + \frac{c_2}{\max(1, N_{\text{Я}} - \lfloor \alpha N_{\text{Я}} \rfloor)} + \frac{c_3}{\max(1, N_{\text{Я}} - \lfloor \alpha N_{\text{Я}} \rfloor)^2},$$

где  $c_1, c_2, c_3$  — некоторые константы.

Подводя итоги, можно сделать вывод об одном варианте зависимости  $T(\alpha)$ :

$$T(\alpha) = a + \frac{1}{\max(1, \lfloor \alpha N_{\text{Я}} \rfloor)} \left( b + \frac{c}{\max(1, N_{\text{Я}} - \lfloor \alpha N_{\text{Я}} \rfloor)} + \frac{d}{\max(1, N_{\text{Я}} - \lfloor \alpha N_{\text{Я}} \rfloor)^2} \right),$$

где  $a, b, c, d$  — константные параметры, которые требуют подбора. Такой подбор был проведен (методом наименьших квадратов) по данным таблицы 4.2. Было показано хорошее согласование вышеприведенной аналитической модели с этими данными при  $a = 62,95$ ,  $b = -288,98$ ,  $c = 8704$ ,  $d = -1928$ . Соответствующие результаты показаны на рис. 4.2. Примечательно, что модель хорошо передала положение минимальных значений  $T(\alpha)$ , что говорит о возможности применения подобных моделей для подбора оптимального значения TASK\_PART.

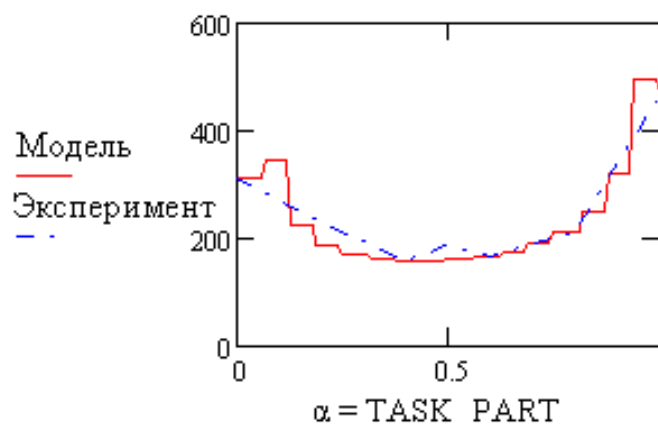


Рис. 4.2. Время счета  $T(\alpha)$ , согласно модели (сплошная линия) и по результатам замеров (штрих-пунктир)

#### 4.6. Краткое описание разработанного программного кода

Разработано простое быстродействующее ядро для символических преобразований, написанное на C++, объемом около 2500 строк.

В зависимости от определяемых условных символов, ядро может компилироваться для платформы Windows или Unix (Linux). Такого рода организация программного кода соответствует фактическим стандартам разработки многоплатформенных прикладных программ.

Программа может работать на произвольном количестве ядер. Тестирование проводилось в следующих конфигурациях:

1. Win32 – консольное приложение, одно или несколько ядер.
2. Linux – приложение, одно или несколько ядер.

На вход ядра подается текстовый файл, описывающий структуру нейронной сети и обучающую выборку. Кроме того, задается (через командную строку) количество используемых ядер, максимальное количество анализируемых вариантов упрощения и доля ядер, используемых для обработки подзадач (проработки вариантов упрощения).

Результатом работы ядра является набор выходных строк, представляющих выражения, в совокупности дающие упрощенный замещающий вариант общей нейросетевой функции.

#### 4.7. Анализ эффективности предложенных алгоритмов

Эффективность распараллеливания  $Q = Q(N_{\text{я}})$  определяется по формуле:

$$Q(N_{\text{я}}) = \frac{S(N_{\text{я}})}{N_{\text{я}}}, \quad (4.1)$$

$$S(N_{\text{я}}) = \frac{t_1}{t_{N_{\text{я}}}}, \quad (4.2)$$

где  $S = S(N_{\text{я}})$  — ускорение;  $t_1$  — время исполнения на одном ядре;  $t_{N_{\text{я}}}$  — время исполнения на  $N_{\text{я}}$  ядрах.

В таблице 4.3 приведены данные, полученные при использовании различного количества ядер стандартной 16-ядерной машины платформы Google's Compute Engine. Произведены расчеты ускорения вычислений (см. рис. 4.3) и эффективности распараллеливания (см. рис. 4.4).

Таблица 4.3. Данные о показателях распараллеливания в эксперименте большой размерности

| Количество ядер | Показатели распараллеливания |                                 |
|-----------------|------------------------------|---------------------------------|
|                 | Ускорение $S(N_{\text{я}})$  | Эффективность $Q(N_{\text{я}})$ |
| 1               | 1                            | 1                               |
| 2               | 1,37                         | 0,68                            |
| 3               | 1,66                         | 0,55                            |
| 4               | 2,43                         | 0,61                            |
| 5               | 2,85                         | 0,57                            |
| 6               | 2,94                         | 0,49                            |
| 7               | 3,11                         | 0,45                            |
| 8               | 3,91                         | 0,49                            |
| 9               | 3,56                         | 0,4                             |
| 10              | 3,72                         | 0,37                            |
| 11              | 3,9                          | 0,35                            |
| 12              | 4,04                         | 0,34                            |
| 13              | 4,4                          | 0,34                            |
| 14              | 4,88                         | 0,35                            |
| 15              | 4,3                          | 0,29                            |
| 16              | 4,6                          | 0,29                            |

Учитывая, что использовались не реальные, а *HyperThreading*-ядра, получены весьма высокие результаты.

Подводя итоги данного пункта, следует признать, что

а) распараллеливание оказалось оправданным при использовании до 14 ядер (это очевидно по данным о времени счета из таблицы 4.2 и об ускорении расчета из таблицы 4.3), дальнейшее увеличение количества ядер положительной динамики не дало;

б) предложенная схема распараллеливания доказала свою эффективность, получены существенные положительные оценки ускорения и эффективности распараллеливания.

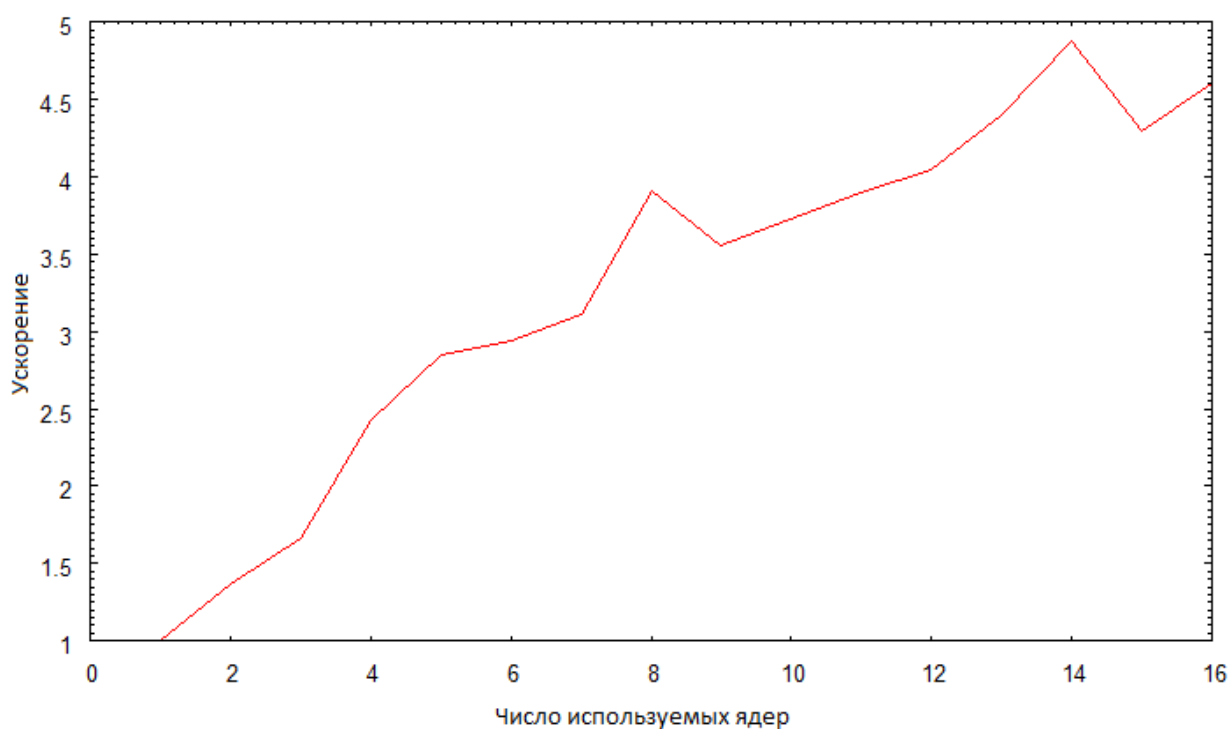


Рис. 4.3. Ускорение в зависимости от количества используемых ядер

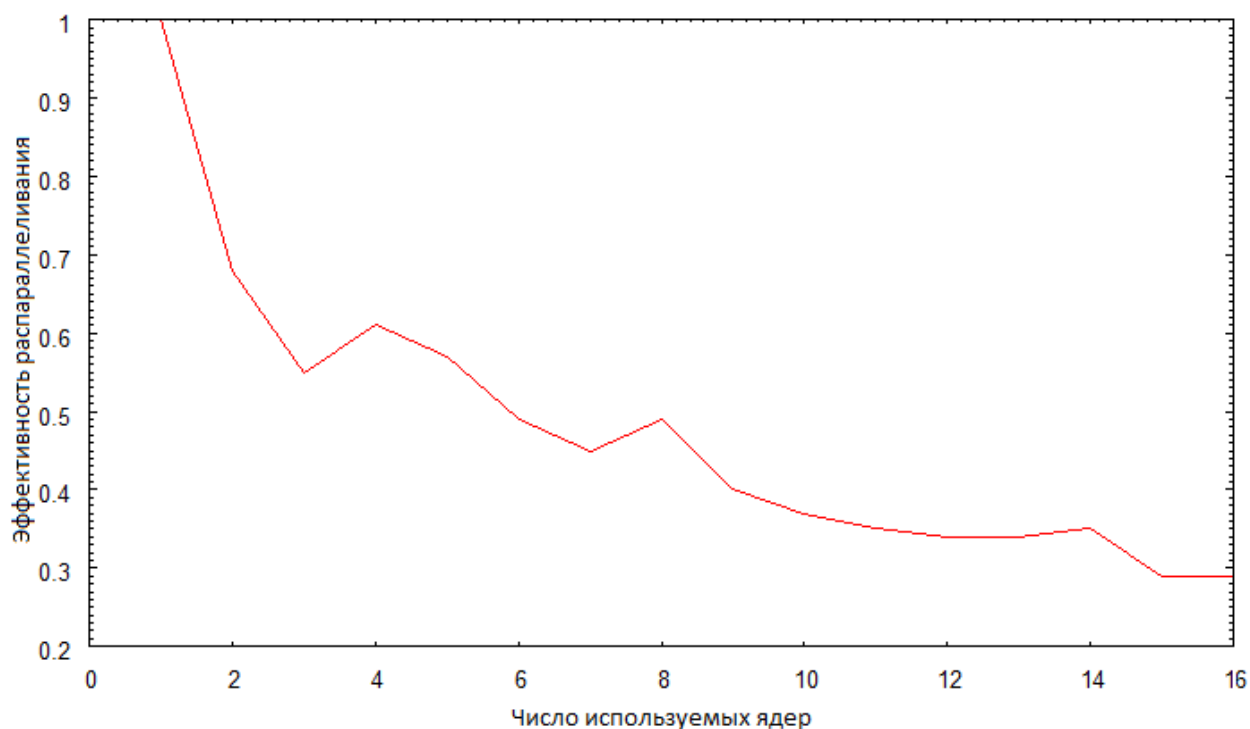


Рис. 4.4. Эффективность распараллеливания в зависимости от количества используемых ядер

### Выводы к четвертой главе

В данной главе рассмотрены вопросы символического комбинирования полученных для нейронной сети приближенных выражений и общего распараллеливания вычислений. Делается краткий обзор современных пакетов программ, выполняющих упрощающие символические преобразования выражений. Сделан вывод о необходимости разработки собственного ядра символических преобразований. Описывается разработанное простое и быстродействующее параллельное символическое ядро. Дана схема иерархии классов ядра. Описаны внутренние оптимизации, преимущественно на основе использования специальной маски мультипликативных операндов. Предлагается новая стратегия упрощения выражений при делении полинома на полином, подразумевающая контроль времени поиска очередного члена делимого, которая позволяет обеспечить достаточный баланс между качеством упрощения и его скоростью.

Проведен обзор основных современных подходов к распараллеливанию вычислений для переборных задач средней сложности на многоядерных одноплатформенных системах. По критериям легкости и очевидности распараллеливания, по важному для параллельных компьютеров требованию кроссплатформенности конечной программы, а также в связи с отсутствием выражено когерентных векторизуемых фрагментов программы, выбрано распараллеливание с применением технологии OpenMP. Описывается общая схема параллельного решения. Предложены эффективные параллельные алгоритмы, в которых используется новый подход к распараллеливанию задач рассматриваемого типа: комбинация идеологии «портфель задач» с внутренним дополнительным векторным распараллеливанием наиболее трудозатратной операции поиска очередного члена частного. Экспериментально показано, что такая схема распараллеливания дает более оптимальные результаты по сравнению с «чистыми» схемами «портфель задач» и «векторное распараллеливание».

Показано, что предложенный подход к распараллеливанию позволяет более полно и равномерно загрузить вычислительные ядра системы, соответственно, получить более высокую эффективность распараллеливания и меньшее время решения. На 16-ядерной машине платформы Google's Compute Engine ускорение составило до 4,9 раза, эффективность распараллеливания до 68%. Учитывая, что использовались не реальные, а HyperThreading-ядра, это весьма высокий результат.

Разработанный программный код, реализующий упрощение нейросетевых моделей с единственным выходом, адекватно работает как на машинах с одноядерным процессором, так и в многоядерных системах. Код оптимизирован для исполнения на Windows- и Linux-системах.

## Глава 5. Апробация предложенных подходов. Компактизация полей физических величин

Целью данной главы является апробация предложенных в главах 3 и 4 данной работы подходов. Поставим следующие задачи: а) апробация разработанного символического ядра путем генерации упрощенных символьных выражений для разных исходных сетей; б) апробация полученных упрощенных моделей турбулентной вязкости путем сравнения результатов экспериментов с исходной моделью (К-Е (RNG)) и с нейросетевыми (исходной и упрощенной) моделями; в) выработка рекомендаций по применению результатов работы в САПР объектов городской застройки; г) в рамках выработанных рекомендаций предложить схему компактного описания и хранения рассчитанных полей физических величин с целью их дальнейшего применения на этапе анализа проектного решения в САПР.

### 5.1. Апробация символического ядра

Данные, полученные в результате эксперимента по моделированию распространения инертного загрязнителя на улицах города, были использованы для обучения трехслойной ( $3 \times 2 \times 1$  нейронов с активационной функцией «экспоненциальная сигмоида» в первых двух слоях и с линейным третьим слоем) нейронной сети с двумя входами [46]:

а) квадратом минимального расстояния  $L_{\min}^2$  до ближайшей твердой стенки;

б) оценкой турбулентной вязкости, рассчитываемой как  $L_{\min} |\bar{U}|$ , где  $\bar{U} = (\bar{U}_1, \bar{U}_2, \bar{U}_3)$  — вектор скорости течения;

Для рассчитанной нейронной сети были получены (с помощью разработанных в данной работе программных средств) упрощенные выражения. Приведем два примера полученных упрощенных выражений (соответственно,  $\nu_{\text{турб}}^1$  и  $\nu_{\text{турб}}^2$ ).

При этом для повышения компактности записи были введены *обозначения*: а —

квадрат расстояния до твердой стенки,  $b$  — оценка величины турбулентной вязкости. Входные и выходные данные (здесь и в последующих экспериментах) предварительно были интерполированы к интервалу  $[0; 1]$ .

$$v_{\text{турб}}^1 = \frac{1,027 - 0,163a - 0,0034a^2 + 9,29b + 0,26ab - 6,35b^2 - 0,62z}{1,988 - 0,14a - 0,003a^2 + 8b + 0,224ab - 5,47b^2 - 0,53z - 6,49a + 2,04a^2 - 0,099b - 0,0006ab + 0,049b^2 - 0,012z},$$

$$v_{\text{турб}}^2 = 0,431 - 0,028a - 0,00056a^2 + 1,48b + 0,044ab - 1,03b^2 - 0,12z,$$

$$z = \sqrt{0,66 + 1,27a - 0,61b}.$$

Здесь при анализе использовались заместительные функции вплоть до третьего порядка ( $1 \leq P \leq 3$ ). Регулировка толерантности осуществлялась при

$$\varepsilon_{\text{мин}} = 10^{-3},$$

$$\alpha = 4 \cdot 10^{-2}.$$

Легко подсчитать, что для расчета  $v_{\text{турб}}^1$  и  $v_{\text{турб}}^2$  требуется не менее 39 и 17 одноктактных команд современных процессоров архитектуры x86, тогда как для расчета двухвходовой нейронной сети конфигурации  $3 \times 2 \times 1$  (с активационной функцией «экспоненциальная сигмоида» в первых двух слоях) требуется не менее 56 одноктактных команд (для вычисления каждой экспоненты требуется не менее трех команд, а в реальности — еще больше). Таким образом, теоретический выигрыш в скорости расчета на 30÷70% (а реальнее, по нашим оценкам — на 40÷50%) был достаточно очевиден. Однако если для первичной апробации символического ядра такого эксперимента вполне достаточно, то для получения реальных значений коэффициента повышения скорости расчета, а также для определения погрешности упрощенной модели по сравнению с исходной нейросетевой, целесообразно провести дополнительные эксперименты с более сложными сетями. Для таких экспериментов была выбрана более простая задача о двумерном обтекании одиночного препятствия.

## 5.2. Апробация полученных упрощенных моделей

Данные, полученные в результате эксперимента по моделированию двумерного обтекания одиночного здания, были использованы для обучения трехслойной ( $4 \times 3 \times 1$  нейронов с активационной функцией «экспоненциальная сигмоида» в первых двух слоях и с линейным третьим слоем) нейронной сети с тремя входами [46]:

а) квадратом минимального расстояния  $L_{\min}^2$  до ближайшей твердой стенки;

б) оценкой турбулентной вязкости, рассчитываемой как  $L_{\min} |\bar{U}|$ , где  $\bar{U} = (\bar{U}_1, \bar{U}_2)$  — вектор скорости течения;

в) функцией деформации  $\sqrt{\sum_{i=1}^2 \sum_{j=1}^2 \frac{\partial U_i}{\partial x_j} \left( \frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right)}$ .

Входные и выходные данные предварительно были интерполированы к интервалу  $[0; 1]$ . Была получена нейронная сеть, описывающая закономерности возникновения и распределения турбулентной вязкости. В следующих таблицах 5.1, 5.2 и 5.3 приведены данные о структуре полученной сети. Обозначения — согласно формулировке общей структуры сети из главы 3.

Таблица 5.1. Информация о весах и смещениях сети в первом слое (4 нейрона)

| Коэффициент | Номер нейрона в первом слое (k) |           |           |           |
|-------------|---------------------------------|-----------|-----------|-----------|
|             | 1                               | 2         | 3         | 4         |
| $w_{11k}$   | -0.225408                       | 0.793240  | -3.016402 | 1.190960  |
| $w_{12k}$   | 2.061399                        | 0.965149  | 3.009361  | 6.566509  |
| $w_{13k}$   | 0.918452                        | 0.564151  | 3.291517  | -1.185477 |
| $b_{1k}$    | -0.609413                       | -0.016302 | 0.093253  | -0.739285 |

Таблица 5.2. Информация о весах и смещениях сети во втором слое (3 нейрона)

| Коэффициент | Номер нейрона во втором слое (k) |          |           |
|-------------|----------------------------------|----------|-----------|
|             | 1                                | 2        | 3         |
| $w_{21k}$   | 0.986223                         | 0.814979 | 1.302509  |
| $w_{22k}$   | 0.791579                         | 0.618735 | 0.300453  |
| $w_{23k}$   | 0.932081                         | 0.557496 | 3.322023  |
| $w_{24k}$   | 0.132056                         | 0.921287 | 5.135825  |
| $b_{2k}$    | -0.294248                        | 0.628936 | -2.489575 |

Таблица 5.3. Информация о весах и смещениях сети в третьем слое (1 нейрон)

| Коэффициент | Значение  |
|-------------|-----------|
| $w_{311}$   | 0.656906  |
| $w_{321}$   | -0.482293 |
| $w_{331}$   | 3.433981  |
| $b_{31}$    | -2.738151 |

Для рассчитанной нейронной сети были получены (с помощью разработанных в данной работе программных средств) упрощенные выражения, среди которых было выбрано одно из самых простых — далее приведены соответствующие символьные выражения на языке C++, в которых были введены *обозначения*:  $a$  — квадрат расстояния до твердой стенки,  $b$  — оценка величины турбулентной вязкости,  $c$  — значение функции деформации:

```
double a2 = a*a;
double b2 = b*b;
double c2 = c*c;
double a3 = a2*a;
double b3 = b2*b;
double c3 = c2*c;
```

```

double z01 = sqrt(-1.6302e-02+7.9324e-01*a+9.65149e-
01*b+5.64151e-01*c);
double z06 = sqrt(1.583708e+00-8.907654e-01*a-
2.844379e-01*a2+1.050942e+00*a3+1.178096e+01*b-
1.865618e+00*b*a-3.145465e+00*b*a2-
5.777153e+00*b2+3.138123e+00*b2*a-
1.043599e+00*b3+1.32778e+00*c+5.849582e-01*c*a-
3.440382e+00*c*a2+1.838371e+00*c*b+6.864703e+00*c*b*a-
3.42434e+00*c*b2-3.013361e-01*c2+3.754167e+00*c2*a-
3.745404e+00*c2*b-1.365524e+00*c3+1.014237e-01*z01);
double vтипб = (-1.477593e+00)/(z06)+1.370237e+00-
6.303167e-02*a-1.495051e-03*a2+2.33748e-02*a3+5.558024e-
02*b+1.178189e-02*b*a-6.996071e-02*b*a2+1.837937e-
02*b2+6.97974e-02*b2*a-2.321149e-02*b3+7.924241e-
02*c+3.392287e-03*c*a-7.652018e-02*c*a2-1.214263e-
02*c*b+1.526831e-01*c*b*a-7.616337e-02*c*b2-1.915279e-
03*c2+8.349931e-02*c2*a-8.33044e-02*c2*b-3.037166e-
02*c3+1.949161e-02*z01;

```

В таблице 5.4 содержится информация о том, заместительные функции какого вида были использованы при генерации вышеуказанного выражения.

Таблица 5.4. Заместительные функции для упрощаемого варианта нейронной сети

| Слой | Номер нейрона в слое |       |       |       |
|------|----------------------|-------|-------|-------|
|      | 1                    | 2     | 3     | 4     |
| 1    | (3.5)                | (3.7) | (3.5) | (3.5) |
| 2    | (3.5)                | (3.5) | (3.8) |       |
| 3    | —                    |       |       |       |

Из таблицы 5.4 достаточно хорошо видна работа *стратегии получения оптимальной комбинации заместительных функций*. В самом деле, в первом слое присутствуют исключительно прямые заместительные функции (3.5) и (3.7), по-

этому стратегия во втором слое повысила вероятность выбора обратной функции (3.8). В результате полученное упрощенное выражение содержит не очень высокие степени переменных (максимальная суммарная степень переменных мультипликативного операнда *не превышает трех*). Это свидетельствует об *оправданности* разработки и применения в данной работе вышеуказанной стратегии, ее практической ценности.

Далее, с целью проверки адекватности были проведены два численных эксперимента в тех же условиях, в которых проводился эксперимент, результаты которого показаны на рис. 2.3, б (задавалось модельное время 40 секунд).

Различие между экспериментами состояло лишь в том, что в первом вместо К-Е (RNG) были применены полученные нами исходные нейросетевые соотношения, а во втором — упрощенные выражения. Средняя относительная погрешность расчета при замене модели К-Е (RNG) упрощенными соотношениями составила около 23%. Таким образом, *дополнительная погрешность* по сравнению с исходной нейронной сетью (20÷22%) составила 1÷3%. Это достаточно хороший результат, косвенно подтверждающих тождественность конечных упрощенных выражений, полученных в результате преобразований разработанным символическим ядром, исходным выражениям, полученным сразу после замены активационных функций упрощенными. Соответственно, это *гарантирует корректность проводимых символическим ядром преобразований*.

*Адекватность* полученных в главах 3 и 4 результатов, корректность использованных методов, алгоритмов и программ также подтверждается данными, приведенными на рис. 5.1, где показаны значения турбулентной вязкости, полученные в экспериментах с нейронной сетью (а) и с упрощенными выражениями (б). Хорошо видно, что значения согласуются как качественно, так и количественно.

Интересно заметить, что погрешность полученной упрощенной нейросетевой модели оказалась меньше, чем у однопараметрической модели турбулентности Абрамовича-Секундова, где средняя относительная погрешность имела величину около 98%. Таким образом, используемый нами класс моделей показывает

*промежуточные результаты между однопараметрическими и двухпараметрическими моделями турбулентности. Это важный практический результат, подтверждающий практическую ценность этой и иных работ в данном направлении.*

Помимо погрешности, замерялось (в сокращенном эксперименте с модельным временем в 10 секунд) время расчета при использовании исходной нейронной сети  $T_1$  и упрощенных выражений  $T_2$ . Также определялось собственно суммарное время, затраченное на вычисление отклика сети  $T_{1\text{чист}}$  и на вычисление упрощенных выражений  $T_{2\text{чист}}$ . Были получены следующие значения:

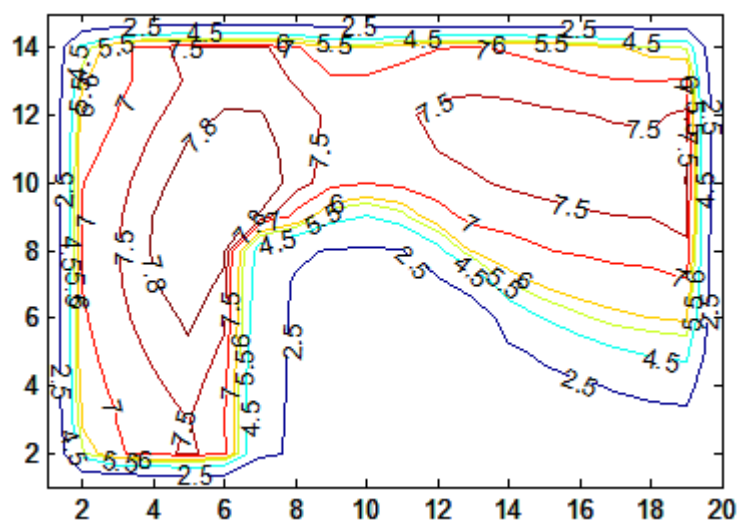
$$T_1 = 286,23 \text{ с};$$

$$T_2 = 267,98 \text{ с};$$

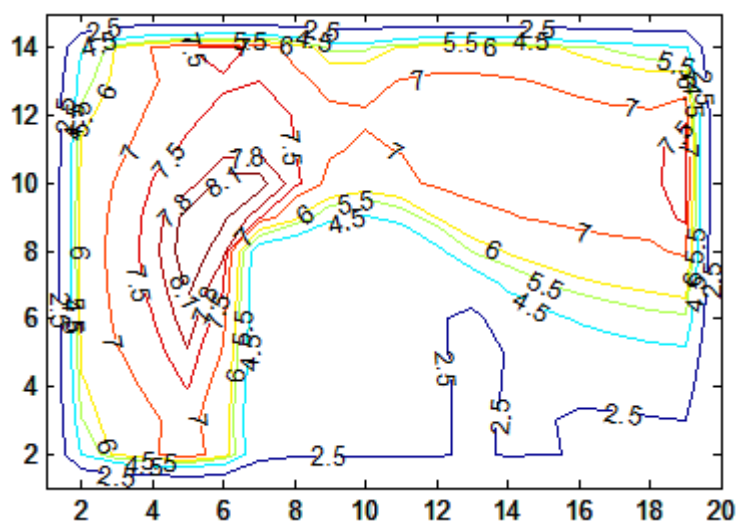
$$T_{1\text{чист}} = 38,16 \text{ с};$$

$$T_{2\text{чист}} = 20,91 \text{ с}.$$

Таким образом, упрощенные соотношения обеспечивают общее ускорение (при расчете системы из 3÷5 дифференциальных уравнений в частных производных) в 1,07 (реальнее в 1,06÷1,08) раза (на 6÷7%), ускорение расчета исключительно турбулентной вязкости составляет около 1,8 раза (на 45%), что вполне согласуется с приведенными ранее теоретическими выкладками для более простых сетей (40÷50%). Вероятно, это и есть *средний показатель эффективности предложенных в главах 3 и 4 приемов.*



а)



б)

Рис. 5.1. Распределения турбулентной вязкости, полученные при использовании нейросетевой (а) и упрощенной нейросетевой (б) моделей

### 5.3. Краткие рекомендации по применению результатов работы в САПР объектов городской застройки

Полученные упрощенные модели турбулентной вязкости могут использоваться для ускорения моделирования воздушных потоков на улицах города (для определения ветровой нагрузки и/или концентраций загрязнителей, выделяемых автотранспортом/предприятиями) при анализе очередного полученного проектного решения. Предполагается итерационная процедура уточнения проектного вари-

анта, при которой на каждой итерации запускается подсистема моделирования воздушных потоков и рассчитываются требуемые показатели, которые позволяют принять (полностью или частично) или окончательно отвергнуть текущий вариант. Это, вероятно, будет оптимизационная процедура, при которой генерируемые показатели входят в общую целевую функцию качества решения.

Оценки загрязненности и ветровой нагрузки, в частности, могут быть применены как при решении задачи поиска оптимального проектного варианта в целом, так и при решении более частных задач автоматизации проектирования. В САПР объектов городской застройки это задачи определения оптимальной плотности застройки, мест размещения и высот зданий и сооружений.

Не исключено косвенное применение моделей при синтезе решений: достаточно, например, в результате моделирования обтекания здания получить приближенные интерполяционные зависимости ветровой нагрузки на здание (определенной формы) от его габаритов, скорости и направления ветра. Тогда по известной наибольшей зарегистрированной скорости ветра легко подобрать такие габариты здания, при которых соответствующая нагрузка на него для всех возможных направлений ветра является допустимой.

В любом из указанных выше случаев разработанные в данной работе программы войдут в состав общей САПР в качестве подсистем.

Необходимо заметить, что вышеописанные варианты применения подсистемы моделирования воздушных потоков подразумевают ее многократный запуск с различными параметрами даже в рамках построения одного проекта. Такой запуск — достаточно затратная по времени процедура, поэтому, несомненно, актуальными задачами становятся: а) обеспечение возможности многократного применения полученных для некоторых стандартных случаев результатов моделирования; б) построение параметризованных приближенных моделей, которые, опираясь на уже рассчитанный набор стандартных случаев, позволяли бы быстро получить интерполированные результаты для некоторых нестандартных вариантов.

Решение первой задачи требует сохранения потенциально огромных объемов данных, в связи с чем остро встает вопрос об их компактизации. В самом де-

ле, при больших размерах расчетной сетки (в задачах расчета распространения загрязнителей — от нескольких сотен до миллионов узлов для областей сложной формы, включающих, например, несколько улиц со строениями разной высоты) объем сохраняемых конечных данных (полученного стационарного распределения воздушных потоков — трех полевых компонент вектора скорости и поля турбулентной вязкости) даже для одного случая при использовании вещественных значений одинарной точности (4-байтных) может достигать нескольких десятков мегабайт. Учитывая, что число случаев может определяться, например, произведением количества возможных направлений ветра ( $15 \div 30$ ) на количество градаций скорости ветра ( $10 \div 15$ ), его величина может составлять от 150 до 450. В результате общий объем сохраняемых данных может достичь одного гигабайта и выше. Как показывает практика, для этих данных максимальный коэффициент сжатия без потерь такими достаточно качественными алгоритмами как PPMD и LZMA не превышает  $0,50 \div 0,65$ . Для получения большей степени сжатия (например, с коэффициентом  $0,01 \div 0,02$ ) необходимо воспользоваться приближенными методами (сжатие с потерями).

Видимо, первую задачу целесообразно решать совместно со второй — достаточно построить некоторую компактную модель, интерполирующую результаты моделирования по значениям некоторым исходных параметров (например, по направлению и скорости ветра). Эта модель строилась бы по полученному набору результатов моделирования для стандартных случаев, позволяя приблизить их с относительно небольшой погрешностью, и, кроме того, позволяла бы генерировать приближенные интерполированные результаты для нестандартных вариантов. Рассмотрим данную проблему в следующем пункте.

#### **5.4. Компактизация полей физических величин в задачах аэродинамики**

Для генерации компактных описаний получаемых вариативных полей физических величин (компонент скорости и турбулентной вязкости) воспользуемся

интерполяцией данных нейронными сетями с сохранением их весовых коэффициентов. Применение нейронных сетей также решит проблему получения интерполированных значений физических величин, например, для случая таких направлений/скоростей ветра, для которых нет сохраненных (сжатых или несжатых) данных.

С вычислительной точки зрения задача сжатия может сводиться к кластеризации и обучению динамически определяемого набора нейронных сетей в каждом кластере. Это достаточно затратные процедуры, поэтому для их решения целесообразно использовать параллельные вычисления.

### 5.4.1. Кластеризация

Поля физических величин обычно характеризуются наличием множества достаточно компактных групп схожих (принадлежащих некоему сравнительно небольшому интервалу) значений, взятых в узловых точках расчетной сетки. Это приводит к идее, что первый и второй этапы описания вариативного поля  $V(x, y, z, a)$ , где  $a$  — некий варьируемый параметр (например, направление ветра), должны состоять в двухслойной кластеризации: сначала на метакластеры, по значениям  $Q(x_i, y_j, z_k) = f(V(x_i, y_j, z_k, a))$  в узловых точках  $(i, j, k)$ , затем, в пределах каждого из полученных метакластеров — на кластеры, по координатам. Можно попытаться описать поле в пределах каждого из полученных кластеров нейронной сетью прямого распространения. На входы сети подаются как значение параметра  $a$ , так и значения координат  $x_i, y_j, z_k$ , а выходом сети будет  $V(x_i, y_j, z_k, a)$  в узловой точке  $(i, j, k)$  при указанном значении  $a$ <sup>1</sup>.

Отметим, что поскольку первичная кластеризация должна выделять блоки с близкими значениями  $V(x, y, z, a)$  по всем возможным значениям  $a$ , то  $Q(x, y, z)$

---

<sup>1</sup> Заметим, что мы сознательно отказались от варианта, при котором сеть сразу же пытается предсказать все  $N_a$  значений: в таком случае вход для  $a$  отсутствует, но появляется  $N_a$  выходов, каждый  $p$ -й из которых вы-

должна являться некоей сверткой значений:  $Q(x, y, z) = \sum_{p=1}^{N_a} w_p V(x, y, z, a_p)$ , где  $N_a$

— количество возможных различных значений  $a$ ,  $w_p$  — относительная значимость  $p$ -го значения  $a$  в общем спектре значений этого параметра. Целесообразно, например, давать бóльшие значения  $w_p$  для таких  $a_p$ , при которых вероятны повышенные ошибки интерполяции нейронной сетью. Такие значения могут быть назначены, исходя из соображений исследователя, или определены, например, путем простого эксперимента по попытке обучения  $N_a$  нейронных сетей, каждая  $p$ -я из которых ( $p = \overline{1, N_a}$ ) приближает все поле  $V(x, y, z, a_p)$  при фиксированном  $a_p$ , принимая на входы *все* узловые точки области [46]. По полученным средним погрешностям приближения  $\varepsilon_p$  исходных данных  $p$ -й сетью и выставляются веса:

$$w_p = w_{\min} + \frac{w_{\max} - w_{\min}}{\sum_{r=1}^{N_a} \varepsilon_r} \varepsilon_p,$$

где  $w_{\min}$  и  $w_{\max}$  — соответственно минимальная и максимальная величины значимости.

#### 5.4.2. Построение нейросетевых описаний в кластерах

Третьим этапом описания вариативного поля  $V(x, y, z, a)$  в каждом кластере, как и указано выше, является его приближение нейронной сетью. При этом достаточно интересным является вопрос, следует ли воспользоваться одной более сложной сетью для всех значений  $a_p$ , или же разбить множество  $\{a_p\}$  на подмножества, в каждом из которых применить более простую нейронную сеть. Как показали эксперименты по описанию полей скорости воздушных потоков, возможны оба варианта, причем второй требует меньшего времени на обучение и дает лучшие результаты приблизительно в  $1,5 \div 2$  раза чаще, что будет продемонстрировано

---

дает значение  $V(x, y, z, a_p)$ . Такой вариант, по нашему мнению, не обеспечивает должного качества интерполяции при небольшом количестве нейронов и требует чрезвычайно большого времени на обучение.

далее, в экспериментальном разделе. Однако в каждом конкретном случае необходим контекстный выбор разбиения  $\{a_p\}$ , что приводит к идее организации *адаптивного алгоритма, который выбирает вариант интерполяции* в пределах кластера. При этом чтобы избежать полного перебора, целесообразно ограничиться лишь несколькими (3÷4) основными вариантами, а для повышения скорости отбора вариантов воспользоваться предикцией погрешности для некоторых из них (по уже полученным данным), с ее частичной коррекцией по данным, полученным путем ограниченного расчета нескольких компонент новых вариантов.

Предлагается следующий адаптивный эвристический алгоритм:

1. Воспользуемся трехслойными нейронными сетями с экспоненциальными сигмоидами в первых двух слоях и с линейной передаточной функцией в третьем выходном слое.

2. Определим начальное количество нейронов при интерполяции единственной нейронной сетью. Для каждого значения  $a_p$ ,  $p = \overline{1, N_a}$ , найдем среднее по кластеру значение  $q_p$  поля  $V(x, y, z, a_p)$ , где  $(x, y, z) \in S$ , причем  $S$  — множество точек текущего рассматриваемого кластера. Получаем вектор значений  $q = (\overline{q_1, q_2, \dots, q_{N_a}})$ , по которому можно определить набор точек  $(p, q_p)$ . По этому набору проводим интерполяцию полиномом достаточно высокого порядка  $r$ , где  $N_a/4 \leq r < N_a$ . Находим количество экстремумов  $N_e$  полученного полинома (достаточно разбить область определения полинома на малые интервалы и подсчитать суммарное количество интервалов, на краях которых производная полинома меняет знак или на одном из краев имеет нулевое значение).

Тогда начальное количество нейронов первого слоя

$$N_1 := \lfloor \max(N_{1\min}, \min(N_{1\max}, \alpha N_e)) \rfloor,$$

а второго слоя

$$N_2 := \lfloor \max(N_{2\min}, \min(N_{2\max}, \beta N_e)) \rfloor,$$

где  $N_{1\min}$ ,  $N_{1\max}$  — минимально и максимально допустимые количества нейронов первого слоя,  $\alpha$  — коэффициент,  $N_{2\min}$ ,  $N_{2\max}$  — минимально и максимально до-

пустимые количества нейронов второго слоя,  $\beta$  — коэффициент. В наших экспериментах использовались следующие значения:

$$N_{1\min} = 7, N_{2\min} = 3, N_{1\max} = 18, N_{2\max} = 10, \alpha = 3,3, \beta = 1,4.$$

3. Задаем начальное количество сетей  $A_1 := 1$ . Пусть  $r := 1$ .

4. Инициализируем массив ошибок  $\forall s : E_s := 0$ .

5. Если  $r > 2$ , то экстраполируем погрешность:

5.1. Пусть  $g(s) = \left( \sum_{m=0}^{r-1} a_m s^m \right)^{-1}$ . Задаем  $r-1$  точек вида  $(s; E_s), s = \overline{1, r-1}$ .

Проводим по этим точкам интерполяцию функцией  $g(s)$  методом наименьших квадратов, в результате определяем коэффициенты  $a_0 \dots a_{r-1}$ .

5.2. Определяем  $E_r := g(r)$ . Если  $E_r < 0$ , то  $E_r := 0$ .

5.3. Оцениваем производную  $D := (E_r - E_{r-1}) / (A_r - A_{r-1})$ .

5.4. Если  $D > 0$  или  $|D| < D_{\text{доп}}$  (здесь  $D_{\text{доп}}$  — минимально допустимое значение производной), то:

5.4.1. Предыдущее  $(r-1)$  приближение с  $A_r/2$  сетями признается наиболее успешным.

5.4.2. Выход из алгоритма.

6. Равномерно распределяем значения  $a_p, p = \overline{1, N_a}$ , по  $A_r$  нейронным сетям.

7. Цикл по  $q = \overline{1, A_r}$ :

7.1. Обучаем  $q$ -ю нейронную сеть по связанным с ней парам  $((\overline{x, y, z, a_p}), V(x, y, z, a_p))$  во всех узлах текущего кластера. Определяем среднюю погрешность  $E_{\text{avr}}$ .

7.2. Если  $q = 1$  и  $r > 2$ , то:

7.2.1. Корректируем предсказанную погрешность:  $\tilde{E} := 0,8E_{\text{avr}} + 0,2E_r$ .

7.2.2. Оцениваем скорректированное значение производной  $D := (\tilde{E} - E_{r-1}) / (A_r - A_{r-1})$ .

7.2.3. Если  $D > 0$  или  $|D| < D_{\text{доп}}$  то:

7.2.3.1.  $E_r := E_{avr}$ .

7.2.3.2. Предыдущее  $(r-1)$  приближение с  $A_r/2$  нейронными сетями является самым успешным.

7.2.3.3. Выход из алгоритма

7.3.  $E_r := E_r + E_{avr}$ .

7.4. Конец цикла по  $q$ .

8. Определяем среднюю погрешность:  $E_r := E_r/A_r$ .

9. Если  $E_r \leq E_{\text{треб}}$  (здесь  $E_{\text{треб}}$  — требуемое минимально допустимое значение погрешности), то переходим к пункту 15.

10.  $A_{r+1} := 2A_r$ .

11.  $r := r + 1$ .

12.  $N_1 := \max(N_{1\min}, \lfloor \gamma \cdot N_1 \rfloor)$ .

13.  $N_2 := \max(N_{2\min}, \lfloor \delta \cdot N_1 \rfloor)$ .

14. Если  $A_r \leq A_{\max}$ , то переходим к пункту 5.

15. Алгоритм завершен, разбиение с  $A_r/2$  сетями является лучшим.

В наших экспериментах использовались следующие значения:

$$D_{\text{доп}} = \frac{0,05}{4}, E_{\text{треб}} = 0,075, \gamma = 0,6, \delta = 0,4, A_{\max} = 8.$$

### 5.4.3. Экспериментальная часть

В качестве примера возьмем построение компактных кластерно-нейросетевых описаний полей скорости воздушных течений, полученных для одной из стандартных рассматриваемых в данной работе задач — в эксперименте по моделированию распространения инертного загрязнителя на улицах города. Напомним, что имеющиеся данные (исходные для построения описаний) представляли результаты моделирования [48, 49] с установившимися значениями полей скорости и турбулентной вязкости, полученные для 18 различных направлений ветра (от 20 до 360 градусов с шагом в 20°). Количество узлов расчетной сет-

ки составляло  $72 \times 86 \times 45 = 278640$ . Для хранения каждой из трех компонент полученных полей скорости использовались четырехбайтные значения одинарной точности. Соответственно, для хранения данных по каждой компоненте задействовались  $18 \times 72 \times 86 \times 45 \times 4 = 20062080$  байт, то есть около 19 Мб.

Для построения компактных описаний была разработана параллельная программа (М-скрипт) для Octave 3.8.4 (Linux-версия), исполнявшаяся на стандартной 16-ядерной машине платформы Google's Compute Engine.

На первом этапе использовался алгоритм k-средних (функция kmeans) для разбиения на 20 метакластеров. На втором этапе использовалось распараллеливание по метакластерам в соответствии с парадигмой «портфель задач» (с функцией pararrayfun): каждый метакластер на отдельном ядре тем же алгоритмом (k-средних) делился на 15 кластеров, таким образом, общее количество кластеров достигло 300. На третьем этапе, при обучении нейронных сетей методом обратного распространения ошибки также использовалось распараллеливание по принципу «портфель задач»: один метакластер — одна задача (одно ядро). При этом задавалось максимальное количество прогонов по обучающей выборке, равное 100. Кроме того, обучение прекращалось при достижении среднеквадратичной ошибки, равной 0,075.

Использовались все 16 ядер, время обработки каждой компоненты скорости занимало в среднем около 28 минут. Для сравнения: на одном ядре процессора Intel Atom 1,86 ГГц (Windows) время обработки одной компоненты составляло около 16,5 часов. С поправкой на различие тактовых частот процессоров (в Google's Compute Engine используются процессоры Xeon 2,6 ГГц, Linux) можно подсчитать реальные (объясняющиеся преимущественно эффектом распараллеливания) ускорение и эффективность распараллеливания. Они составляют, соответственно,  $\frac{16,5 \cdot 60 \cdot (1,86/2,6)}{28} = 25,3$  и  $\frac{16,5 \cdot 60 \cdot (1,86/2,6)}{28 \cdot 16} = 158\%$ . Такой «суперлинейный» эффект объясняется, видимо, двумя факторами:

а) большей эффективностью скомпилированного для Linux кода Octave по сравнению с кодом, полученным для Windows;

б) более высокой производительностью процессора Intel Xeon в сравнении с Intel Atom.

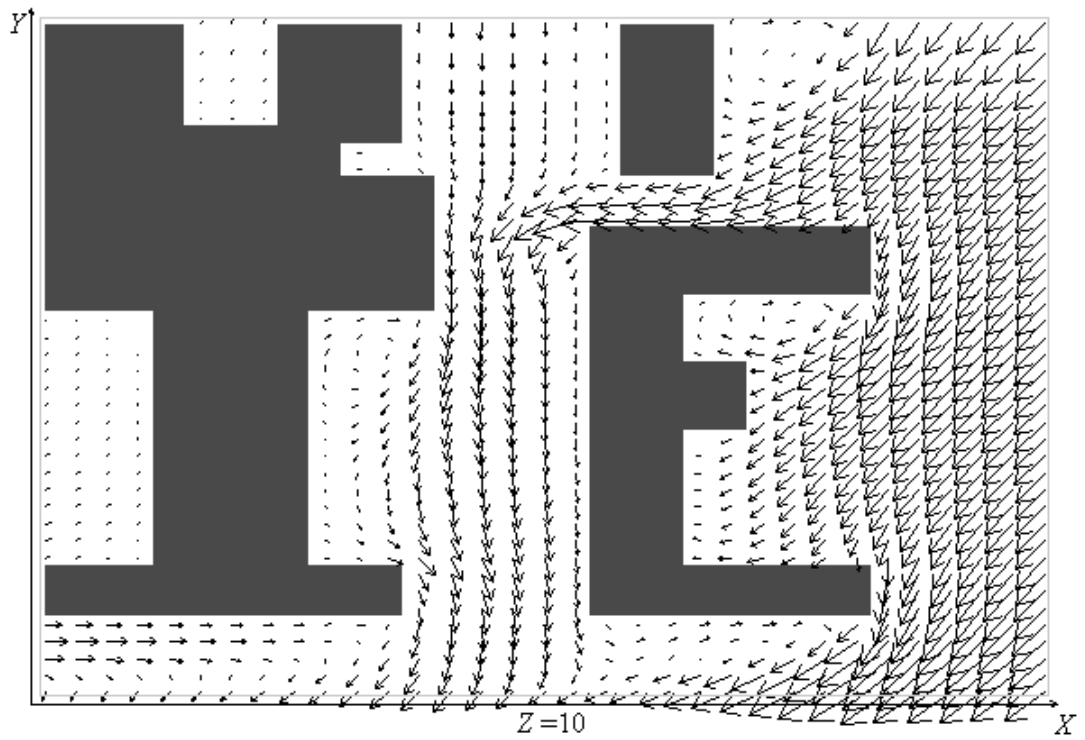
В таблице 5.5 представлены результаты построения компактных описаний по каждой из трех проекций вектора скорости.

Таблица 5.5. Результаты построения компактных описаний для скорости

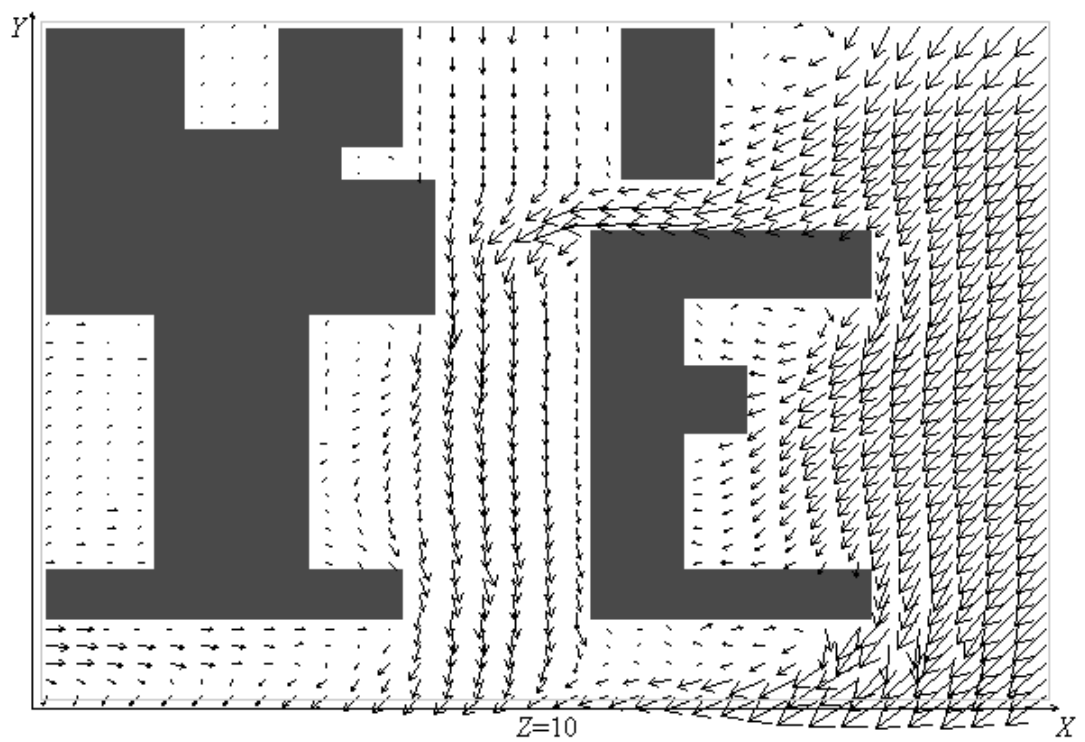
| Параметр  | Проекция скорости на ось x | Проекция скорости на ось y | Проекция скорости на ось z |
|---|----------------------------|----------------------------|----------------------------|
| Количество весов сетей  | 68483                      | 90505                      | 55394                      |
| Количество сетей  | 965                        | 1371                       | 522                        |
| Количество нейронов   | 11372                      | 15494                      | 7833                       |
| Среднее число сетей на кластер  | 3,22                       | 4,57                       | 1,74                       |
| Объем памяти (байт), требуемый для хранения весовых коэффициентов, сжатой информации о распределении узлов по кластерам, о конфигурации сетей | 351306                     | 454010                     | 283002                     |
| Коэффициент сжатия  | 0,018                      | 0,023                      | 0,014                      |
| Число кластеров с одной сетью   | 27                         | 15                         | 138                        |
| Число кластеров с двумя сетями  | 159                        | 92                         | 136                        |
| Число кластеров с четырьмя сетями   | 73                         | 93                         | 24                         |
| Число кластеров с восемью сетями  | 41                         | 100                        | 2                          |
| Средняя абсолютная погрешность, м/с   | 0,218                      | 0,217                      | 0,081                      |

Из таблицы очевидно, что при весьма высокой степени сжатия (в 43÷71 раз) средняя абсолютная погрешность достаточно невелика с учетом того, что скорость

течения в расчетной области принимала значения в диапазоне  $0,1 \div 9,5$  м/с. Это подтверждается также визуально — на рис. 5.2 приведены картины скорости воздушных течений во фрагменте области на высоте 10 метров, полученные в расчете (а) при направлении ветра в  $20^\circ$  и по компактным описаниям (б) расчетных данных. Очень показательны распределения количеств кластеров по числу задействованных сетей. По проекциям на оси  $Ox$  и  $Oy$  преобладают кластеры с более чем одной сетью, что объясняется достаточно сложным характером интерполируемых данных (подтверждается более высокой средней абсолютной погрешностью интерполяции). В то же время по проекции на ось  $Oz$ , напротив, доминируют кластеры с одной-двумя сетями, которые при сравнительно несложном характере исходных данных (погрешность интерполяции существенно ниже) уже дают требуемый результат.



а)



б)

Рис. 5.2. Скорости воздушных течений, полученные в расчете (а) и по компактным описаниям (б)

На рис. 5.2, (а, б) отмечаются лишь небольшие искажения, что, вообще говоря, свидетельствует об адекватности построенных компактных кластерно-нейросетевых описаний. Соответственно, моделирование переноса загрязнителей при использовании восстановленных из описаний полей скорости также должно давать результаты, близкие к тем, которые можно получить при использовании исходных полей скорости.

### **Выводы к пятой главе**

В данной главе проведена апробация разработанного символического ядра путем генерации упрощенных выражений для некоторых нейронных сетей, описывающих турбулентную вязкость. Доказывается (на основании анализа количества требуемых операций), что эти выражения вычисляются быстрее исходной сети, в среднем на 40÷50%. Приведены реальные данные о времени счета (с нейронными сетями и с упрощающими их выражениями), подтверждающие (45%) вышеуказанную выкладку. Показано, что погрешность полученных упрощенных моделей по сравнению с исходными нейросетевыми достаточно мала (1÷3%). Это позволяет сделать выводы о корректности работы разработанного символического ядра, о достаточной точности использованных в главах 3 и 4 приемов и методов и об адекватной их реализации в программном коде.

Даны краткие рекомендации по применению полученных результатов в САПР объектов городской застройки как на этапе анализа, так и на этапе синтеза предлагаемых проектных решений. Показано, что для наиболее успешного применения полученных результатов необходима разработка технологии компактного хранения величин полей, получаемых в ходе численного моделирования для стандартных случаев, и интерполяции величин этих полей для нестандартных случаев.

Соответственно, предложены основные принципы построения компактных кластерно-нейросетевых описаний варьирующихся в зависимости от некоего параметра физических величин. Подробно описан адаптивный эвристический алгоритм построения описаний. Алгоритм отличается наличием процедуры предик-

ции-коррекции погрешности для предполагаемого к использованию количества нейронных сетей.

На данных, полученных в результате моделирования воздушных течений на трехмерном участке, включающем несколько улиц города Ганновер, показано, что получаемые таким образом приближенные описания в  $43\div 71$  раз компактнее исходных данных, при этом погрешность восстановленных данных имеет вполне приемлемую величину. Показаны высокие степень ускорения и эффективность распараллеливания процедуры построения описаний на стандартной 16-ядерной машине платформы Google's Compute Engine

## Заключение

Все задачи, поставленные в работе, выполнены. Получены следующие основные результаты:

1. Предложена концепция расширенных машин Тьюринга (PMT) как теоретического базиса для описания, генерации и трансформации последовательных и параллельных алгоритмов и/или моделей, в том числе ИНС. Предложены параллельная, эволюционная и порождающая PMT. Сформулировано общее утверждение о представимости системной PMT формализмов, сводимых к множеству последовательных и параллельных процессов, подразумевающих исполнение итераций с решением одной или нескольких СЛАУ. Продемонстрировано, что система моделирования на базе ОСМ (расширений PMT) PGEN++, позволяет компактно описывать/генерировать, обучать, анализировать и трансформировать (упрощать) ИНС прямого распространения.

2. Предложена технология планирования и реализации эксперимента по построению комбинированных моделей данных (по принципу МГУА с каскадным частичным пересчетом при выборе наилучшего интерполятора), включающих нейронные сети, полиномы и произвольные функции.

3. Разработаны нейросетевые локально-интерполяционная и интегро-локальная модели турбулентной вязкости, отличающиеся от аналогов применением входных функций (ИНС — ядерные машины), занимающие промежуточное место по точности между однопараметрическими и двухпараметрическими моделями (среднее отклонение локально-интерполяционной модели от наиболее точной К-Е-модели составило 20%), обладающие повышенной скоростью сходимости к решению.

4. Предложен новый подход к контролю погрешности численного расчета поставленных аэродинамических задач с применением перемещающихся частиц-индикаторов в поле «критичности», определяющих порядок точности и степень «неявности» разностной схемы. Подход показал лучшие результаты по скорости схождения к стационарному решению.

5. Разработана новая схема получения приближенных алгебраических описаний для математических выражений, лежащих в основе нейронных сетей, с их последующим символьным упрощением, позволяющим получить более простые соотношения для турбулентной вязкости. В отличие от аналогов, используется дополнительное символическое упрощение нейросетевой модели турбулентной вязкости, позволяющее ускорить процесс расчета при сохранении достаточной точности. Ключевой новой особенностью предложенной схемы является стратегия невозрастания максимальной степени переменных в генерируемом выражении, позволяющая получать более простые и правдоподобные результаты.

6. Предложена стратегия упрощения получаемых приближенных выражений, позволяющая за счет некоторого снижения простоты генерируемой формулы существенно повысить скорость ее генерации при делении полинома на полином. Существенной новой особенностью стратегии является контроль времени поиска очередного члена делимого.

7. Предложены эффективные параллельные алгоритмы, в которых используется новый подход к распараллеливанию задач получения упрощенных нейросетевых моделей: множество процессоров эффективно делится на несколько подмножеств («портфель задач»), каждое из которых обрабатывает часть вариантов упрощения нейронной сети, а в каждом из таких подмножеств реализовано внутреннее дополнительное векторное распараллеливание наиболее трудозатратной операции поиска очередного члена частного. Экспериментально показано, что такая схема распараллеливания дает более оптимальные результаты по сравнению с «чистыми» схемами «портфель задач» и «векторное распараллеливание». На 16-ядерной машине платформы Google's Compute Engine ускорение составило до 4,9 раза, эффективность распараллеливания до 68%. Учитывая, что использовались не реальные, а HyperThreading-ядра, это весьма высокий результат.

8. Разработано простое и быстродействующее параллельное ядро символических преобразований, оптимизированное для Windows- и Linux-систем. Применение данного ядра позволяет быстро и эффективно упрощать символьные выражения. Ядро может применяться в сложных САПР, осуществляющих символьное

или символично-численное преобразование математических моделей для получения упрощенного аналитического или разностного решения.

9. Разработана программа получения приближенных алгебраических описаний для математических выражений, лежащих в основе нейронных сетей, моделирующих турбулентную вязкость. Программа позволяет получить упрощенные, более быстро вычисляемые соотношения, которые можно использовать в сложных математических моделях воздушных течений, применяемых при анализе проектных решений в САПР объектов городской застройки с точек зрения ветровой нагрузки на здания и экологической безопасности. Такой подход позволяет ускорить анализ проектных решений.

10. Проведены численные эксперименты, подтвердившие повышение скорости расчета фактора турбулентности в классической аэродинамической задаче обтекания одиночного препятствия (здания) на  $40\div 45\%$  при вполне допустимой погрешности  $1\div 3\%$  по отношению к исходной нейронной сети. Это подтверждает *достоверность и обоснованность* предложенных схем, стратегий и алгоритмов упрощения нейронных сетей, *адекватность* их реализации в программном коде.

11. Предложена новая схема компактизации полей физических величин, полученных при моделировании картины воздушных потоков с варьирующимся параметром, отличающаяся от прочих алгоритмов высоким коэффициентом сжатия (в  $43\div 71$  раз) при использовании обучения ИНС по кластерам данных с предикционным выбором количества ИНС для каждого кластера. Данная технология может использоваться в САПР для эффективного хранения полученных данных о нестандартных случаях и оперативного получения приближенных данных для нестандартных случаев.

В качестве перспектив развития данной работы можно назвать: а) дальнейшую разработку концепции РМТ для описания не рассмотренных в работе формализмов моделирования; б) усовершенствование алгоритмов упрощения выражений, в частности работу с дробными степенями при переменных; в) реализация алгоритмов распараллеливания для работы с кластерными системами.

ми; г) ввод большего количества заместительных функций, позволяющих более точно описывать отдельные участки активационных сигмOID.

**Библиографический список**

1. Аладьев В.З., Бойко В.К., Ровба Е.А. "Программирование в пакетах Maple и Mathematica: Сравнительный аспект" / Монография.— Гродно: Гродненский Госуниверситет, 2011.— 517 с.
2. Алексеев Б.В., Гришин А.М. Физическая газодинамика реагирующих сред.— М.: Высш. шк., 1985. — 464 с.
3. Балаев Э.Ф., Нуждин Н.В., Пекунов В.В. и др. Численные методы и параллельные вычисления для задач механики жидкости, газа и плазмы: Учебное пособие. — Иваново: Изд-во ИГЭУ, 2003. — 336 с.
4. Бахвалов Н.И., Жидков Н.П., Кобельков Г.М. Численные методы. — Лаборатория Базовых Знаний, 2003. — 632 с.
5. Белоцерковский С. М. О моделировании на ЭВМ турбулентных струй и следов методом дискретных вихрей // Этюды по турбулентности. — М.: Наука, 1994. — С. 246-248.
6. Бендерская Е.Н., Граничин О.Н., Кияев В.И. Параллельные вычисления на базе нелинейных динамических элементов: проблемы и перспективы // В сб.: Труды Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: поиск новых решений», г. Новороссийск, 17-22 сентября 2012 г., М: Изд-во Моск. ун-та. С. 325-329.
7. Бояршинов М.Г. Математическое моделирование взаимодействия растительного массива с воздушным потоком // Восьмой Всероссийский съезд по теоретической и прикладной механике. Анн. докладов.— Пермь, 2001. — С.118.
8. Бусленко Н.П., Калашников В.В., Коваленко И.Н. Лекции по теории сложных систем.— М.: Советское радио.— 1973.— 440 с.
9. Владимирович А. Г., Граничин О. Н., Макаров А. А. Нестандартная машина Тьюринга // «Стохастическая оптимизация в информатике». — Изд-во СПбГУ. — 2005. — С. 28–48.
10. Воеводин А.Ф., Гончарова О.Н., Протопопова Т.В. Трехмерная тепловая конвекция: численный метод решения на основе расщепления по физическим

процессам // Восьмой Всероссийский съезд по теоретической и прикладной механике. Анн. докладов.— Пермь, 2001. — С.159.

11. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. — СПб.: БХВ-Петербург, 2002. — 608 с.

12. Волков А.В. Метод численного исследования обтекания пространственных конфигураций путем решения уравнений Навье-Стокса на основе схем высокого порядка точности // Дис. докт. физ.-мат. наук.— Москва, 2010. — 189 с.

13. Гаврилов М.В. Развитие трехмерных математических моделей приборов М-типа и их применение к магнетронным усилителям: Автореф. дис. канд. физ.-мат. наук. — Саратов, 2001. — 20 с.

14. Галиаскарова Г.Р. Математическое моделирование процесса накопления и распространения тяжелых выбросов в атмосфере // Восьмой Всероссийский съезд по теоретической и прикладной механике. Анн. докладов.— Пермь, 2001. — С.173.

15. Гандер В., Гржебичек И. Решение задач в научных вычислениях с применением Maple и MATLAB.— Изд. "Вассамедина", 2005.— 520 с.

16. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных машин: Уч. пос. — Нижний Новгород: Изд-во ННГУ, 2000. — 176 с.

17. Годунов С.К., Рябенький В.С. Разностные схемы.— М.: Наука, 1973.— 400 с.

18. Гришин А.М. и др. Общая математическая модель и некоторые результаты математического моделирования лесных пожаров // Восьмой Всероссийский съезд по теоретической и прикладной механике. Анн. докладов.— Пермь, 2001. — С.211.

19. Гувернюк С.В., Лоханский Я.К. О моделировании ограниченных закрученных потоков вязкого теплопроводного газа с частицами жидкой и твердой фазы. Применение компьютерной технологии STAR-CD // Тез. докл. III Международной конференции по неравновесным процессам в соплах и струях (NPNJ-2000). — М.: МГИУ, 2000. — С.144-146.

20. Гудкова О.С. Модель «вложенных струй» в описании динамики распространения струй различного состава в атмосфере // Восьмой Всероссийский съезд по теоретической и прикладной механике. Анн. докладов.— Пермь, 2001. — С.216.

21. Давыдов Ю.М., Давыдова И.М. Современные разностные схемы повышенной точности с уменьшением полной вариации на решении для актуальных задач машиностроения // Математ. моделир. систем и процессов: Межвуз. сб. науч. тр./ Перм. гос. техн. ун-т. — Пермь, 2003. — №11. — С.4-18.

22. Дьяконов В. MATLAB 6: УЧЕБНЫЙ КУРС. СПб.: Питер, 2001. — 592 с.

23. Дюк В., Самойленко А. Data mining: учебный курс.— СПб: Питер, 2001. — 368 с.

24. Ершов С.В., Русанов А.В. Численное моделирование турбулентных отрывных течений в пространственных решетках с использованием неявной ENO схемы С.К.Годунова // Пробл. машиностроения. — 1998. — Т.1. — №1. — С.70-78.

25. Ильин В.П. Проблемы высокопроизводительных технологий решения больших разреженных СЛАУ // Вычислительные методы и программирование.— 2009. — Т.10. — №3. — С.141-147.

26. Калиткин Н.Н. Численные методы. — М.: Наука, 1978. — 512 с.

27. Коновалов Н.А., Крюков В.А. DVM-система разработки параллельных программ // Высокопроизводительные вычисления и их приложения: Тр. Всеросс. науч. конф. — М.: Изд-во МГУ, 2000. — С.33.

28. Коньшин В.Н. Параллельная реализация программного комплекса FlowVision // САПР и графика. — 2006. — №12. — С.57-60.

29. Косяков С.В., Исаев С.А., Ермошин А.В., Сизяков Р.А. Разработка ГИС для анализа чрезвычайных ситуаций // Тез. докл. Междунар. науч.-техн. конф. «Х Бенардосовские чтения». - Иваново, 2001. — Т.2. — С.52.

30. Кудинов П.И. Сравнительное тестирование моделей турбулентности Спаларта-Аллмараса и Менгера на задаче о трансзвуковом обтекании одиночного

профиля gae2822 // Вісник Дніпропетровського університету. Серія Механіка. — 2004. — Випуск 8. — Т.1. — С.34-42.

31. Куриземба А.Ж. Метод Монте-Карло для решения разностных уравнений Навье-Стокса // Сб. ст. VII Междунар. науч.-техн. конф. «Информационная среда вуза». — Иваново, 2000. — С.239.

32. Лацис А.О. Как построить и использовать суперкомпьютер.— М.: Бестселлер, 2003.— 240 с.

33. Лойцянский Л.Г. Механика жидкости и газа.— М.: Наука, 1978.—736 с.

34. Марчук Г.И. Математическое моделирование в проблеме окружающей среды. — М.: Наука, 1982. — 319 с.

35. Марчук Г.И. Методы вычислительной математики. — М.: Наука, 1989.— 608 с.

36. Муха Ю.П., Поляков В.С., Поляков С.В. Использование теории автоматов, машин Тьюринга и сетей Петри для создания концепции нейроподобных систем // Известия ВолГТУ.— 2014.— Т.7.— Вып.3(106).— С.76-81.

37. Мэтьюз Дж.Г., Финк К.Д. Численные методы. Использование MATLAB. — М.: Издательский дом «Вильямс», 2001. — 720 с.

38. Нуждин Н.В., Ясинский Ф.Н. Представление о вычисляющей среде и его применение для распараллеливания алгоритмов в механике жидкостей и газов // Вестник ИГЭУ. — Иваново, 2003. — Вып.1. — С.82-84.

39. Одинцов И.О., Черноиванов Д.И. Методы реализации стандарта OpenMP в компиляторах // Высокопроизводительные вычисления и их приложения: Тр. Всеросс. науч. конф. — М.: Изд-во МГУ, 2000. — С.140-144.

40. Орлов А.А. Принципы построения архитектуры программной платформы для реализации алгоритмов метода группового учета аргументов // УСиМ.— 2013.— №2.— С. 65-71.

41. Осипов В.Ю. Ассоциативные интеллектуальные машины // Информационные технологии и вычислительные системы. — 2010.— №2.— С.59-67.

42. Остерн М.Г. Обобщенное программирование и STL: Использование и наращивание стандартной библиотеки шаблонов C++. — СПб.: Невский Диалект, 2004. — 544 с.

43. Охорзин В.А. Прикладная математика в системе MATHCAD Учебное пособие.— СПб.: Лань, 2009.— 352 с.

44. Очков В.Ф. Mathcad 14 для студентов и инженеров: русская версия.— СПб.: BHV, 2009.

45. Пекунов, В.В. Автоматизация параллельного программирования при моделировании многофазных сред. Оптимальное распараллеливание // Автоматика и телемеханика.— 2008.— №7.— С.170-180.

46. Пекунов В.В. Локальные нейросетевые модели турбулентности // Мат. Междунар. науч.-техн. конф. "XVIII Бенардосовские чтения".— Иваново, 2015.— Т.2.— С.331-334.

47. Пекунов В.В. Метаслой моделирования алгоритма, данных и функциональных характеристик последовательных и параллельных программ // Информационные технологии.— 2011.— №6.— С.51-56.

48. Пекунов В.В. Модель образования и распространения твердых, жидких и газообразных загрязнителей. Оптимальное распараллеливание // Математическое моделирование.— 2009. — Т.21. — №3. — С.69-82.

49. Пекунов, В.В. Новые методы параллельного моделирования распространения загрязнений в окрестности промышленных и муниципальных объектов // Дис. докт. тех. наук. — Иваново, 2009. — 274 с.

50. Пекунов, В.В. О классификации лиц методом голосования с нейросетевым арбитром. Распараллеливание вычислений на многоядерных видеокартах // Информационные технологии.— 2013. — №3.— С.61-65.

51. Пекунов, В.В. Параллельное решение задачи классификации лиц методом голосования с нейросетевым арбитром // Сб. матер. XII Всеросс. конф. "Высокопроизводительные параллельные вычисления на кластерных системах".— Нижний Новгород:Изд-во ННГУ, 2012.— С.320-324.

52. Пекунов, В.В. Теория объектно-событийных моделей. Индукция, моделирование и синтез последовательных и параллельных программ.— LAP LAMBERT Academic Publishing, 2012.— 132 с.
53. Полак Л.С., Гольденберг М.Я., Левицкий А.А. Вычислительные методы в химической кинетике. — М.: Наука, 1984. — 280 с.
54. ПРИЗМА-ПРЕДПРИЯТИЕ - унифицированный программный комплекс расчета загрязнений атмосферы. — <http://www.infars.ru/leaflets/prism.html>.
55. Рассел С., Норвиг П. Искусственный интеллект: современный подход.— М.: «Вильямс», 2007. — 1408 с.
56. Роуч П. Вычислительная гидродинамика.— М.: Мир, 1980. — 612 с.
57. Рофэйл Эш, Шохауд Я. СОМ и СОМ+: Полное руководство.— К.: ВЕК+, К.: НТИ, М.: Энтроп, 2000.— 560 с.
58. Савицкий Г.А. Ветровая нагрузка на сооружения. — М.: Стройиздат, 1972.— 110 с.
59. Самарский А.А., Попов Ю.П. Разностные методы решения задач газовой динамики. — М.: Наука, 1992. — 424 с.
60. Сидоров, С.Г. Разработка ускоренных алгоритмов обучения нейронных сетей и их применение в задачах автоматизации проектирования: Дис. ... канд. тех. наук.— Иваново, 2003.— 161 с.
61. Степашко В.С., Булгакова А.С. Обобщенный итерационный алгоритм метода группового учета аргументов // УСиМ.— 2013.— №2.— С. 5-18.
62. Тихонов А.Н., Самарский А.А. Уравнения математической физики.— М.: Наука, 1977. — 742 с.
63. Турбулентное смешение газовых струй/ Абрамович Г.Н., Крашенинников С.Ю., Секундов А.Н., Смирнова И.П. — М.: Наука, 1974.— 272 с.
64. Турбулентность: модели и подходы. Курс лекций. Часть I / П.Г.Фрик; Перм. гос. техн. ун-т. — Пермь, 1988. — 108 с.
65. Турбулентность: модели и подходы. Курс лекций. Часть II / П.Г.Фрик; Перм. гос. техн. ун-т. — Пермь, 1988. — 136 с.

66. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика.— М.: Мир, 1992.
67. Форрестер Дж. Мировая динамика.— М.: Наука, 1978. — 168 с.
68. Хокни Р., Иствуд Дж. Численное моделирование методом частиц. — М.: Мир, 1987. — 640 с.
69. Чефранов С.Г., Чефранов А.Г., Данилин М.В. Параллельная реализация гибридного эйлерово-лагранжевого алгоритма дальнего переноса примесей на сетевых кластерах с использованием технологии DCOM // Высокопроизводительные вычисления и их приложения: Тр. Всеросс. науч. конф. — М.: Изд-во МГУ, 2000. — С.216-220.
70. Шпаковский Г.И., Серикова Н.В. Программирование для многопроцессорных систем в стандарте MPI. — Минск: БГУ, 2002. — 324 с.
71. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования. — М.: Издательский дом «Вильямс», 2003. — 512 с.
72. Якобовский М.В. Вычислительная среда для моделирования задач механики сплошной среды на высокопроизводительных системах: Автореф. дис. докт. физ.- мат. наук. — Москва, 2006. — 37 с.
73. Яненко Н.Н. Метод дробных шагов решения многомерных задач математической физики. — Новосибирск: Наука, 1967. — 197 с.
74. Abdol-Hamid, K.S., Massey, S.J., Caldwell, S. Unified process management system for computational fluid dynamics (UPMS). 41st AIAA Aerospace Sciences Meeting and Exhibit, 2003. Reno, Nevada, USA, paper AIAA 2003-0803, 2003.
75. Ben-Israel, A., Gilbert, R.: Computer-Supported Calculus. — Springer: Wien, 2002.
76. Benyahia, S., Arastoopour, H., Knowlton, T.M., Massah, H. Simulation of particles and gas flow behavior in the riser section of a circulating fluidized bed using the kinetic theory approach for the particulate phase // Powder Technology Journal. — 2000. — Vol.112. — pp 24-33.

77. Celik, I., Chen, C.J., Roache, P.J. and Scheurer, G., eds. 1993, "Quantification of Uncertainty in Computational Fluid Dynamics," ASME Fluids Engineering Division Summer Meeting, 158, Washington, DC, 20-24 June.

78. Celik, I., Karatekin, O. Numerical Experiments on Application of Richardson Extrapolation With Nonuniform Grids // ASME Journal of Fluid Engineering, 1997, 119. P.584-590.

79. Chung, T.J. Computational Fluid Dynamics. — Cambridge University Press, 2002. — 1012 p.

80. Dupuis, M., Dervedde, E., Methot, J-C. La modélisation de la turbulence dans une enceinte avec ouvertures et sources chaudes localisées // The Canadian Journal of Chemical Engineering.— 1989.— Vol.67.— pp 713-721.

81. Efficient Dynamic Parallelism with OpenMP on Linux SMPs. Antonopoulos C.D., Venetis I.E., Nikolopoulos D.S., Papatheodorou T.S. / Proc. of the Sixth International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, July 2000. CSREA Press. — Vol.5. — pp 2507-2514.

82. Ferziger, J.H., Peric, M. Computational Methods for Fluid Dynamics. — Springer-Verlag, 2002. — 424 p.

83. FLUENT 6 User's Guide. — Fluent Inc., 2001.

84. Guideline on Air Quality Models (Revised) and Supplements. — EPA-450/2-78-027R et seq., Appendix W to 40 CFR Part 51 (7-1-99 Edition). U. S. Environmental Protection Agency, Research Triangle Park, North Carolina, 1999.

85. Ito, T., Sakamoto, M., Taniue, A., Kono, M. Parallel Turing machines on four-dimensional input tapes // Artificial Life and Robotics.— 2010.— Vol.15.— No.2.— pp.212-215.

86. Ketzler M., Louka P., Sahm P., Guilloteau E., Sini J.-F. The use of computational fluid dynamics in modelling air quality in street canyons / Included in the final report of the TMR research network TRAPOS «Optimisation of Modelling Methods for Traffic Pollution in Streets», submitted to the European Commission, July 2001.

87. Ketzl M., Louka P., Sahn P., Guilloteau E., Sini J.-F., Moussiopoulos N. Intercomparison of Numerical Urban Dispersion Models — Part II: Street Canyon in Hannover, Germany / The 3rd Urban Air Quality conference – Loutraki 19-23 March 2000.

88. Leitl B., Schatzmann M. CEDVAL — Compilation of Experimental Data for Validation of Microscale Numerical Dispersion Models / Proc. 2<sup>nd</sup> East European Conference on Wind Engineering. Prague, September 7-11, 1998.

89. Mali O., Neittaanmaki P., Repin S. Accuracy Verification Methods: Theory and Algorithms. — Springer, 2013. — 386 pp.

90. Mieth, P., Unger, S., Moussiopoulos, N. et al. ECOSIM Telematics Applications Project: ECOSIM Deliverable D05.02. ECOSIM User Documentation. — <http://www.ess.co.at/ECOSIM/Deliverables/D0502.html>.

91. MPI-2: Extensions to the Message-Passing Interface. — <http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html>.

92. Mumovic D., Crowther J.M., Stevanovic Z. The Effect of Turbulence Models on Numerical Prediction of Air Flow within Street Canyon / The First International Conference on Computational Mechanics (CM'04) Belgrade, November 15-17. — University of Belgrade, 2004.

93. Ponomarev, D., Mizonov, V., Gatumal, C., Berthiaux, H., Barantseva, E. Markov-chain modelling and experimental investigation of powder-mixing kinetics in static revolving mixers // Chemical Engineering and Processing.— №48 (2008).— 828-836 pp.

94. Press, W.H. et al. Numerical recipes in C: The art of scientific computing. — Cambridge University Press, 1992.— 994 p.

95. Segelstein, D.J. The complex refractive index of water: M.S.Thesis. — University of Missouri, Kansas City, 1981.

96. Schatzmann M., Leitl B., Liedtke J. Ausbreitung von Kfz-Abgasen in Straßenschluchten / Forschungsbericht FZKA-BWPLUS (Final report of the research project PEF 2 96 001). — 1999.

97. Traversa F.L., Di Ventra M. Universal memcomputing machines // IEEE Trans. Neural Netw. Learning Syst. — 2015. — Vol. 26(11). — pp 2702-2715.

98. User's Guide for CAL3QHC Version 2: A Modeling Methodology for Predicting Pollutant Concentrations Near Roadway Intersections. — EPA-454/R-92-006, U. S. Environmental Protection Agency, Research Triangle Park, North Caroline, 1992.

99. User's guide for the Industrial Source Complex (ISC3) dispersion models. Volume II - description of model algorithms. — EPA-454/B-95-003b, U. S. Environmental Protection Agency, Research Triangle Park, North Caroline, 1995.

100. Wang D., Ruuth S. J. Variable step-size implicit-explicit linear multistep methods for timedependent partial differential equations // Journal of Computational Mathematics, 26(6):838–855, 2008. 22, 23.

101. Wiedermann, J., Parallel Turing machines, Tech. Report RUU-CS-84-11, Dept. of Computer Science, Univ. of Utrecht, 1984.

102. Wiedermann Jiří, Pardubská, D.: On the Power of Broadcasting in Mobile Computing. In: New Computational Paradigms. Changing Conceptions of what is Computable. Berlin : Springer, 2008 - (Ed.: Cooper B., Löwe B., Sorbi A.), pp. 195-209.

103. Wilcox, D.C. Turbulence modeling for CFD. — DCW Industries, Inc, 1994. — 460 p.

104. Worsch T. Parallel Turing machines with one-head control units and cellular automata // Theor. Comput. Sci. — 1999.— Vol.217.— No 1.— pp. 3-30.

105. Yap C. J. Turbulent Heat and Momentum Transfer in Recirculating and Impinging flows, PhD Thesis, Faculty of Technology, University of Manchester, United Kingdom. — 1987.

106. Yershov, S.V., Rusanov, A.V. (1996), The new high resolution method of Godunov's type for 3D viscous flow calculations. The 3rd Colloq. Process Simulation, ed. A. Jokilaakso, 12 - 14 June 1996, Espoo, Finland, pp. 69-85.