УЧРЕЖДЕНИЕ РОССИЙСКОЙ АКАДЕМИИ НАУК
КОНСТРУКТОРСКО-ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ
ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ СИБИРСКОГО ОТДЕЛЕНИЯ РАН

На правах рукописи

Панов Никита Владимирович

Разработка рандомизированных алгоритмов в интервальной глобальной оптимизации

01.01.07 — вычислительная математика

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель
д. ф.-м. н. Шарый Сергей Петрович

Новосибирск — 2012

Оглавление

Введение

Предмет представляемой работы — задача глобальной оптимизации вещественнозначной функции $f : \boldsymbol{X} \to \mathbb{R}$ на прямоугольном брусе $\boldsymbol{X} \subset \mathbb{R}^n$ со сторонами, параллельными координатным осям:

$$\text{найти} \quad \min_{x \in \boldsymbol{X}} f(x) \ . \tag{1}$$

Задача оптимизации — одна из востребованных проблем современной вычислительной и прикладной математики. Решение подобных проблем требуется во многих задачах из различных отраслей науки и техники. Леонард Эйлер (1707–1783), один из величайших математиков, говорил: «В мире не происходит ничего, в чём бы не был виден смысл какого-нибудь максимума или минимума» [57].

Особый интерес представляет нахождение так называемых глобальных экстремумов (оптимумов) функции, т. е. таких, которые являются наилучшими на всей рассматриваемой области определения целевой функции, а не только в сравнении с близлежащими точками из некоторой своей окрестности. Следует отметить, что в самом общем случае задача глобальной оптимизации является труднорешаемой. Современные численные методы её решения сводятся, по существу, к нахождению значений всех локальных оптимумов целевой функции и сравнению их между собой, причём подобное положение не является следствием ущербности, недоработанности и т. п. существующих методик или недостаточным уровнем развития современной теории оптимизации, а носит принципиальный характер. В 1984 году А.А. Гагановым [11] было строго показано, что даже для полиномиальной целевой функции $f(x)$ задача глобальной оптимизации (1) на прямоугольном брусе является NP-трудной, что, фактически, равносильно признанию того, что для её решения требуются не менее чем экспоненциальные в зависимости от размерности n трудозатраты. Один из последних эффектных результатов в этом направлении — теорема Кирфотта-Крейновича [100], утверждающая, что за пределами класса выпуклых целевых функций решение задачи глобальной оптимизации (1) является NP-трудным.

Осознание специфичности поиска глобальных оптимумов и его сложности, и, как следствие, оформление глобальной оптимизации в отдельную ветвь общей теории оптимизации произошло в 70-е годы прошлого века. Впервые этот термин прозвучал в работах Торна, Бекера и Лаго, а также Диксона и Сзего [79, 87, 88, 129, 130].

К настоящему моменту наработан богатый инструментарий поиска глобального оптимума [9, 10, 17, 21, 58, 78, 85, 86, 91, 94, 97, 106]. Существующие методы можно условно разделить, с одной стороны, на детерминистские и стохастические, а с другой, по применяемой в них технике, на классические точечные и интервальные. Точнее это разделение можно представить в виде дерева на рис. 1.

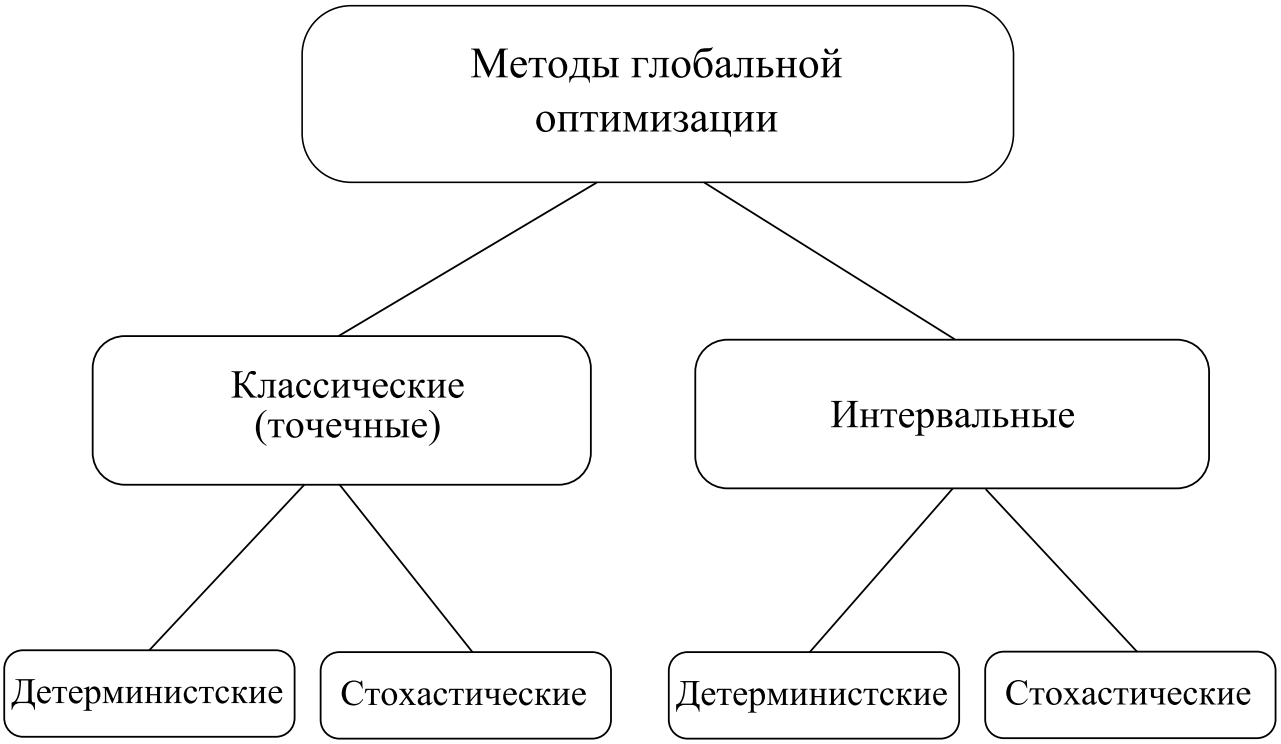


Рис. 1: Классификация алгоритмов глобальной оптимизации

В представленной схеме правая нижняя ветка, отвечающая интервальным стохастическим методам, является весьма неразвитой и до сих пор представлена лишь публикациями [54, 70, 71]. Настоящая работа призвана в какой-то мере восполнить этот дисбаланс и, соответственно, посвящена разработке интервальных стохастических алгоритмов глобальной оптимизации функций.

Актуальность темы диссертационной работы. Многие задачи, возникающие в различных сферах человеческой деятельности, могут быть сведены к задаче поиска глобального оптимума. Оптимизация является неотъемлемой частью важнейших этапов моделирования технических, социальных,

экономических и других систем. В ряде случаев именно сложность возникающей оптимизационной задачи становится тем ограничением, которое не позволяет решить обратную задачу или исследовать общую постановку проблемы.

В настоящее время глобальная оптимизация — широко востребованное и интенсивно развивающееся направление вычислительной математики. Трудности численного решения оптимизационных задач во многом связаны с видом оптимизируемой целевой функции и количеством её аргументов. Целевая функция может быть невыпуклой, недифференцируемой, негладкой, многоэкстремальной. Кроме того, каждое вычисление значений целевой функции может требовать значительных вычислительных ресурсов.

В различных областях науки выдвигается всё больше задач, сводящихся к поиску именно глобального оптимума. Это закономерно привело к росту интереса к проблемам глобальной оптимизации и выделению её в отдельную ветвь математического программирования. Подходы глобальной оптимизации существенно отличаются от техники стандартных методов поиска локальных оптимумов функции (часто неспособных найти глобальный оптимум рассматриваемых многоэкстремальных задач) и характеризуются высокой вычислительной трудоемкостью.

Во многих задачах, возникающих в практике оптимизации, требуется не просто приближённое численное решение, но ещё и гарантия его близости к идеальному математическому оптимуму, а также часто гарантия того, что найденный оптимум действительно является глобальным. Подобные постановки задач обычно характеризуют термином «доказательная глобальная оптимизация», и они являются чрезвычайно трудными. Традиционные подходы к их решению основываются на привлечении той или иной априорной информации о целевой функции (например, того, что она удовлетворяет условию Липшица). Существенное продвижение в решении задач доказательной глобальной оптимизации достигнуто благодаря использованию методов интервального анализа. Эти методы позволяют успешно решать задачи с осложнёнными целевыми функциями (нелипшицевыми, недифференцируемыми и т. п.). Но и в случаях, не требующих доказательного ответа, интервальные методы также вносят новые возможности в технику решения задачи глобальной оптимизации. Несмотря на явный прогресс в этой области за последние два десятилетия и экспоненциальный рост возможностей вычислительной техники, существующие алгоритмы доказательной глобальной оптимизации недостаточно эффективны, что не позволяет решать множество актуальных задач.

Учитывая практическую важность задач глобальной оптимизации функций многих переменных (в том числе доказательной оптимизации), существующие сложности на пути их решения и современное состояние вопроса, исследования по разработке эффективных алгоритмов решения подобных задач представляются важными и актуальными. Им и посвящена данная диссертационная работа.

Объектом диссертационного исследования являются алгоритмы интервальной глобальной оптимизации, которые могут быть использованы для успешного решения сложных многомерных и многоэкстремальных задач в физике, химии, биологии, экономике, приборостроении и пр.

Предметом диссертационного исследования являются методы интервального расширения функций, интервальные алгоритмы глобальной оптимизации, рандомизированные (стохастические) методы глобальной оптимизации, математические модели генетических (эволюционных) алгоритмов.

Целью диссертационного исследования является повышение вычислительной эффективности интервальных алгоритмов доказательной глобальной оптимизации.

Общая методология исследования базируется на положениях математического анализа и алгебры, интервального анализа, теории оптимизации, информатики и математического моделирования.

Научная новизна. В работе получены следующие результаты:

1) Исследована точность различных способов вычисления интервальных расширений функций.
2) Выявлены и проанализированы причины недостаточной вычислительной эффективности существующих детерминистских алгоритмов интервальной глобальной оптимизации, основанных на адаптивном дроблении области поиска. Рассмотрены способы повышения вычислительной эффективности интервальных методов глобальной оптимизации.
3) Продемонстрирована вычислительная эффективность стохастических (рандомизированных) алгоритмов интервальной глобальной оптимизации в сравнении с детерминистскими аналогами на ряде общеупотребительных тестовых задач.
4) Модифицированы существующие рандомизированные (стохастические) алгоритмы интервальной глобальной оптимизации, такие как случайное интервальное дробление[1]. Предложены и исследованы некоторые новые

[1]Предложено ранее научным руководителем работы, см. *Шарый С.П. Стохастические подходы в интервальной глобальной оптимизации. Труды XIII Байкальской международной школы-семинара «Методы оптимизации и их приложения», Иркутск–Северобайкальск, 2–8 июля 2005 года. Том 4 «Интервальный*

подходы, в частности, интервальное дробление в случайной пропорции и направлении.

5) Разработаны и экспериментально исследованы различные версии интервального эволюционного алгоритма для глобальной оптимизации функций.
6) Для предложенных алгоритмов сформулирован и доказан ряд утверждений и теорем о сходимости разработанных методов. В частности, доказано, что метод интервального случайного поиска с приоритетом сходится к глобальному оптимуму почти всегда (которая сильнее «сходимости по вероятности»), а интервальный алгоритм имитации отжига и интервальный эволюционный алгоритм сходятся к глобальному оптимуму.
7) Разработан и апробирован интервальный мультиметодный[2]. алгоритм глобальной оптимизации, созданный на основе стохастических интервальных подходов. Приведены теоретические оценки эффективности и времени работы алгоритма, в том числе выделен худший для алгоритма случай и показано преимущество мультиметодного подхода по сравнению с традиционными. С помощью мультиметодного алгоритма успешно решена задача оптимизации в промышленной модели предсказания погоды.

Достоверность и научная обоснованность результатов диссертации обеспечивается использованием классических математических методов при разработке и обосновании алгоритмов, применением современных методик тестирования и валидации вычислительных программ, сопоставлением полученных результатов с теоретическими и результатами других методов глобальной оптимизации.

Теоретическая и практическая ценность. Работа носит как теоретический, так и практический характер. Алгоритмы, разработанные в диссертации, и их программные реализации могут быть использованы для решения сложных задач глобальной оптимизации, возникающих в физике, биологии, химии, экономических исследованиях, в промышленности, на транспорте и в других отраслях. Кроме того, результаты, полученные в диссертации, могут быть использованы при дальнейшем развитии интервальных рандомизиро-

анализ». – Иркутск, ИСЭМ СО РАН, 2005. – С. 85–105.

[2]По терминологии, берущей начало с работ А.И. Тятюшкина и А.Ю. Горнова. См. Горнов А.Ю. Тятюшкин А.И. Программная реализация мультиметодной технологии для задач оптимального управления // *Сборник трудов 3-й Междунар. конф. «Проблемы управления и моделирования в сложных системах»*, Самара. 2001. С. 301–307. и Горнов А.Ю. *Вычислительные технологии решения задач оптимального управления.* Новосибирск: Наука. 2009.

ванных (стохастических) методов, в частности, перспективных интервальных генетических алгоритмов.

Личный вклад. Соискателем проведено экспериментальное исследование точности различных форм интервальных расширений. Это позволяет обоснованно выбирать те или иные формы интервальных расширений при решении практических задач. Установлено, что зачастую естественное интервальное расширение более предпочтительно, чем центрированные формы, несмотря на более высокий асимптотический порядок точности последних. Этот результат стал основным исследовательским итогом главы 1 диссертации.

Значительная часть современных интервальных методов глобальной оптимизации основана на детерминистской схеме распространенного алгоритма «адаптивного интервального дробления». Автором выделены и проанализированы причины недостаточной эффективности алгоритмов, опирающихся на это подход. На основании чего предложен и исследован ряд способов преодоления выявленных узких мест алгоритмов интервальной глобальной оптимизации и повышения их производительности. Автором проведено сравнение эффективности различных модификаций интервальных алгоритмов глобальной оптимизации и процедур отбраковки. Для сравнительного анализа работы существующих и новых алгоритмов автором разработана программная система, включающая в себя сервер приложений, позволяющий выбирать методы оптимизации, исполнять их в различных режимах и модули трёхмерной визуализации.

Соискателем модифицированы существующие стохастические (рандомизированные) алгоритмы интервальной глобальной оптимизации, такие как случайное интервальное дробление (предложенное ранее научным руководителем). Предложены и исследованы новые алгоритмы, в частности, интервальное дробление в случайной пропорции и направлении. Разработаны и экспериментально исследованы различные версии интервального генетического алгоритма для доказательной глобальной оптимизации функций. Для предложенных алгоритмов сформулированы и доказаны результаты о сходимости разработанных алгоритмов. В частности, доказано, что алгоритм интервального случайного поиска с приоритетом сходится к глобальному оптимуму почти наверное (почти всегда), а интервальный алгоритм имитации отжига и интервальный генетический алгоритм гарантированно сходятся к глобальному оптимуму. Разработан и апробирован интервальный мультиметодный алгоритм глобальной оптимизации, созданный на основе стохастических интервальных подходов. Приведены теоретические оценки эффективно-

сти и времени работы алгоритма. Выявлен худший для алгоритма случай и показано преимущество мультиметодного подхода по сравнению с традиционным. Проработаны вопросы параллелизма предлагаемых алгоритмов и их реализации на современных вычислительных системах.

Основные результаты диссертации докладывались и обсуждались

- на объединённом семинаре Института вычислительных технологий СО РАН, Конструкторско-технологического института СО РАН и Новосибирского государственного университета (г. Новосибирск, 2011);
- на XII Всероссийской конференция молодых ученых по математическому моделированию и информационным технологиям (г. Новосибирск, 2011);
- на Международной конференции «Современные проблемы прикладной математики и механики: теория, эксперимент и практика», посвященной 90-летию со дня рождения академика Н.Н. Яненко. (г. Новосибирск, 2011);
- на XV Международной Байкальской школе-семинаре «Методы оптимизации и их приложения». (п. Листвянка, 2011);
- на семинаре Института систем энергетики СО РАН (г. Иркутск, 2010);
- на семинаре Института динамики систем и теории управления СО РАН (г. Иркутск, 2009, 2010);
- на семинаре Сибирского федерального университета (г. Красноярск, 2009);
- на семинаре кафедры математического моделирования НГУ и Института вычислительных технологий СО РАН (г. Новосибирск, 2006, 2008, 2009);
- на семинаре математического факультета Алтайского Государственного Университета (г. Барнаул, 2009);
- на IX Всероссийской конференция молодых ученых по математическому моделированию и информационным технологиям (г. Кемерово, 2008);
- на Международной конференции «Современные проблемы математического моделирования и вычислительных технологий - 2008» (г. Красноярск, 2008);
- на VIII Всероссийской конференции молодых учёных по математическому моделированию и информационным технологиям (г. Новосибирск, 2007);

— на Всероссийской конференции по вычислительной математике «КВМ-2007» (г. Новосибирск, 2007);
— на конференции «Twelfth ECMWF Workshop "Use of High Performance Computing in Meteorology"» (Англия, 2007);
— на VII Всероссийской конференции молодых ученых по математическому моделированию и информационным технологиям (г. Красноярск, 2006);
— на Всероссийском (с международным участием) совещании по интервальному анализу и его приложениям «ИНТЕРВАЛ-06» (г. Петергоф, 2006);
— на VI Всероссийской (с участием иностранных ученых) конференции молодых ученых по математическому моделированию и информационным технологиям (г. Кемерово, 2005);
— на X Всероссийской научной конференции студентов-физиков и молодых ученых (г. Красноярск, 2004);
— на V Международной конференции «Перспективы систем информатики» памяти акад. А.П. Ершова – PSI'03 (г. Новосибирск, 2003);
— на XLI Международной научной студенческой конференции «Студент и научно-технический прогресс» (г. Новосибирск, 2003).

Публикации. По теме диссертации опубликовано 20 работ, из них 16 в форме трудов, тезисов и расширенных тезисов докладов конференций, а так же 4 статьи в журналах, рекомендованных ВАК для публикации основных результатов диссертаций. (выделены жирным шрифтом в списке литературы).

Апробация результатов работы. Разработанные в диссертации алгоритмы реализованы в виде программного продукта для решения задач глобальной оптимизации, а также в виде подключаемых библиотек, которые могут быть использованы другими пакетами для решения задачи глобальной оптимизации. Разработанные алгоритмы нашли применение при решении задачи всесторонней автоматической верификации микропроцессорных устройств [39], что позволило повысить качество тестового покрытия и уменьшить сроки тестирования. Один из разработанных в диссертации алгоритмов был встроен в промышленную модель предсказания погоды [112], что позволило увеличить её производительность. В настоящее время автор работает над эффективным решением проблем глобальной оптимизации, возникающих в задачах эллипсометрии. Следующей возможной областью применения разработанных методов будет расчёт конфигураций молекул белка.

На защиту выносятся:

— Результаты исследований точности различных интервальных расширений функций.

— Результаты сравнительных исследований детерминистских и рандомизированных интервальных алгоритмов глобальной оптимизации.

— Создание интервального эволюционного алгоритма доказательной глобальной оптимизации.

— Разработка универсального самоподстраивающегося мультиметодного алгоритма глобальной оптимизации.

— Создание оптимизированно адаптивного параллельного алгоритма глобальной оптимизации.

Структура и объем работы. Диссертационная работа состоит из введения, трёх глав основного текста, заключения и библиографического списка. Работа изложена на 178 страницах машинописного текста, включающих 50 рисунков и список литературы из 131 наименования.

СОДЕРЖАНИЕ РАБОТЫ

Во введении описан предмет представляемой работы — задача глобальной оптимизации вещественнозначной функции $f : \boldsymbol{X} \to \mathbb{R}$ на прямоугольном брусе $\boldsymbol{X} \subset \mathbb{R}^n$ со сторонами, параллельными координатным осям.

Подчёркивается, что задача оптимизации — одна из востребованных проблем современной вычислительной и прикладной математики, при этом особый интерес представляет нахождение так называемых глобальных экстремумов (оптимумов) функции, т. е. таких, которые являются наилучшими на всей рассматриваемой области определения целевой функции, а не только в сравнении с близлежащими точками из некоторой своей окрестности.

Отмечается, что в общем случае задача глобальной оптимизации является NP-трудной, что, фактически, равносильно признанию того, что для её решения требуются не менее чем экспоненциальные в зависимости от размерности n трудозатраты.

Даётся краткая классификация существующих методов глобальной оптимизации.

В главе 1 кратко приведены основы интервального анализа. Помимо традиционных результатов по интервальной арифметике и интервальным расширениям функций, также рассмотрены различные способы нахождения на

ЭВМ производных функций и их интервальных расширений. Соответствующий материал слабо освещён в литературе.

Основным исследовательским итогом главы 1 диссертации является экспериментальное исследование точности различных интервальных расширений, позволяющее обоснованно выбирать те или иные формы интервальных расширений при решении практических задач. Сделанные выводы являются неочевидными и свидетельствуют о том, что в ряде случаев естественное интервальное расширение более предпочтительно, чем центрированные формы, имеющие более высокий асимптотический порядок точности [46].

В главе 2 проведён обзор современных алгоритмов глобальной оптимизации, классических и интервальных.

Существующие интервальные методы глобальной оптимизации, основанные на адаптивном дроблении области поиска, используют тот факт, что большинство интервальных оценок области значений функции — асимптотически точные:

$$\operatorname{dist}\left(\boldsymbol{f}(\boldsymbol{x}), \operatorname{ran}_{\boldsymbol{x}} f\right) \to 0 \qquad \text{при} \quad \|\operatorname{wid} \boldsymbol{x}\| \to 0, \tag{2}$$

где dist — хаусдорфово расстояние между интервальным расширением $\boldsymbol{f}(\boldsymbol{x})$ и точной областью значений $\operatorname{ran}_{\boldsymbol{x}} f$ функции f на интервале $\boldsymbol{x}$ (возможно, многомерном), а $\operatorname{wid} \boldsymbol{x}$ — ширина интервала $\boldsymbol{x}$. Это означает, что при уменьшении размеров области определения точность интервального расширения функции увеличивается. При этом не следует ожидать, что уменьшение размеров конкретного бруса области определения в какое-то число раз приведёт к пропорциональному улучшению реальной точности интервальной оценки. Приведённое соотношение является, во-первых, всего лишь оценкой сверху, и, во-вторых, носит асимптотической характер. Тем не менее, отмеченный факт может быть положен в основу процедуры уточнения интервальной оценки области значений функции.

В самом деле, если разбить исходный брус $\boldsymbol{x}$ на два подбруса $\boldsymbol{x}'$ и $\boldsymbol{x}''$, дающие в объединении весь $\boldsymbol{x}$, то есть такие, что $\boldsymbol{x}' \cup \boldsymbol{x}'' = \boldsymbol{x}$, то

$$\{\, f(x) \mid x \in \boldsymbol{x} \,\} = \{\, f(x) \mid x \in \boldsymbol{x}' \,\} \cup \{\, f(x) \mid x \in \boldsymbol{x}'' \,\}.$$

Соответственно, можно вычислить интервальные расширения на каждом подбрусе, а в качестве новой оценки минимума целевой функции на $\boldsymbol{x}$ взять

$$\min\{\, \underline{\boldsymbol{f}(\boldsymbol{x}')}\,,\ \underline{\boldsymbol{f}(\boldsymbol{x}'')} \,\},$$

и она будет, вообще говоря, более точна, чем исходная оценка $\boldsymbol{f}(\boldsymbol{x})$, так как у брусов $\boldsymbol{x}'$ и $\boldsymbol{x}''$ размеры меньше, чем у исходного $\boldsymbol{x}$. Брусы-потомки $\boldsymbol{x}'$ и $\boldsymbol{x}''$

можно, в свою очередь, опять разбить на более мелкие части, найти для них интервальные расширения и далее уточнить оценку для минимума, потом снова повторить процедуру и так далее. Все брусы-потомки добавляются в «рабочий список». Ведущим или наиболее перспективным называется брус, на котором в настоящий момент достигается наибольшая (наименьшая при поиске минимума) оценка значения функции. Критерием остановки может быть максимальное количество итераций, достаточная ширина оценки оптимума и т. п.

Далее в главе 2 проанализированы и выделены причины недостаточной эффективности алгоритмов, опирающихся на эту схему [44, 54]. Кроме того, проанализированы особенности современных машинных вычислений, приведена информация об аппаратных факторах, влияющих на быстродействие программы, сделаны выводы об оптимальной с этой точки зрения организации данных. Приведены рекомендации по наиболее оптимальной программной реализации соответствующих алгоритмов на различных вычислительных системах.

Для анализа работы существующих алгоритмов и сравнения новых использована специально разработанная программная система, включающая в себя сервер приложений, позволяющий выбирать методы оптимизации, исполнять их (в том числе в режиме профилирования) и модули трёхмерной визуализации [47].

В результате проведённого анализа в качестве основных причин неуспешности простейшего интервального адаптивного алгоритма глобальной оптимизации можно назвать следующие:

— получаемые интервальные оценки области значений функции весьма избыточны;

— зачастую границы интервальных оценок целевой функции на подбрусах не коррелируют с действительными областями значений. То есть, несмотря на то, что $\min\limits_{\boldsymbol{a}} f(x) > \min\limits_{\boldsymbol{b}} f(x)$, величина $\underline{\boldsymbol{f}(\boldsymbol{a})}$ часто оказывалась меньше, чем $\underline{\boldsymbol{f}(\boldsymbol{b})}$, где $\boldsymbol{a}$ и $\boldsymbol{b}$ — два каких-то подбруса из «рабочего списка»;

— эффект «застаивания интервальной оценки» (ситуация, когда существенное уменьшение области определения функции привело лишь к незначительному улучшению точности интервальной оценки) вместе с двумя указанными выше явлениями заставляет алгоритм делать очень много дроблений впустую, фактически не приближаясь к глобальному оптимуму;

— растущий размер рабочего списка дополнительно замедляет процесс оптимизации из-за особенностей архитектуры современных компьютеров.

Таким образом сделан вывод, что алгоритмы, основанные на адаптивном дроблении области определения в сочетании с интервальным оцениванием по получающимся подобластям, запрограммированы на неудачу в ряде задач. В соответствии со своей внутренней логикой они будет последовательно мельчить ложные ведущие брусы, лишь незначительно улучшая точность интервальной оценки.

На основе проделанного анализа в работе предложен и исследован ряд способов преодоления выявленных узких мест алгоритмов интервальной глобальной оптимизации и повышения их производительности [23]. В рассматриваемом нами аспекте интервальные оптимизационные алгоритмы с достаточной полнотой практически никем ранее не изучались. Новыми являются также выводы о сравнительной эффективности различных модификаций интервальных алгоритмов глобальной оптимизации, процедур отбраковки[3] и т. д.

Глава 3 является центральной в работе и посвящена разработке, реализации, тестированию и сравнительному анализу интервальных алгоритмов глобальной оптимизации, основанных на использовании стохастики и рандомизации. Вначале рассматривается реализация случайного интервального поиска. Мы анализируем и предлагаем возможные модификации, а также развиваем на этом пути новые подходы — интервальные эволюционный а также мультиметодный интервальный алгоритмы, сочетающий в себе положительные стороны большинства из составляющих его вычислительных схем.

В главе 3 описаны следующие новые интервальные оптимизационные алгоритмы, основанные на привлечении стохастики (рандомизации):

- алгоритм бисекции с перекрытием,
- случайный интервальный поиск,
- варианты дробления в случайной пропорции и направлении,
- случайный интервальный поиск с приоритетом,
- интервальный алгоритм имитации отжига,
- несколько вариантов интервального биологического алгоритма,
- адаптивный мультиметодный алгоритм,

[3]Интервальные критерии и методы, позволяющие наверняка утверждать, что глобальный оптимум недостижим на некотором брусе.

- оптимизированный адаптивный параллельный алгоритм глобальной интервальной оптимизации.

Простейшим из алгоритмов, сочетающих интервальную технику оценивания значений функций со стохастическим управлением, является алгоритм случайного интервального дробления, на каждой итерации выбирающий из рабочего списка для дробления случайный брус. У него есть как преимущества перед детерминистским аналогом, так и существенные недостатки. Вычислительные эксперименты показали, что такой метод неэффективен.

Дальнейшим развитием алгоритма «случайного интервального дробления» является алгоритм, названный «случайным интервальным дроблением с приоритетом». Существенным отличием от предшественника является модифицированная процедура выбора очередного кандидата на дробление. Выбор по-прежнему происходит случайно, но некоторые, например, более широкие брусы имеют бóльшую вероятность быть раздробленными.

В диссертации для этого алгоритма доказана следующая теорема.

Теорема 1 *Алгоритм случайного интервального дробления с приоритетом сходится к глобальному оптимуму почти наверное (почти всегда).*

Далее в работе подробно рассмотрена интервальная версия алгоритма имитации отжига, предложенная С.П. Шарым в 2005 году[4]. За основу взят популярный классический (точечный) метод оптимизации (известный также как «алгоритм Метрополиса»[5]), моделирующий физические процессы отжига или кристаллизации и хорошо зарекомендовавший себя для многоэкстремальных проблем с большим количеством возможных решений. Упрощённый алгоритм приведён на врезке 1. Алгоритм оперирует таким понятием как «температура». В нашем случае, чем она выше, тем больше вероятность того, что на определенном шаге для уточнения оптимума будет выбран брус, не доставляющий рекордную[6] на данный момент оценку оптимума. Таким образом, чем больше величина T, тем больше «рысканье» алгоритма по области определения. По мере работы алгоритма температура понижается и «блуждания» прекращаются.

[4]Шарый С.П. Стохастические подходы в интервальной глобальной оптимизации // *Труды XIII Байкальской международной школы-семинара «Методы оптимизации и их приложения». Том 4 «Интервальный анализ»*. Иркутск, ИСЭМ СО РАН, 2005. С. 85–105.

[5]Lambert M.S., Miriam T.T., Susan F.M. *Simulated Annealing: Global Optimization, Applied Mathematics, Global Optimum, Metropolis-Hastings Algorithm, Monte Carlo Method, Nicholas Metropolis, Quantum Annealing.* – Betascript Publishers, 2009.

[6]Минимальную при поиске минимума, максимальную при поиске максимума. Такой брус ещё называют ведущим.

В процессе работы алгоритма величина T постепенно уменьшается от некоторого начального значения T_0 до конечного T_{fin}. Заметим, что в классическом (точечном) случае выбор стратегии уменьшения T является очень важным вопросом, так как от этого напрямую зависит сходимость метода к глобальному оптимуму. Иными словами, в отличие от интервального аналога классический алгоритм имитации отжига не гарантирует нахождение глобального оптимума.

Алгоритм 1. Схема простейшего интервального алгоритма «имитации отжига».

присваиваем $\boldsymbol{y} \leftarrow \boldsymbol{x}$ и $T \leftarrow T_0$;
назначаем целочисленную величину N_T
— «количество испытаний на один температурный уровень»;
вычисляем $\boldsymbol{f}(\boldsymbol{y})$ и инициализируем список $\mathcal{L}$ записью $\{ (\boldsymbol{y}, \underline{\boldsymbol{f}(\boldsymbol{y})}) \}$;
DO WHILE $(T > T_{fin})$
DO FOR $j = 1$ TO N_T
случайно выбираем из $\mathcal{L}$ запись $(\boldsymbol{z}, \underline{\boldsymbol{f}(\boldsymbol{z})})$ по правилу $\mathsf{S}(\boldsymbol{y})$;
DO (с вероятностью $\mathsf{P}_T(\boldsymbol{y}, \boldsymbol{z})$)
рассекаем $\boldsymbol{z}$ по самой длинной компоненте пополам
на брусы-потомки $\boldsymbol{z}'$ и $\boldsymbol{z}''$;
вычисляем $\boldsymbol{f}(\boldsymbol{z}')$ и $\boldsymbol{f}(\boldsymbol{z}'')$;
удаляем запись $(\boldsymbol{z}, \underline{\boldsymbol{f}(\boldsymbol{z})})$ из списка $\mathcal{L}$;
помещаем записи $(\boldsymbol{z}', \underline{\boldsymbol{f}(\boldsymbol{z}')})$ и $(\boldsymbol{z}'', \underline{\boldsymbol{f}(\boldsymbol{z}'')})$ в список $\mathcal{L}$;
обозначаем через $(\boldsymbol{y}, \underline{\boldsymbol{f}(\boldsymbol{y})})$ ту из записей $(\boldsymbol{z}', \underline{\boldsymbol{f}(\boldsymbol{z}')})$
и $(\boldsymbol{z}'', \underline{\boldsymbol{f}(\boldsymbol{z}'')})$, которая имеет меньшее значение
второго поля;
END DO
END DO
уменьшаем значение температуры $T \leftarrow \alpha T$
END DO
$f^* \leftarrow \underline{\boldsymbol{f}(\boldsymbol{y})}$;

Для интервального алгоритма имитации отжига в работе доказано следующее утверждение.

Утверждение 1 В процессе работы предложенного интервального алгоритма имитации отжига при любом законе предельного перехода $T \to 0$ существуют такие натуральные числа i_{max} и n, что любой брус $\boldsymbol{x}'$ из рабочего списка $\boldsymbol{\mathcal{L}}$, удовлетворяющий равенству $\underline{\boldsymbol{f}(\boldsymbol{x}')} = \min_{1 \le i \le S} \underline{\boldsymbol{f}(\boldsymbol{x}_i)}$, начиная с шага с номером $i_{\max}$ подвергается дроблению не реже чем одного раза в продолжение любых n последовательных шагов алгоритма.

Опираясь на это утверждение доказана теорема о сходимости:

Теорема 2 *Интервальный алгоритм имитации отжига сходится к глобальному оптимуму.*

Дальнейшим развитием идеи интервальных стохастических алгоритмов стало обобщение для интервального случая алгоритмов «биологического» типа.

Биологические алгоритмы это эвристические алгоритмы поиска, так или иначе использующие для решения задач оптимизации случайный подбор, комбинирование и вариации входных параметров с использованием механизмов, напоминающих биологическую эволюцию. Идеи применить биологические механизмы или, скорее, принципы их функционирования для создания новых алгоритмов, возникали в нескольких модификациях практически одновременно у ряда авторов в шестидесятых годах XX века. Уже первые работы показали, что идея использовать принципы биологической эволюции оказалась плодотворной. В настоящее время множество разновидностей биологических алгоритмов (в основном «генетические» и «поведенческие») используются в том числе и для решения задачи глобальной оптимизации. Однако, несмотря на то, что различные биологические алгоритмы продемонстрировали свою эффективность для решения оптимизационных задач и, тем самым, сыскали себе заслуженную популярность, интервальных биологических алгоритмов пока предложено не было.

Алгоритмы, использующие принципы биологической эволюции можно условно разделить на *эволюционные*, *поведенческие* и *генетические*.

Алгоритмы глобального случайного поиска, построенные на принципе биологической эволюции, моделируют механизм естественного отбора (выживает и даёт потомство наиболее приспособленный). Поведенческие алгоритмы случайного поиска моделируют поведение живых организмов, при котором имитируется присущая живым организмам способность стремиться к комфортным условиям и уклоняться от некомфортных. Для целей интервальной глобальной оптимизации нам показался наиболее удобным эволюционный подход.

Идея алгоритма состоит в следующем. Каким-либо образом (чаще всего случайно) задаётся начальная «*популяция*» — некое множество объектов. Они оцениваются с использованием «*функции приспособленности*», в результате чего каждому объекту присваивается определённое значение (*приспособленность*), которое определяет вероятность выживания организма, представленного данным объектом. После этого с использованием полученных

значений приспособленности выбираются объекты (производится *селекция*), допущенные к «*размножению*». Также к этим объектам могут применяться «*генетические операторы*». Особи следующего поколения также оцениваются, затем снова производится *селекция*, применяются «генетические операторы» и т. д. Так моделируется «*эволюционный процесс*», продолжающийся несколько жизненных циклов («*поколений*») до тех пор, пока не будет выполнен критерий остановки алгоритма.

В описанной схеме применение генетических операторов не обязательно. Их использование позволяет повысить рандомизацию метода, увеличить вариабельность популяции и избежать застоя вблизи локальных оптимумов. Кроме того, в классических генетических алгоритмах операции мутации и скрещивания могут порождать новые решения, которые никогда не встречались в предыдущих поколениях. Интервальный алгоритм, описанный в рамках настоящей работы их не использует. Тем не менее, в отличии от классических вариантов он позволяет гарантированно найти именно глобальный оптимум. Для сохранения доказательности, являющейся характерной чертой интервальных методов, мы не отбрасываем никакие подобласти исходной области поиска из рассмотрения до тех пор, пока не будет доказано что они гарантированно не содержат оптимум. Подробнее о различных подходах к выявлению бесперспективности брусов (их называют ещё «критериями отбраковки»), можно прочитать, например, в [54, 37]. Таким образом, в алгоритме особи либо рождаются нежизнеспособными (не выдерживают критериев отбраковки применяемых сразу после дробления) или погибают в результате «*эпидемий*» — срабатывания уточненного критерия отбраковки.

В интервальном эволюционном алгоритме исходная область поиска разбивается на непересекающиеся подобласти — брусы. Каждый такой брус в описываемой здесь разновидности эволюционного алгоритма выступает в качестве «особи». В качестве меры приспособленности особи можно выбрать нижнюю границу (для определённости рассматриваем ситуацию поиска минимума) интервальной оценки целевой функции. Чем лучше мера приспособленности (чем меньше нижняя граница в привычных терминах), тем больше шансов у данной особи размножиться (данному брусу быть раздробленным) и тем многочисленнее будет потомство (брус может быть раздроблен на бóльшее количество подбрусов).

В приведённом примере в качестве меры приспособленности мы использовали нижнюю границу интервальной оценки функции на подбрусе (оценку оптимума на подбрусе снизу). Вообще, использование интервальных объектов позволяет использовать в качестве меры приспособленности ширину ин-

Алгоритм 2. Схема интервального биологического алгоритма.

Задаётся начальная популяция и передаётся в основной цикл
Основной цикл

1. **Вычислить значения функции приспособленности новорожденных особей** (этот шаг включает вычисление интервального расширения целевой функции по новым подбрусам, как необходимое для определения приспособленности подбруса).
2. N **из наиболее приспособленных брусов с вероятностью** P_n **порождают от** L_n **до** U_n **потомков.**
3. M **из неприспособленных брусов с вероятностью** P_m **порождают от** L_m **до** U_m **потомков.**
4. **Потомки проверяются на жизнеспособность** (применяются интервальные критерии отбраковки).
5. **Если критерий отбраковки был улучшен, возможно случается эпидемия** (улучшенные критерии применяются ко всем особям).

тервальной оценки или же размер бруса. Несмотря на то, что все три метода очень похожи и критерии, которыми они руководствуются, направлены на достижение одной и той же цели, работают они по-разному. Исследовав их влияние на алгоритм, нами была предложена следующая объединённая функция приспособленности бруса.

$$Fit\,(\boldsymbol{b}) = \sum_{i=1}^{n} \big(\alpha_i \cdot \mathrm{wid}\ \boldsymbol{b}_i\big) + \beta \cdot \underline{\boldsymbol{f}(\boldsymbol{b})} + \gamma \cdot \mathrm{wid}\ \boldsymbol{f}(\boldsymbol{b}). \tag{3}$$

Приведён её вид для поиска минимума. Здесь

$\boldsymbol{f}(\boldsymbol{b})$ — интервальная оценка значений целевой функции f на брусе $\boldsymbol{b}$;

$\underline{\boldsymbol{f}(\boldsymbol{b})}$ — нижняя граница интервальной оценки (оценка минимума снизу);

$\mathrm{wid}\ \boldsymbol{f}(\boldsymbol{b})$ — ширина (точность) интервальной оценки области значений;

$\mathrm{wid}\ \boldsymbol{b}_i$ — ширина i-ой стороны бруса области определения;

$\alpha_1, \ldots, \alpha_n, \beta, \gamma$ — соответствующие весовые коэффициенты.

Для интервального эволюционного алгоритма в работе сформулированы и доказаны следующие утверждения.

Утверждение 2 В ходе работы интервального эволюционного алгоритма границы интервальной оценки оптимума монотонно сужаются.

Утверждение 3 Интервальный эволюционный алгоритм не упускает из рассмотрения глобальные оптимумы.

На основе этих результатов в диссертации доказана теорема:

Теорема 3 *Интервальный генетический алгоритм гарантированно сходится к глобальному оптимуму.*

В работе исследована сравнительная эффективности разработанных алгоритмов. В большинстве случаев, особенно не имея априорной информации о целевой функции, бывает невозможно указать заранее, какой алгоритм справится лучше с той или иной конкретной проблемой. Нами выработан способ, позволяющий отказаться от предварительного тестирования эффективности алгоритмов. Идея заключается в том, чтобы запускать все алгоритмы решения (в том числе параллельно), при этом динамически отслеживая эффективность каждого алгоритма и давая больше вычислительных ресурсов более успешным алгоритмам.

Конкретизация этой идеи следующая. Запускается какой-то алгоритм глобальной оптимизации из заданного семейства. Через короткий промежуток времени мы останавливаем его и на существующем рабочем списке брусов запускается уже другой алгоритм, т. е., фактически, происходит подмена алгоритма оптимизации в процессе поиска оптимума. При этом отслеживается относительную эффективность каждого из алгоритмов — сколько итераций было сделано, насколько улучшилась оценка оптимума, как изменился размер рабочего списка и т. п. По каждому алгоритму отслеживается как глобальная статистика, то есть результаты за всё время работы алгоритма, так и локальная — результаты каждого конкретного сеанса работы. Исходя их сравнительной успешности алгоритмов динамически регулируется квант времени, выделяемый каждому из алгоритмов, таким образом обеспечивая оптимальность их применения. Таким образом, нами реализуется «мета-алгоритм», адаптивный алгоритм второго порядка (по аналогии с терминологией, принятой в функциональных языках программирования для функций, оперирующих функциями), или мультиметодный подход по терминологии, берущей начало с работ А.И. Тятюшкина и А.Ю. Горнова. На рис. 2 показан пример работы такого алгоритма для случая трёх алгоритмов глобальной оптимизации. Для простоты показана схема, в которой решение, какой алгоритм будет признан перспективным для следующего шага, принимается только на основе информации с предыдущего шага. То есть, анализ суммарной успешности алгоритмов за всё время работы не проводится.

На основном графике показана зависимость ширины интервальной оценки оптимума целевой функции от времени. По оси времени пунктиром отмечены моменты переключения между алгоритмами. Работающий на каждом из этапов алгоритм помечен соответствующей цифрой над графиком. Угол наклона графика на каждом из шагов и есть, фактически, эффективность соот-

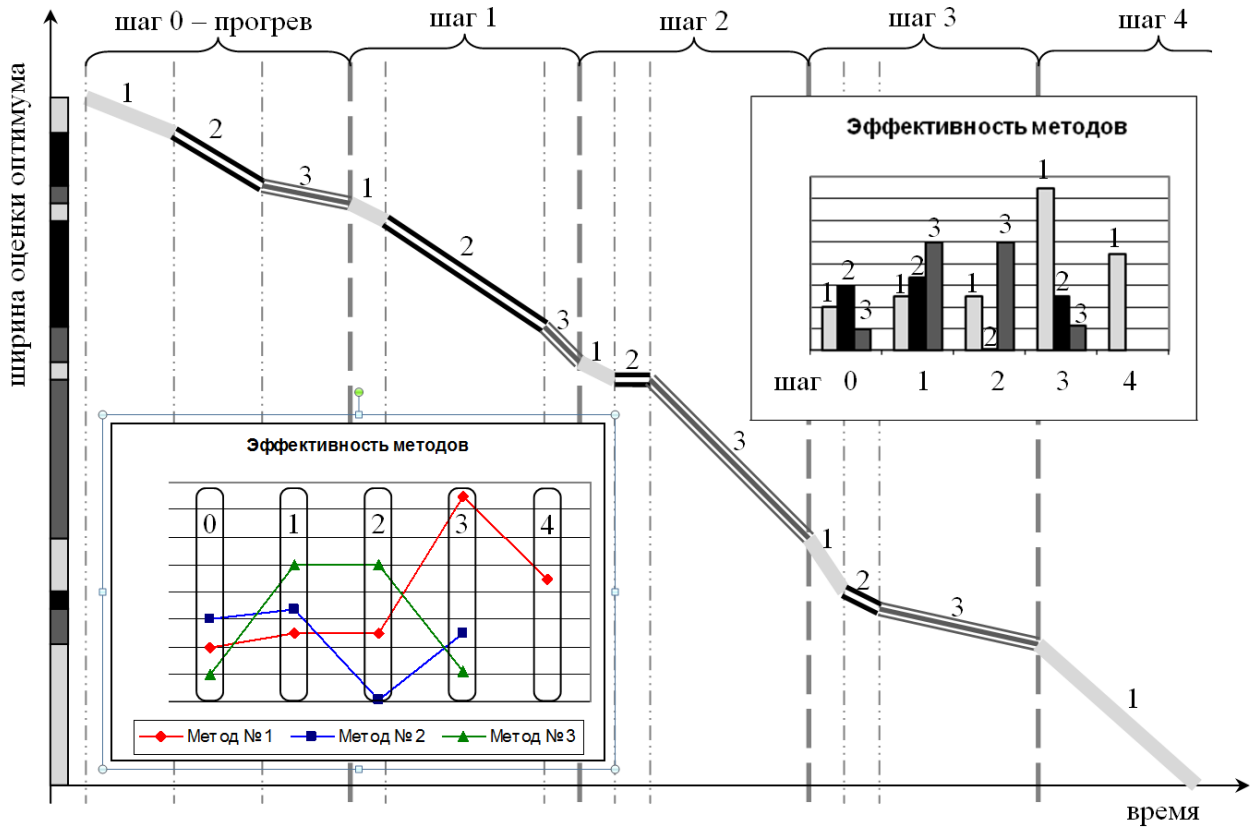


Рис. 2: Пример переключения между алгоритмами

ветствующего метода на этом шаге. Эффективность каждого из методов и её динамика по шагам представлены на двух меньших графиках. На начальном этапе (стадия прогрева) алгоритмам предоставляется некоторое, одинаковое для всех, время работы. По результатам выбирается наиболее успешный, то есть тот, которому удалось улучшить оценку оптимума более других. В приведённой иллюстрации это будет алгоритм номер два. На следующей итерации ему будет выделено больше времени для работы. В показанной реализации всем остальным алгоритмам будет предоставлено одинаковое небольшое время для работы. На следующем шаге относительная успешность алгоритмов снова сравнивается и выделяемое время корректируется.

В работе приводятся оценки скорости работы мультиметодного алгоритма и делается вывод об его эффективности.

Основные итоги главы 3 могут быть сформулированы следующим образом:

1) Продемонстрирована вычислительная эффективность стохастических (рандомизированных) алгоритмов интервальной глобальной оптимизации. в сравнении с детерминистскими аналогами на ряде общеупотребительных тестовых задач;

2) модифицированы существующие стохастические (рандомизированные) алгоритмы интервальной глобальной оптимизации, такие как случайное интервальное дробление. Предложены и исследованы некоторые новые алгоритмы, в частности, интервальное дробление в случайной пропорции и направлении;
3) разработаны и экспериментально исследованы различные версии интервального эволюционного алгоритма для доказательной глобальной оптимизации;
4) для предложенных алгоритмов сформулирован и доказан ряд утверждений и теорем о сходимости разработанных алгоритмов. В частности, доказано, что алгоритм интервального случайного поиска с приоритетом сходится к глобальному оптимуму почти всегда (Сходимость «почти всегда» сильнее «сходимости по вероятности»); а интервальный алгоритм имитации отжига и интервальный генетический алгоритм гарантированно сходятся к глобальному оптимуму.
5) разработан и апробирован интервальный мультиметодный алгоритм глобальной оптимизации, созданный на основе стохастических интервальных подходов. Приведены теоретические оценки эффективности и времени работы алгоритма, в том числе выделен и проанализирован худший для алгоритма случай.

Кроме того, в главе 3 проработаны вопросы параллелизма предлагаемых алгоритмов и их эффективной реализации на современных вычислительных системах.

Глава 1

Инструменты интервального анализа

1.1 Основы интервального анализа

Определение 1.1 *Одномерные интервалы — замкнутые отрезки вещественной оси* $\mathbb{R}$. *Далее в тексте они обозначаются жирным шрифтом —* $\boldsymbol{A}$, $\boldsymbol{B}$, $\ldots$, $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$. *При этом* $\boldsymbol{x} = [\underline{\boldsymbol{x}}, \overline{\boldsymbol{x}}]$, *а* $\underline{\boldsymbol{x}}$ *и* $\overline{\boldsymbol{x}}$ *называются, соответственно, левым (или нижним) и правым (или верхним) концами интервала.*

Формально всё вышесказанное можно записать как

$$\boldsymbol{x} = [\underline{\boldsymbol{x}}, \overline{\boldsymbol{x}}] = \{\, x \in \mathbb{R} \mid \underline{\boldsymbol{x}} \leqslant x \leqslant \overline{\boldsymbol{x}} \}.$$

Для $\boldsymbol{x} = [\underline{\boldsymbol{x}}\,, \overline{\boldsymbol{x}}]$ пишут $\boldsymbol{x} \geqslant 0$, если $\underline{\boldsymbol{x}} \geqslant 0$ и говорят о неотрицательном интервале. Соответственно, пишут что $\boldsymbol{x} \leqslant 0$, если $\underline{\boldsymbol{x}} \leqslant 0$ и говорят, что интервал неположителен.

Если нижняя граница равна верхней, то интервал содержит только одно число и называется *вырожденным*.

Если $0 \in \boldsymbol{x}$, то интервал называют нульсодержащим.

Определение 1.2 *Многомерные интервалы (также называемые брусами) определяются как вектор (вектор-столбец или вектор-строка) с интервальными компонентами.*

Формально говоря, интервальный вектор — это упорядоченный кортеж интервалов. По-сути он является прямым произведением одномерных интервалов, так что его геометрическим образом является прямоугольный параллелепипед (брус) в $\mathbb{R}^n$ со сторонами, параллельными координатным осям.

Одним из основных инструментов интервального анализа являются так называемые *интервальные арифметики* — алгебраические системы, формализующие операции между интервалами. Наиболее популярная из них — *классическая интервальная арифметика.*

Определение 1.3 *Классическая интервальная арифметика — алгебраическая система* $\langle \mathbb{IR}, +, -, \cdot, / \rangle$, *носителем которой является множество всех вещественных интервалов, а операции сложения, вычитания, умножения и деления определяются следующим образом:*

$$\boldsymbol{x} * \boldsymbol{y} = \{x * y \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\} \qquad \text{для } * \in \{+, -, \cdot, /\}.$$

Иными словами, результирующий интервал любой операции есть множество всевозможных результатов этой операции между точками-представителями интервалов-операндов. Явные формулы для интервальных сложения, вычитания, умножения и деления выглядят следующим образом:

$$\begin{aligned}
\boldsymbol{x} + \boldsymbol{y} &= [\underline{\boldsymbol{x}} + \underline{\boldsymbol{y}}, \overline{\boldsymbol{x}} + \overline{\boldsymbol{y}}], \\
\boldsymbol{x} - \boldsymbol{y} &= [\underline{\boldsymbol{x}} - \overline{\boldsymbol{y}}, \overline{\boldsymbol{x}} - \underline{\boldsymbol{y}}], \\
\boldsymbol{x} \cdot \boldsymbol{y} &= [\min\{\underline{\boldsymbol{x}} \cdot \underline{\boldsymbol{y}}, \underline{\boldsymbol{x}} \cdot \overline{\boldsymbol{y}}, \overline{\boldsymbol{x}} \cdot \underline{\boldsymbol{y}}, \overline{\boldsymbol{x}} \cdot \overline{\boldsymbol{y}}\}, \max\{\underline{\boldsymbol{x}} \cdot \underline{\boldsymbol{y}}, \underline{\boldsymbol{x}} \cdot \overline{\boldsymbol{y}}, \overline{\boldsymbol{x}} \cdot \underline{\boldsymbol{y}}, \overline{\boldsymbol{x}} \cdot \overline{\boldsymbol{y}}\}], \\
1 \,/\, \boldsymbol{y} &= [1/\overline{\boldsymbol{y}},\ 1/\underline{\boldsymbol{y}}], \qquad \text{для} \quad \boldsymbol{y} \not\ni 0.
\end{aligned} \tag{1.1}$$

Если один из операндов невырожден, т. е. имеет ненулевую ширину, то и результатом арифметической операции является невырожденный интервал. Отсюда, в частности, следует, что для невырожденного интервала $\boldsymbol{x}$ не существует обратных по сложению и умножению элементов, так как если $\boldsymbol{x} + \boldsymbol{y} = 0$, $\boldsymbol{y} \cdot \boldsymbol{z} = 1$, то $\boldsymbol{x}, \boldsymbol{y}$ и $\boldsymbol{z}$ должны быть, в силу вышеизложенного, вырожденными. Иными словами, вычитание не обратно сложению, деление не обратно умножению. В отличие, например, от поля вещественных чисел, где операции вычитания и деления вводятся как обратные к сложению и умножению. То есть в общем случае

$$(\boldsymbol{a} + \boldsymbol{b}) - \boldsymbol{b} \neq \boldsymbol{a}, \qquad (\boldsymbol{a} \cdot \boldsymbol{b})/\boldsymbol{b} \neq \boldsymbol{a}. \tag{1.2}$$

Кроме того,

$$\begin{aligned}
\boldsymbol{a} - \boldsymbol{a} &\neq 0, \\
\boldsymbol{a}/\boldsymbol{a} &\neq 1,
\end{aligned} \tag{1.3}$$

что означает, что $\mathbb{IR}$ не является полем, так как его элементы не имеют обратных относительно сложения и умножения.

Напомним, что полем называется множество с двумя бинарными операциями — «+» (аддитивная операция или сложение) и «·» (мультипликативная операция или умножение), если оно (вместе с этими операциями) образует коммутативное ассоциативное кольцо с единицей, все ненулевые элементы которого обратимы.

Вместо (1.3) действуют два других, более слабых правила сокращения:

$$\begin{aligned} \boldsymbol{a}+\boldsymbol{b}=\boldsymbol{a}+\boldsymbol{c} \;&\Rightarrow\; \boldsymbol{b}=\boldsymbol{c}, \\ \boldsymbol{a}\boldsymbol{b}=\boldsymbol{a}\boldsymbol{c} \;&\Rightarrow\; \boldsymbol{b}=\boldsymbol{c}. \end{aligned} \tag{1.4}$$

Нейтральными элементами относительно сложения и умножения в $\mathbb{IR}$ являются соответственно нуль и единица, кроме того, интервальные арифметические операции, определённые в (1.1), обладают следующими свойствами:

$$(\boldsymbol{a}+\boldsymbol{b})+\boldsymbol{c}=\boldsymbol{a}+(\boldsymbol{b}+\boldsymbol{c}) \text{ — ассоциативность сложения,} \tag{1.5}$$

$$(\boldsymbol{a}\cdot\boldsymbol{b})\cdot\boldsymbol{c}=\boldsymbol{a}\cdot(\boldsymbol{b}\cdot\boldsymbol{c}) \text{ — ассоциативность умножения,} \tag{1.6}$$

$$\boldsymbol{a}+\boldsymbol{b}=\boldsymbol{b}+\boldsymbol{a} \text{ — коммутативность сложения,} \tag{1.7}$$

$$\boldsymbol{a}\boldsymbol{b}=\boldsymbol{b}\boldsymbol{a} \text{ — коммутативность умножения.} \tag{1.8}$$

Интервал $\boldsymbol{a}$ меньше либо равен интервала $\boldsymbol{b}$, тогда и только тогда, когда $\underline{\boldsymbol{a}} \leqslant \underline{\boldsymbol{b}}$ и $\overline{\boldsymbol{a}} \leqslant \overline{\boldsymbol{b}}$. В многомерном случае это правило применяется покомпонентно:

$$\begin{gathered} \mathbb{IR}^n \ni \boldsymbol{a}=(\boldsymbol{a}_1,\boldsymbol{a}_2,\,\ldots\,,\boldsymbol{a}_n) < \boldsymbol{b}=(\boldsymbol{b}_1,\boldsymbol{b}_2,\,\ldots\,,\boldsymbol{b}_n)\in\mathbb{IR}^n \\ \Updownarrow \\ \boldsymbol{a}_i<\boldsymbol{b}_i \;\text{ для всех } i=\overline{1,n}. \end{gathered}$$

Для интервалов, являющихся по сути дела множествами, естественно также определить частичное упорядочение (порядок) по включению как

$$\boldsymbol{a}\subseteq\boldsymbol{b} \iff \underline{\boldsymbol{a}}\geqslant\underline{\boldsymbol{b}} \;\text{ и }\; \overline{\boldsymbol{a}}\leqslant\overline{\boldsymbol{b}}.$$

Расстояние dist между двумя интервалами $\boldsymbol{a}$ и $\boldsymbol{b}$ определяется как хаусдорфова метрика

$$\operatorname{dist}(\boldsymbol{a},\boldsymbol{b})=\max\bigl\{\,|\,\underline{\boldsymbol{a}}-\underline{\boldsymbol{b}}\,|,\,|\,\overline{\boldsymbol{a}}-\overline{\boldsymbol{b}}\,|\,\bigr\}.$$

Ширина интервала $\boldsymbol{a}$ естественно определяется как

$$\operatorname{wid}\boldsymbol{a}=\overline{\boldsymbol{a}}-\underline{\boldsymbol{a}}.$$

Особенностью интервальных арифметических операций (1.1) является отсутствие свойства дистрибутивности умножения относительно сложения (не выполняется ещё один признак поля). Так, в общем случае

$$\boldsymbol{a}(\boldsymbol{b}+\boldsymbol{c})\neq\boldsymbol{a}\boldsymbol{b}+\boldsymbol{a}\boldsymbol{c}. \tag{1.9}$$

Вместо этого классическая интервальная арифметика обладает более слабым свойством субдистрибутивности:

$$\boldsymbol{a}(\boldsymbol{b}+\boldsymbol{c}) \subseteq \boldsymbol{ab}+\boldsymbol{ac}. \tag{1.10}$$

Надо отметить, что в ряде частных случаев дистрибутивность всё-таки выполняется. Например:

$$\boldsymbol{a}(\boldsymbol{b}+\boldsymbol{c}) = \boldsymbol{a}\cdot\boldsymbol{b}+\boldsymbol{a}\cdot\boldsymbol{c},$$

если $\boldsymbol{b}\cdot\boldsymbol{c} \geqslant 0$, или

$$a(\boldsymbol{b}+\boldsymbol{c}) = a\boldsymbol{b}+a\boldsymbol{c},$$

если a — вещественное число.

Интересно заметить, что при этом $\boldsymbol{a}(b+c) \neq \boldsymbol{a}b+\boldsymbol{a}c$, даже если b и $c \in \mathbb{R}$. Например, положим $\boldsymbol{a} = [0,1],\ b = 1,\ c = -1$:

$$\begin{gathered}[0,1]\cdot(1-1) \neq [0,1]\cdot 1+[0,1]\cdot(-1),\\ [0,1]\cdot 0 \neq [0,1]+[-1,0],\\ 0 \neq [-1,1].\end{gathered}$$

Как следствие отсутствия дистрибутивности, пространство $\mathbb{IR}^n$ не является линейным векторным пространством, так как не выполняется одна из аксиом линейного векторного пространства:

$$(\mu+\nu)\boldsymbol{a} \neq \mu\boldsymbol{a}+\nu\boldsymbol{a}.$$

Интервальные арифметические операции обладают важным свойством *монотонности по включению*, то есть

$$\boldsymbol{a}\subseteq\boldsymbol{a}', \boldsymbol{b}\subseteq\boldsymbol{b}' \ \Rightarrow\ \boldsymbol{a}*\boldsymbol{b}\subseteq\boldsymbol{a}'*\boldsymbol{b}' \qquad \text{для } * \in \{+,-,\cdot,/\}. \tag{1.11}$$

Таким образом, если $\boldsymbol{f}(\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_n)$ является рациональным выражением от интервальных переменных $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, то есть конечной комбинацией интервалов $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ и конечного количества констант, соединенных интервальными арифметическими операциями, то из $\boldsymbol{x}_i^1 \subset \boldsymbol{x}_i^2,\ i=\overline{1,n}$, следует, что

$$\boldsymbol{f}(\boldsymbol{x}_1^1,\boldsymbol{x}_2^1,\ldots,\boldsymbol{x}_n^1) \subset \boldsymbol{f}(\boldsymbol{x}_1^2,\boldsymbol{x}_2^2,\ldots,\boldsymbol{x}_n^2)$$

при любом наборе аргументов $\boldsymbol{x}^i$, для которого интервальные арифметические операции в выражении $\boldsymbol{f}$ имеют смысл (то есть не встречается деление на интервал, содержащий нуль).

1.2 Автоматическое дифференцирование

Из математического анализа хорошо известно определение производной функции $y = f(x)$ по переменной x в точке x_0, как предела отношения приращения функции $\Delta y = f(x_0 + \Delta x) - f(x_0)$ к соответствующему приращению аргумента Δx при $\Delta x \to 0$, т. е.

$$f'(x) = \lim_{\Delta x \to 0} \left(\frac{\Delta f(x)}{\Delta x} \right) = \lim_{\Delta x \to 0} \left(\frac{f(x + \Delta x) - f(x)}{\Delta x} \right). \qquad (1.12)$$

Производные играют важнейшую роль в исследовании поведения функций и при нахождении их экстремумов, т. е. в теории и практике оптимизации. Весьма широко используются производные и в интервальных методах, в особенности при решении нелинейных задач. В связи с этим большое значение приобретают способы нахождения производных от функций, и соответствующие алгоритмы часто используются в различных оптимизационных процедурах. Как показывается в курсах дифференциального исчисления (см., к примеру, [67]), дифференцирование функций, заданных явными аналитическими выражениями, может быть сведено к весьма рутинному процессу, подчиняющемуся строго заданным правилам, так что его естественно поручить вычислительной машине.

В современной вычислительной практике широко применяют три подхода к нахождению производных:

— символьное,

— численное,

— и автоматическое

дифференцирование.

Символьными называют такие вычисления, результаты которых представляются в аналитическом виде. Вычисления в символьном виде отличаются большей общностью и позволяют судить о математических, физических и иных закономерностях решаемых задач.

Задача машинного нахождения производной в символьном виде в свое время положила начало целому направлению исследований, получившему название «символьные вычисления» или «компьютерная алгебра».

Фактически, задача символьного дифференцирования сводится к синтаксическому разбору выражения с последовательным применением определенных правил. Кроме того, при дифференцировании эти правила достаточно строги и их немного. Листинг 3 представляет программу из 40 строк на языке

Лисп (в свое время задача машинного нахождения производной в символьном виде являлась одной из мотиваций создания нового языка Лисп как языка обработки символов [66]), способную найти производную любой элементарной функции.

Однако результаты её работы выглядят довольно неожиданно. Попробуем простой пример: найдём производную функции x^2 относительно x:

```
(diff '(expt x 2) 'x)
(+ (* 1 2 (expt x (- 2 1))) (* 0 (expt x) (log x)))
```

Вероятно, потребуется некоторое время, чтобы понять, что это выражение действительно эквивалентно $2{\cdot}x$. Программа «не знает» что $2{\cdot}1 = 2$, $2{-}1 = 1$, $0 \cdot x = 0$ и тому подобное. То есть, она не может упростить выражение. В действительности, упрощение — одна из главных проблем в компьютерной алгебре [8, 83, 127].

Конечно, можно было бы дополнить нашу программу набором правил упрощения выражений и добиться приемлемой формы ответов. Проблема в том, что если нашей целью является получать ответы в «самом простом» виде, то нам придется полностью пересмотреть способ представление данных и очень сильно усложнить логику программы [76, 84].

При этом надо помнить, что в интервальных алгебрах форма записи функции, её вид существенно влияют на точность интервального расширения. Например, в общем случае, $\boldsymbol{x}^2 \neq \boldsymbol{x} \cdot \boldsymbol{x}$. Так, $\boldsymbol{x}^2 = [0, 4]$ на интервале $[-2, 1]$, в то время как $\boldsymbol{x} \cdot \boldsymbol{x} = [-2, 4]$. Подробнее этот вопрос обсуждается в §1.4.

1.2.1 Численное дифференцирование

Численное дифференцирование позволяет *приближённо* вычислить производные функции по заданным в конечном числе точек значениям этой функции [24].

Один из универсальных способов построения формул численного дифференцирования основан на аппроксимации или интерполяции функции полиномом. Так, по заданным значениям функции $f(x)$ в некоторых узлах $x_0, x_1, \cdots, x_N$ строят интерполяционный полином $P_N(x)$ (обычно в форме Лагранжа) и приближенно полагают

$$f^{(r)}(x) \approx P_N^{(r)}(x), \qquad 0 < r < N.$$

В ряде случаев, например, используя формулу Тейлора, удается получить точное равенство вида

$$f^{(r)}(x) = P_N^{(r)}(x) + R, \qquad 0 < r < N$$

Листинг 3. Код алгоритма символьного автоматического дифференцирования

```
(define (diff expr x)

   (define (dx t)
      (if (pair? t)
          (if (null?  (cddr t))
              (unary  (car t) (cadr t)))
              (binary (car t) (cadr t)(caddr t))))
          (if (eq? t x) 1 0)))

   (define (binary f u v)
      (cond
         ((eq? f '+) (list '+ (dx u) (dx v)))
         ((eq? f '-) (list '- (dx u) (dx v)))
         ((eq? f '*) (list '+
                           (list '* (dx u) v)
                           (list '* (dx v) u)))
         ((eq? f '/) (list '/
                           (list '-
                                 (list '* (dx u) v)
                                 (list '* (dx v) u))
                           (list 'expt v 2)))
         ((eq? f 'expt)(list '+
                             (list '*
                                   (dx u) v
                                   (list 'expt u (list '- v 1)))
                             (list '*
                                   (dx v)
                                   (list 'expt u)
                                   (list 'log u))))
         (else (error "неизвестная операция"))))

   (define (unary f u)
      (list '*
            (dx u)
            (cond ((eq? f 'exp) (list 'exp u))
                  ((eq? f 'log) (list '/ u))
                  ((eq? f 'sin) (list 'cos u))
                  ((eq? f 'cos) (list '- (list 'sin u)))
                  (else (error "неизвестная операция")))))

   (dx expr))
```

содержащее остаточный член R, называемый *погрешностью численного дифференцирования*. Результатом классического численного дифференцирования является число, *приближённо* равное значению производной в окрестности точки. Хорошо известно, что операция численного дифференцирования является некорректной математической задачей, и при сближении узлов её результат становится всё менее устойчивым к ошибкам расчётов [5].

«Лобовое» применение подобного вычисления производной для интервальнозначной функции, когда требуется делить интервал внешней оценки приращения функции на интервал изменения аргумента, фактически, равносильно внесению большой погрешности в исходные данные. При ширине интервала, стремящейся к нулю, по определению производной и в силу соотношения (1.12) мы получим чрезвычайно неустойчивый результат, качество которого будет совершенно неприемлемым при близкоотстоящих узлах.

1.2.2 Наклоны функции

Пусть вещественнозначную функцию от n переменных $f : \mathbb{R}^n \to \mathbb{R}$ нам удалось представить в виде

$$f(x) = f(\tilde{x}) + f^{\angle}(\tilde{x}, x)(x - \tilde{x}), \tag{1.13}$$

где $f^{\angle}(\tilde{x}, x)$ мы будем называть наклоном функции f между точками $\tilde{x}$ и x при заданных $\tilde{x}, x$.

В одномерном случае

$$f^{\angle}(\tilde{x}, x) = \frac{f(x) - f(\tilde{x})}{x - \tilde{x}}.$$

Таким образом, в одномерном случае наклон функции совпадает с её разделенной разностью первого порядка. Предполагая непрерывность функции наклона, получаем

$$f^{\angle}(\tilde{x}, \tilde{x}) = f'(\tilde{x}).$$

В многомерном неинтервальном случае наклон определяется неоднозначно. Интервальный наклон не есть интервальное расширение обычного наклона, но множество всех представителей наклонов функции между точками x и y, когда y пробегает заданный брус $\boldsymbol{y}$.

Определение 1.4 *Интервальным наклоном функции $f : \mathbb{R}^n \to \mathbb{R}$ на интервале $\boldsymbol{y} \in \mathbb{IR}^n$ относительно точки x называется интервальный вектор $\boldsymbol{s} \in \mathbb{IR}^n$, такой что для любого $y \in \boldsymbol{y}$ существует $s \in \boldsymbol{s}$, являющийся наклоном функции $f(x)$ между точками x и y.*

Мы можем использовать интервальные наклоны функций следующим образом. Если разложение (1.13) справедливо для всех x из некоторой области $\boldsymbol{x}$, а наклон $f^{\angle}(\cdot,\cdot)$ может быть выражен как некоторое арифметическое выражение от своих аргументов, то мы можем найти внешнюю оценку области значений функции f на брусе $\boldsymbol{x}$ как

$$f_{sl}(\boldsymbol{x},\tilde{x}) = f(\tilde{x}) + f^{\angle}(\tilde{x},\boldsymbol{x})(\boldsymbol{x}-\tilde{x}). \tag{1.14}$$

Такое интервальное расширение функции принято называть *наклонным*. К теме наклонов и наклонному расширению функций мы вернёмся в §1.3.3. Там же мы поговорим об их точности и сравним с другими формами интервальных расширений, которые будут введены в следующих параграфах.

1.2.3 Автоматическое дифференцирование

По-видимому, первыми этот способ предложили Кахримэниан [96] и Нолан [111], опубликовавшие свои результаты в 1953 г. в США, а также Л.М. Беда и др. [6], создавшие программу автоматического дифференцирования для компьютера БЭСМ в 1959 г. в СССР. Определяющее влияние на работы по этому направлению оказала статья Л.В. Канторовича [26], в которой был предложен формализм для представления любого выражения в виде последовательности тетрад, состоящих из операндов, бинарных и унарных операций.

Для нужд интервального анализа этот способ вычисления производной весьма удобен. Он не вызывает дополнительных сложностей с упрощением полученных символьных выражений, сравнительно несложно реализуется программно и позволяет вычислять интервальные оценки производных даже на интервалах. Поэтому на нём мы остановимся подробнее.

Рассмотрим для примера функцию

$$F_{Tre4}(x,y) = x^4 + 4x^3 + 4x^2 + y^2, \tag{1.15}$$

также известную как «функция Треккани» (Treccani function, Tre4) [90]. Эту функцию мы ещё не раз будем использовать для иллюстративных примеров, и вид её графика можно посмотреть на рис. 3.2 на стр. 111.

Функции, значения которых можно вычислить с помощью арифметических операций и элементарных функций, часто называются *кодируемыми функциями* или *разложимыми функциями*. В результате анализа формулы для такой функции, рассматриваемой в виде последовательности арифметических операций и обращений к библиотечным функциям, можно получить эквивалентное представление функции в виде кодового списка, или представить в виде *дерева Канторовича*.

Например, функция (1.15) может быть представлена в виде следующего кодового списка:

$$\begin{aligned} T1 &= X^4; \\ T2 &= X^3; \\ T3 &= 4 \cdot T2; \\ T4 &= X^2; \\ T5 &= 4 \cdot T4; \\ T6 &= Y^2; \\ T7 &= T1 + T3; \\ T8 &= T7 + T5; \\ F &= T8 + T6. \end{aligned} \tag{1.16}$$

Кодовые списки успешно применяются для вычисления значения функции и её дальнейшего анализа в самых различных ситуациях, к примеру, в компиляторах с языков программирования высокого уровня, математических библиотеках и т. п. В то же время, каждая строка кодового списка (1.16) представляет собой элементарную задачу дифференцирования. Поскольку в общем случае для функции f можно построить несколько кодовых списков, полезно иметь метод для визуализации процесса вычисления функции, из которого можно получить один или несколько кодовых списков. Этот формализм известен как дерево Канторовича [26, 4]. Пример такого дерева для функции (1.15) показан на рис. 1.1. Как мы видим, этот ориентированный граф подобен блок-схеме программы.

Хотя узлы дерева помечены в соответствии с обозначениями (1.16), из него можно получить несколько разных кодовых списков. Например, такую отличную от (1.16) последовательность команд:

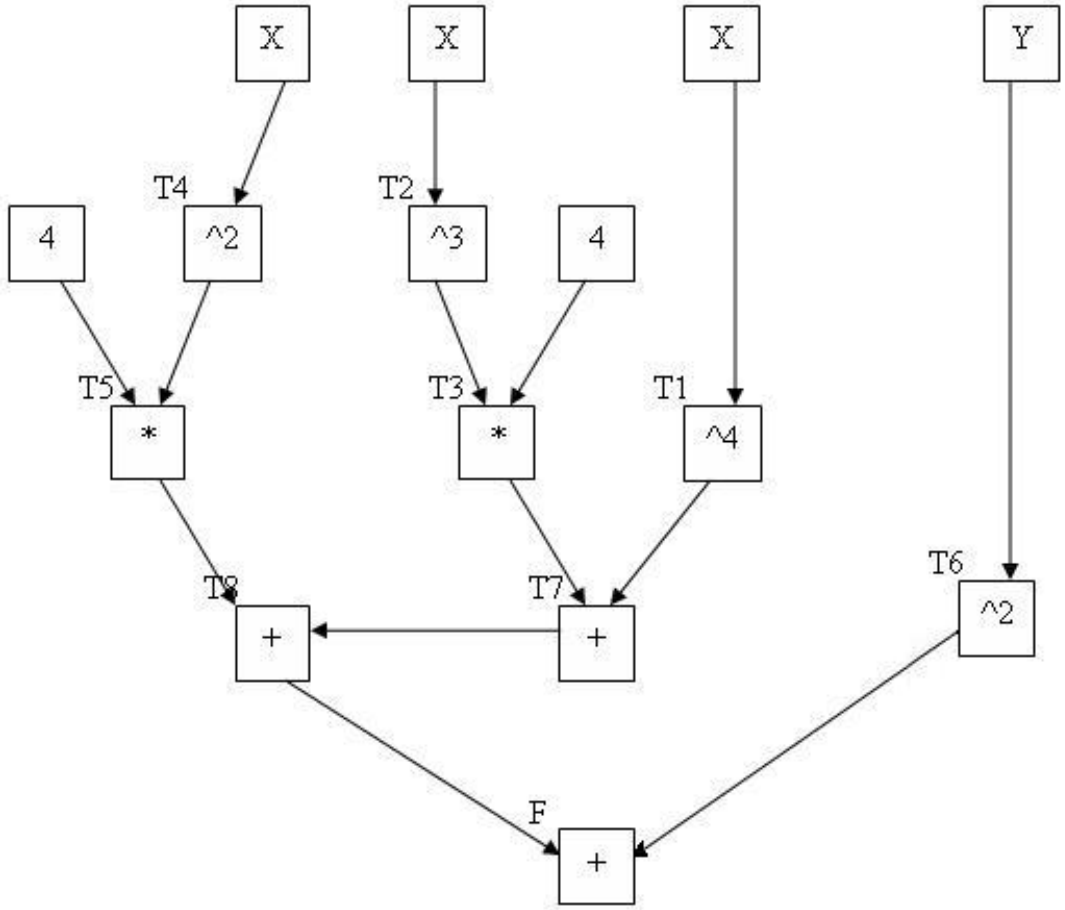


Рис. 1.1: Представление функции (1.15) в виде дерева Канторовича

$$
\begin{aligned}
T4 &= X^2;\\
T2 &= X^3;\\
T1 &= X^4;\\
T6 &= Y^2;\\
T5 &= 4 \cdot T4;\\
T3 &= 4 \cdot T2;\\
T7 &= T1 + T3;\\
T8 &= T7 + T5;\\
F &= T8 + T6.
\end{aligned}
$$

Основная идея автоматического дифференцирования проста — к кодовому списку функции строка за строкой применяются элементарные правила дифференцирования до тех пор, пока не получится список команд, то есть алгоритм для вычисления производной [2]. Для сложных функций применяется известное из математического анализа *цепное правило*. Это правило дифференцирования сложных функций позволяет вычислить производную композиции двух и более функций.

Рассмотрим для начала одномерный случай. Будем обозначать символом $\text{Ш}_a(x)$ шар радиуса a с центром x. Тогда, пусть даны функции, определённые на числовой прямой в окрестностях точки x_0, такие что $f : \text{Ш}_a(x_0) \to \text{Ш}_b(y_0)$, где $y_0 = f(x_0)$ и $g : \text{Ш}_b(y_0) \to \mathbb{R}$. Пусть также функция f дифференцируема в $\text{Ш}_a(x_0)$, а функция g дифференцируема в $\text{Ш}_b(y_0)$. Тогда их композиция также дифференцируема, и её производная имеет вид: $h'(x_0) = g'\big(f(x_0)\big) \cdot f'(x_0)$. Соответственно, цепное правило для вычисления производной функции $f = f(x)$, где $x = x(t)$, принимает следующий вид:

$$
\frac{df}{dt} = \frac{\partial f}{\partial x} \cdot \frac{\partial x}{\partial t} \tag{1.17}
$$

Если в процессе обхода дерева Канторовича зависимость x от t будет потеряна, результат будет неверен. Такая ситуация может встретиться даже при обработке сравнительно простых выражений. Например, выражение

$$
f(x, y, z) = x^2 + \exp\Big(x \cdot (y + x \cdot z)/y\Big) + y + x \cdot z
$$

может быть записано в виде

$$\begin{aligned} U &= Y + X \cdot Z \\ V &= (X \cdot U)/Y \\ F &= X^2 + \exp(V) + U \end{aligned} \tag{1.18}$$

Формальная частная производная от f по x определяется последней строкой (1.18) и дает $2 \cdot x$. Чтобы получить корректную частную производную по x от функции f, заданной последовательностью формул (1.18), необходимо продифференцировать U и V и применить формулу (1.17).

Форма первого дифференциала остаётся одной и той же вне зависимости от того, является ли независимая переменная функцией или нет. Так, пусть дифференциал функции $z = g(y)$ в точке y_0 имеет вид

$$dz = g'(y_0)\, dy,$$

где dy — дифференциал тождественного отображения $y \to y$:

$$dy(e) = e, \quad h \in \mathbb{R}.$$

Теперь, если $y = f(x)$, то $dy = f'(x_0)\, dx$, и согласно цепному правилу:

$$dz = g'\big(f(x_0)\big) \cdot f'(x_0)\, dx = g'(y_0)\, dy.$$

В многомерном случае, когда нам даны две дифференцируемые функции f и g, такие что, $f : \text{Ш}_a(x_0) \subset \mathbb{R}^m \to \text{Ш}_b(y_0) \subset \mathbb{R}^n$, где $y_0 = f(x_0)$, и $g : \text{Ш}_b(y_0) \subset \mathbb{R}^n \to \mathbb{R}^p$, их композиция тоже дифференцируема, и её дифференциал имеет вид

$$dh(x_0) = dg(y_0) \circ df(x_0).$$

При этом матрица Якоби функции h является произведением матриц Якоби функций g и f:

$$\frac{\partial(h_1, \ldots, h_p)}{\partial(x_1, \ldots, x_m)} = \frac{\partial(h_1, \ldots, h_p)}{\partial(y_1, \ldots, y_n)} \cdot \frac{\partial(y_1, \ldots, y_n)}{\partial(x_1, \ldots, x_m)}.$$

Для частных производных сложной функции справедливо

$$\frac{\partial h(x_0)}{\partial x_j} = \sum_{i=1}^{n} \frac{\partial g(y_0)}{\partial y_i} \frac{\partial y_i}{\partial x_j}, \quad j = 1, \ldots, m.$$

Таким образом, для автоматического вычисления производных дифференцируемых функций требуется рекуррентное применение цепного правила. Применительно к нашей задаче для сложной функции от одной переменной

$$f(x) = g(h(x)) \tag{1.19}$$

это правило даёт

$$f'(x) = g'(h(x)) \cdot h'(x), \tag{1.20}$$

или в терминах композиции функций

$$f(x) = (g \circ h)(x),$$
$$f'(x) = (g' \circ h)(x) \cdot h'(x).$$

Очевидно, что для функции, являющейся композицией n функций

$$f = g_1 \circ g_2 \circ \cdots \circ g_{n-1} \circ g_n \tag{1.21}$$

производная $f'(x)$ в точке x имеет вид

$$f'(x) = (g_1' \circ g_2 \circ \cdots \circ g_{n-1} \circ g_n)(x) \cdot (g_2' \circ g_3 \circ \cdots \circ g_{n-1} \circ g_n)(x) \cdots (g_{n-1}' \circ g_n)(x) \cdot g_n'(x). \tag{1.22}$$

Таким образом, для нахождения производной функции суперпозиции f, заданной формулой (1.21), потребуется n производных $\frac{\partial f}{\partial x}$ и значения n множителей в правой части (1.21).

Если положить

$$\begin{aligned} f_0 &= x, \\ f_1 &= g_n(f_0), \\ f_2 &= g_{n-1}(f_1), \\ &\cdots \\ f_{n-1} &= g_2(f_{n-2}), \\ f_n &= g_1(f_{n-1}), \end{aligned}$$

то видно, что вычисление последовательности значений $f_1, f_2, \ldots, f_n$, даёт значение $f(x) = f_n$ функции f в точке x. Более того, имея эту последовательность и производные g_i', где $i = 1, 2, \ldots, n$ можно вычислить величины

$$\begin{aligned} f_0' &= 1, \\ f_1' &= g_n'(f_0) \cdot f_0', \\ f_2' &= g_{n-1}'(f_1) \cdot f_1', \\ &\cdots \\ f_{n-1}' &= g_2'(f_{n-2}) \cdot f_{n-2}', \\ f_n' &= g_1'(f_{n-1}) \cdot f_{n-1}'. \end{aligned}$$

По цепному правилу значение производной f' функции f в точке x есть $f'(x) = f_n'$, таким образом

$$f'(x) = \prod_{i=1}^{n} g_i'(f_{n-1}) \circ f_0' \;=\; g_1'(f_{n-1}) \circ g_2'(f_{n-2}) \circ \ldots \circ g_{n-1}'(f_1) \circ g_n'(f_0').$$

1.3 Интервальные расширения функций

Договоримся обозначать множество всех интервальных векторов, содержащихся в произвольном множестве $D \in \mathbb{R}^n$ как $\mathbb{I}D$, то есть

$$\mathbb{I}D = \{\, \boldsymbol{x} \in \mathbb{IR}^n \mid \boldsymbol{x} \subseteq D \,\},$$

или же, более подробно,

$$\mathbb{I}D = \{\, [x, y] \mid x, y \in \mathbb{R}^n,\, x \in D, y \in D, x \leqslant y \,\}.$$

Задача определения области значений функции на том или ином подмножестве области её определения, эквивалентная интересующей нас *задаче оптимизации*, в интервальном анализе принимает специфическую форму задачи о вычислении так называемого *интервального расширения функции*.

Определение 1.5 *Пусть D — непустое подмножество пространства $\mathbb{R}^n$. Интервальная функция $\boldsymbol{f} : \mathbb{I}D \to \mathbb{IR}^m$ называется интервальным продолжением вещественной функции $f : D \to \mathbb{R}^m$, если $\boldsymbol{f}(x) = f(x)$ для всех $x \in D$.*

Определение 1.6 *Пусть D — непустое подмножество пространства $\mathbb{R}^n$. Интервальная функция $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{IR}^m$ называется интервальным расширением вещественной функции $f : \mathbb{R}^n \to \mathbb{R}^m$, если*

1) $\boldsymbol{f}(x)$ является интервальным продолжением $f(x)$ на D,

2) $\boldsymbol{f}(\boldsymbol{x})$ монотонна по включению, т. е.
$\boldsymbol{x} \subseteq \boldsymbol{y} \;\Rightarrow\; \boldsymbol{f}(\boldsymbol{x}) \subseteq \boldsymbol{f}(\boldsymbol{y})$ *на* $\mathbb{I}D$.

Таким образом, интервальное расширение $\boldsymbol{f}(\boldsymbol{x})$ функции $f(x)$ на брусе $\boldsymbol{x} \subset \mathbb{R}^n$ является для неё внешней оценкой (оценкой объемлющим множеством):

$$\{\, f(x) \mid x \in \boldsymbol{x} \,\} \subseteq \boldsymbol{f}(\boldsymbol{x}).$$

Это свойство напрямую следует из первого пункта определения 1.6 и чрезвычайно удобно для задач оптимизации. Действительно, для непрерывной функции

$$\operatorname{range}_{\boldsymbol{x}} f := \big[\, \min_{x \in \boldsymbol{x}} f(x),\; \max_{x \in \boldsymbol{x}} f(x) \big],$$

где через $\operatorname{range}_{\boldsymbol{x}} f := \{f(x) \mid x \in \boldsymbol{x}\}$ обозначена область значений функции f на брусе $\boldsymbol{x}$. Таким образом, если $\boldsymbol{f}(\boldsymbol{x})$ — интервальное расширение функции $f(x)$ на области $\boldsymbol{x}$, то $\underline{\boldsymbol{f}(\boldsymbol{x})}$ — оценка искомого минимума этой функции снизу:

$$\underline{\boldsymbol{f}(\boldsymbol{x})} \leqslant \min_{x \in \boldsymbol{x}} f(x).$$

Сделаем замечание по поводу терминологии. Интервальным расширением обычно называют как интервальную функцию, так и её значения на конкретных брусах. Как правило, это не приводит к путанице, хотя некоторую осмотрительность нужно проявлять.

Неприятным для практики свойством интервальной арифметики является сравнительно быстрый рост ширины результирующего интервала с увеличением количества арифметических операций, производимых с интервалами для получения результата — так называемый «эффект зависимости». Это обстоятельство послужило причиной появления различных модификаций способов вычисления интервальной оценки, а также создания других интервальных арифметик, целью которых являлась, в основном, попытка препятствовать этому явлению. Наиболее известными из интервальных арифметик являются интервальная арифметика с нестандартным вычитанием и делением [75], обобщённая интервальная арифметика [93], сегментная арифметика [121], аффинная интервальная арифметика [77, 126]. Из различных методик вычисления интервальной оценки области значений функции большое значение имеют центрированная форма интервального расширения [117, **?**] и среднезначная форма интервального расширения [81, 106, 108] (которую в последнее время всё чаще называют «дифференциальной центрированной формой»).

Несмотря на то, что часто эти подходы оказываются эффективными применительно к определенным классам задач, проблема построения интервальных расширений функций остаётся одной из важнейших задач интервального анализа, поиски различных решений которой продолжаются и в настоящее время. Мы рассматриваем вопросы точности интервальных оценок в §1.4. Но вообще говоря, рост результирующих интервалов оценок в полной мере отражает реальность и, в сущности, соответствует принципу возрастания неопределённости (энтропии).

Далее приводится описание некоторых типов интервальных расширений.

1.3.1 Естественное интервальное расширение

Если для рациональной функции $f(x)$ на интервале $\boldsymbol{x}$ определён результат $\boldsymbol{f}_\natural(\boldsymbol{x})$ подстановки вместо аргументов функции *интервалов их изменения* и выполнения всех действий над ними по правилам интервальной арифметики, то полученная таким образом интервальная функция $\boldsymbol{f}_\natural(\boldsymbol{x})$ будет интервальным расширением функции f на брусе $\boldsymbol{x}$. Это интервальное расширение называется *естественным интервальным расширением.*

Вычислять такое интервальное расширение просто, но, по сложившемуся мнению, использование его приводит к весьма грубым результатам при оценивании области значений функции [97, 117]. Действительно, весьма часто вычисляемая с помощью естественного интервального расширения оценка значительно шире реальной области значений функции. Например, вычислив естественное интервальное расширение функции $f(x) = x^2 - x$ на интервале $[-1, 0]$, получим, что $\boldsymbol{f}(\boldsymbol{x}) = [-1, 2]$. Точная же оценка области значений функции на выбранной области определения равна $[0, 2]$.

К вопросу точности интервальных расширений мы вернёмся позже, здесь же подчеркнем, что *точность естественного интервального расширения существенно зависит от способа записи выражения для функции.* В частности, если в рациональное выражение $f(x_1, \ldots, x_n)$ каждая переменная входит не более одного раза и только в первой степени, то его интервальная оценка полученная способом естественного интервального расширения $\boldsymbol{f}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ будет точной для любого такого набора $(\boldsymbol{x}_1 \ldots \boldsymbol{x}_n)$, что $\boldsymbol{f}(\boldsymbol{x}_1 \ldots \boldsymbol{x}_n)$ имеет смысл.

Доказательство этого утверждения в книге [106] проведено с помощью простых теоретико-множественных рассуждений, но оно не является вполне строгим. Полное и математически строгое обоснование этого факта приводится в электронной книге [72] на основе понятия «дерева Канторовича» и понятия зависимости (связанности) интервальных величин.

В случае, когда не удаётся добиться единственности вхождения в выражение каждой переменной, многие исследователи [25, 106] рекомендуют стараться хотя бы уменьшить число этих вхождений. В частности, для предварительного упрощения формулы целевой функции рекомендуется использовать в качестве алгоритма предобработки так называемую схему Горнера [33], достаточно простой и универсальный способ вычисления значений многочленов, который известен ещё со времён Ньютона. Он основан на выделении из многочлена

$$a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n \tag{1.23}$$

множителей вида $x - c$. В результате получается многочлен

$$b_0 x^{n-1} + b_1 x^{n-2} + \cdots + b_{n-2} x + b_{n-1}, \tag{1.24}$$

где $b_0 = a_0, \quad b_k = a_k + c b_{k-1}$. Кроме этого, результат такого деления имеет остаток b_n.

Схема проста и изящна (слово «схема» в названии этого алгоритма связано с тем, что исторически его реализуют в виде таблицы [12]). Вынося переменную x за скобки везде, где это возможно мы переписываем выражение (1.23)

в виде

$$f(x) = (\dots(((x + a_1) \cdot x + a_2) \cdot x + a_3) \dots) \cdot x + a_n. \tag{1.25}$$

Вычисление $f(x)$ происходит в соответствии с порядком, задаваемым скобками в (1.25):

$$\begin{aligned} p_1 &= a_0 x + a_1; \\ p_2 &= p_1 x + a_2; \\ p_3 &= p_2 x + a_3; \\ &\dots \\ p_n &= p_{n-1} x + a_n; \\ f(x) &= p_n. \end{aligned} \tag{1.26}$$

Количество умножений в этом алгоритме на единицу меньше, чем количество ненулевых коэффициентов у многочлена $p(x)$. Кроме того, алгоритм выполняет n сложений.

Случай двух переменных можно свести к рассмотренному ранее, если допустить, что $a_0, a_1, \dots, a_{n-1}, a_n$ — не константы, а полиномы от одной переменной $a_0(y), a_1(y), \dots a_{n-1}(y), a_n(y)$. Таким образом, проведя схему Горнера по одной переменной и придя к записи вида

$$a_n + x(a_{n-1} + x(\cdots))),$$

можем последовательно применить схему Горнера к коэффициентам $a_i(y)$. Стройную теорию здесь портит тот факт, что простого алгоритма для автоматического символьного преобразования формулы внутри программы мы пока предложить не можем.

Далее мы рассматриваем другие формы интервальных расширений, а к детальному обсуждению их точности обращаемся в §1.4.

1.3.2 Среднезначная форма интервального расширения

Естественное интервальное расширение, описанное в предыдущей главе, использовало лишь информацию о верхней и нижней оценках значений каждой из элементарных функций, а затем преобразовывало полученные интервалы в соответствии с правилами интервальной алгебры.

Для повышения точности интервального расширения мы можем привлекать информацию о производных интересующей нас функции. Так, если $\boldsymbol{f}'(\boldsymbol{x})$ — интервальное расширение производной $f'(x)$ функции $f(x)$ на интервале

$\boldsymbol{x} \in \mathbb{IR}^n$, то $\overline{\boldsymbol{f}'(\boldsymbol{x})}$ — оценка максимального значения производной на этом интервале сверху, а $\underline{\boldsymbol{f}'(\boldsymbol{x})}$ — оценка минимального значения производной снизу. Таким образом, значение искомой функции на брусе $\boldsymbol{x}$ может быть оценено следующим образом:

$$\min\left(f(c) + \sum_{i=1}^{n} \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial x_i} \cdot \frac{\mathrm{wid}(\boldsymbol{x}_i)}{2}\right) \leqslant f(\boldsymbol{x}) \leqslant \max\left(f(c) + \sum_{i=1}^{n} \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial x_i} \cdot \frac{\mathrm{wid}(\boldsymbol{x}_i)}{2}\right),$$

где $c \in \boldsymbol{x}$ — некоторая фиксированная точка. Идея подхода проиллюстрирована на рис. 1.2.

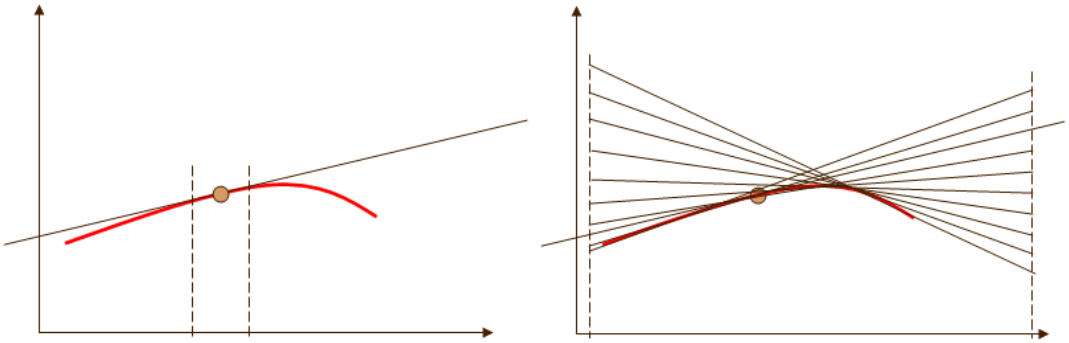


Рис. 1.2: Иллюстрация идеи среднезначного интервального расширения

Выражение

$$\boldsymbol{f}_{mv}(\boldsymbol{x}) = f(c) + \boldsymbol{f}'(\boldsymbol{x}) \cdot (\boldsymbol{x} - c) \tag{1.27}$$

по определению является интервальным расширением функции f на интервале $\boldsymbol{x}$. В случае, если c — просто какая-то точка, это утверждение весьма очевидно. Если же мы выбираем в качестве c среднюю точку интервала, понадобится теорема Капрани-Мадсена [81], чтобы показать это. При этом именно середина интервала традиционно используется в качестве c. Отсюда и название этого типа интервального расширения — среднезначное. В англоязычной литературе оно называется mean-value, сокращение mv от него уже было использовано в выражении (1.27).

Отметим, что в последнее время рассмотренную форму интервального расширения всё чаще называют «дифференциальной центрированной формой», так как она является представителем общих центрированных форм, у которого в качестве коэффициентов разложения взяты интервальные расширения производных.

1.3.3 Наклонная форма интервального расширения

Наклонная форма интервального расширения функции — ещё одна разновидность центрированной формы. Она представляет собой выражение вида

$$\boldsymbol{f}_s(\boldsymbol{x}) = f(c) + f^{\angle}(c, \boldsymbol{x})(\boldsymbol{x} - c), \tag{1.28}$$

где $f^{\angle}(c, \boldsymbol{x})$ — *наклон* функции f в точке c на множестве $\boldsymbol{x}$ (наклонам посвящён §1.2.2). Точка c, которая, вообще говоря, может быть произвольной точкой, даже не принадлежащей интервалу оценивания $\boldsymbol{x}$ — называется *центром формы*.

Если центр формы c находится на интервале $\boldsymbol{x}$, на котором мы вычисляем интервальное расширение, то рассматриваемому типу интервального расширения также присущ второй порядок точности, характерный для центрированных форм.

1.4 Точность интервального расширения функций

Оценим область значений следующей простейшей функции

$$f(x) = x - x, \tag{1.29}$$

скажем, на интервале $[-1, 1]$. Если мы построим естественное интервальное расширение, то по правилам, описанным выше, область значений функции будет оценена как $[-1, 1] - [-1, 1] = [-2, 2]$. На брусе $[-100, 100]$ естественное интервальное расширение даст уже оценку $[-200, 200]$.

Среднезначная форма, оценивающая функцию по формуле (1.27), выдаст границы $[0, 0]$, независимо от размеров входного интервала. В то же время, если мы, для примера рассмотрим следующую функцию,

$$f(x) = 4x^4 - 3x^3 + 2x^2 - x, \tag{1.30}$$

чья производная равна $f'(x) = 16x^3 - 9x^2 + 4x - 1$, а, следовательно, интервальное расширение, вычисленное по формуле (1.27) на интервале $[-1, 1]$ даст следующую оценку:

$$\begin{gathered} c = \operatorname{mid}(\boldsymbol{x}) \\ f\Big(\operatorname{mid}(\boldsymbol{x})\Big) = 0 \\ \boldsymbol{f}'(\boldsymbol{x}) \;=\; [-16, 16] - [0, 9] + [-4, 4] - 1 \;=\; [-30, 19] \end{gathered} \tag{1.31}$$

Соответственно,

$$\begin{aligned}\boldsymbol{f}_{mv}(\boldsymbol{x}) &= 0 + ([-16,16] - [0,9] + [-4,4] - 1) \cdot [-1,1] \\ &= [-25,16] + [-4,4] - 1 = [-30,19].\end{aligned} \tag{1.32}$$

При этом, естественное интервальное расширение, вычисленное по формуле (1.27) на том же интервале $[-1,1]$ даёт оценку

$$\begin{aligned}\boldsymbol{f}_{\natural}(\boldsymbol{x}) &= 4 \cdot [-1,1]^4 - 3 \cdot [-1,1]^4 + 2 \cdot [-1,1]^2 - [-1,1] \\ &= 4 \cdot [0,1] - 3 \cdot [-1,1] + 2 \cdot [0,1] - [-1,1] \\ &= [0,4] - [-3,3] + [0,2] - [-1,1] \\ &= [-4,10].\end{aligned} \tag{1.33}$$

Налицо так называемый «эффект зависимости», уже упоминавшийся в §1.3 — втягивание «лишних» точек при математических преобразованиях. Корни проблемы в том, что в простейших интервальных вычислениях неявно подразумевается, что любая переменная в каждый момент времени может принять любое значение из своего интервала. То есть все входящие интервалы рассматриваются как независимые. Это особенно очевидно на примере (1.29). Действительно, если бы вместо $f = x - x$ было $f = x - y$, то оценка была бы точной.

Возможно, нам удастся улучшить точность среднезначной оценки, получив более точную оценку на значения производной. Посчитаем естественное интервальное расширение производной по схеме Горнера. Как отмечалось в §1.3.1, посвященной естественному интервальному расширению, некоторые классики рекомендуют именно этот способ вычисления для повышения точности:

$$\begin{aligned}\boldsymbol{f}'_H(\boldsymbol{x}) &= \big((16\boldsymbol{x} - 9)\boldsymbol{x} + 4\big)\boldsymbol{x} - 1 \\ &= \big(([-16,16] - 9) \cdot [-1,1] + 4\big) \cdot [-1,1] - 1 \\ &= ([-25,7] \cdot [-1,1] + 4) \cdot [-1,1] - 1 \\ &= [-21,29] \cdot [-1,1] - 1 \\ &= [-30,28].\end{aligned} \tag{1.34}$$

Соответственно,

$$\boldsymbol{f}^H_{mv}(\boldsymbol{x}) \;=\; 0 + [-30,28] \cdot [-1,1] \;=\; [-30,28]. \tag{1.35}$$

Вопреки ожиданиям, использование этого метода увеличило ширину интервальной оценки. (Сравните с (1.31)). Что не так уж и удивительно, если

вспомнить, что в интервальных алгебрах форма записи функции, её вид существенно влияют на точность интервального расширения и в общем случае $\boldsymbol{x}^n \neq \underbrace{\boldsymbol{x} \cdot \ \cdots \ \cdot \boldsymbol{x}}_{n \text{ раз}}$.

Действительно, $\boldsymbol{x}^2 = [0, 4]$ на интервале $[-2, 1]$, в то время как $\boldsymbol{x} \cdot \boldsymbol{x} = [-2, 4]$, $\boldsymbol{x}^3 = [-8, 1]$ на том же интервале, когда $\boldsymbol{x} \cdot \boldsymbol{x} \cdot \boldsymbol{x} = [-8, 4]$ и тому подобное. Схема Горнера как раз переводит возведение в степень в каскады умножений. Правда, есть и обратные примеры: на интервале $[-1, -5]$ $\boldsymbol{x}^3 = \boldsymbol{x} \cdot \boldsymbol{x} \cdot \boldsymbol{x} = [-125, -1]$. Что бы не строить свою теорию лишь на одном примере, мы сгенерировали случайным образом ещё несколько полиномов различных степеней и произвольными коэффициентами. Для них посчитали естественное расширение способом «в лоб» и по схеме Горнера на интервалах различной ширины.

Наши эксперименты подтверждают сформулированные выше рассуждения. Это означает что иногда использование схемы Горнера давало более точный результат, а иногда — совсем наоборот. Характерно, что для отрицательных интервалов точнее схема Горнера, для нуль-содержащих — наоборот, её использовать категорически не рекомендуется. Что же касается вычисления интервального расширения полинома над положительными интервалами, то неважно, какую вычислительную схему использовать: обе обеспечивают одинаковую точность.

Обратимся теперь к сравнению точности естественного, среднезначного и наклонных интервальных расширений.

Для построения наклонной формы интервального расширения, описываемой в §1.3.3, необходимо вычисление наклона функции на данном интервале (см. §1.2.2). Если в качестве центра форма будет традиционно выбрана середина интервала, то наклонная форма интервального расширения оценит область определения функции (1.30) на выбранном нами интервале $[-1, 1]$ как $[-\infty, +\infty]$ из-за деления на интервал, содержащий нуль. Выберем другой интервал интереса, например, $[35, 40]$. В этом случае наклонная форма интервального расширения с центром формы c в середине интервала обозначит границы как $[4\,973\,701, 23\,805\,845]$ (результат округлён до целых). В то же время, естественное интервальное расширение позволяет указать более точные границы: $[5\,812\,910, 10\,114\,540]$.

На время оставим модельные примеры и перейдем к жизненному и чуть более сложному примеру. Вычислим естественное интервальное расширение для функции

$$f_{P5}(x) = (2x^3y - y^3)^2 + (6x - y^2 + y)^2 \tag{1.36}$$

на брусе $[-5,5]\times[-2,6]$. Действуя по определению, получаем:

$$\begin{aligned}\boldsymbol{f}_{\natural}^{P5}(\boldsymbol{x}) &= \left(2\cdot[-5,5]^3\cdot[-2,6]-[-2,6]^3\right)^2+\left(6\cdot[-5,5]-[-2,6]^2+[-2,6]\right)^2\\ &= \left(2\cdot[-125,125]\cdot[-2,6]-[-8,216]\right)^2+\left([-30,30]-[0,36]+[-2,6]\right)^2\\ &= \left([-250,250]\cdot[-2,6]-[-8,216]\right)^2+\left([-66,30]+[-2,6]\right)^2\\ &= \left([-1\,500,1\,500]-[-8,216]\right)^2+[-68,36]^2\\ &= [-1716,1508]^2+[-68,36]^2\\ &= [0,2\,944\,656]+[0,4\,624]\\ &= [0,2\,949\,280].\end{aligned} \tag{1.37}$$

Функция (1.36) также известна под именем «Price 5» (P5) [90], и мы ещё не раз в дальнейшем будем использовать её в качестве тестовой. Точные значения минимума и максимума на указанной области определения равны 0 и 2 948 256 соответственно. Получившаяся разница менее одного процента является совсем неплохой для способа, позволившего за одну итерацию найти гарантированные внешние оценки минимума и максимума функции, особенно с учётом характера функции (см. рис. 1.3 и 1.4).

Как уже не раз отмечалось, точность естественного интервального расширения существенно зависит от способа записи выражения для функции. В качестве иллюстрации предлагаем раскрыть скобки в формуле (1.36), тем самым существенно увеличив количество вхождений обеих переменных:

$$\begin{aligned}f &= (2x^3y-y^3)^2+(6x-y^2+y)^2\\ &= 4x^6y^2-4x^3y^4+36x^2+12xy-12xy^2+y^2-2y^3+y^4+y^6.\end{aligned} \tag{1.38}$$

Естественное интервальное расширение такого выражения на том же брусе, что и в (1.37), даст уже оценку $[-698\,524,\ 2\,900\,952]$.

Вообще, для дальнейшего важно, что точность интервального оценивания при использовании любой из форм интервального расширения критическим образом зависит от ширины бруса оценивания. Для естественного интервального расширения

$$\operatorname{dist}\left(\boldsymbol{f}_{\natural}(\boldsymbol{x}),\ \operatorname{range}_{\boldsymbol{x}} f\right)\leqslant C\cdot\|\operatorname{wid}\ \boldsymbol{x}\| \tag{1.39}$$

с некоторой константой C, тогда как среднезначная форма имеет уже **второй порядок точности** [110], при условии, что мы оцениваем производную достаточно аккуратно — с первым порядком точности.

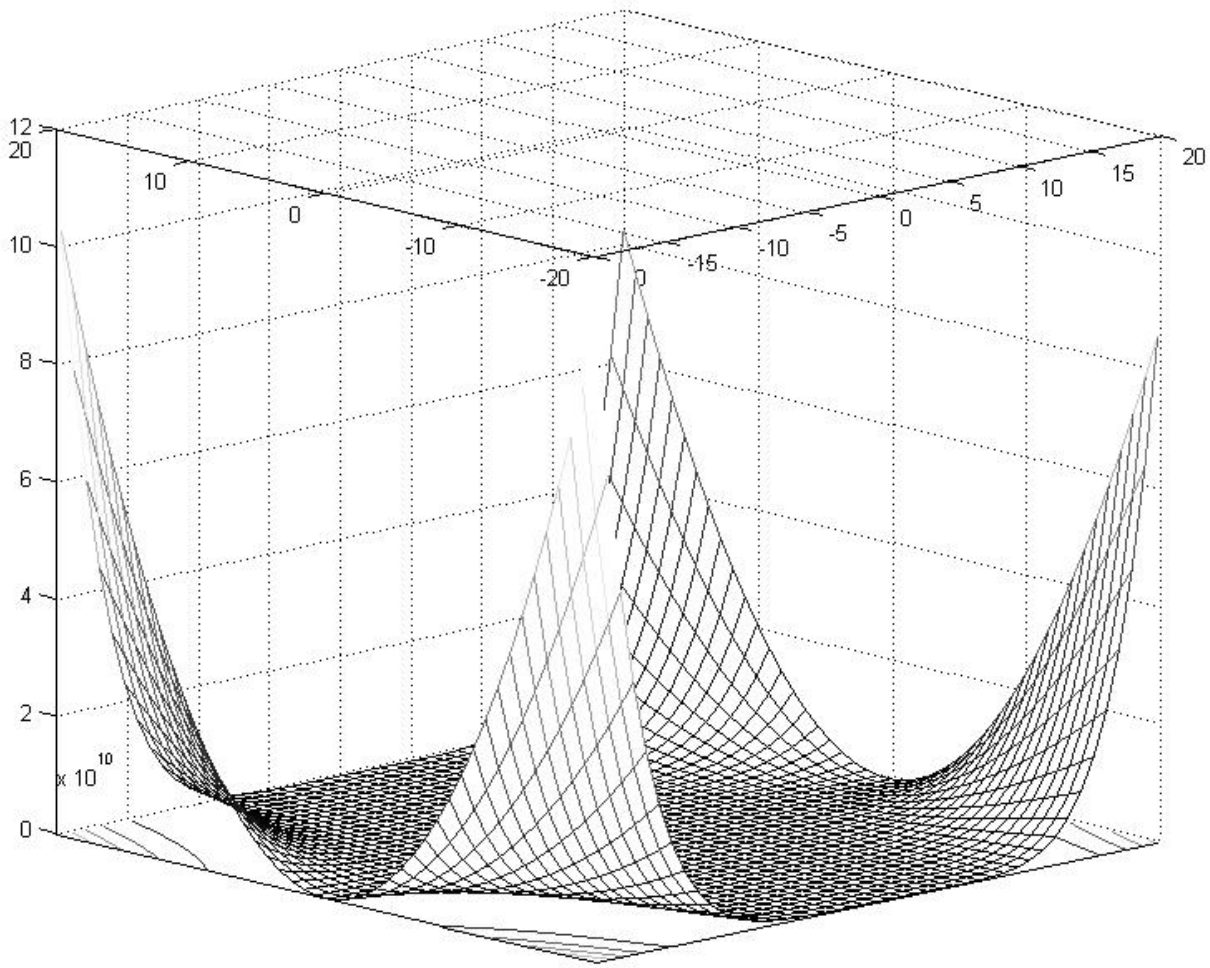


Рис. 1.3: Вид графика функции «Price 5» (P5).

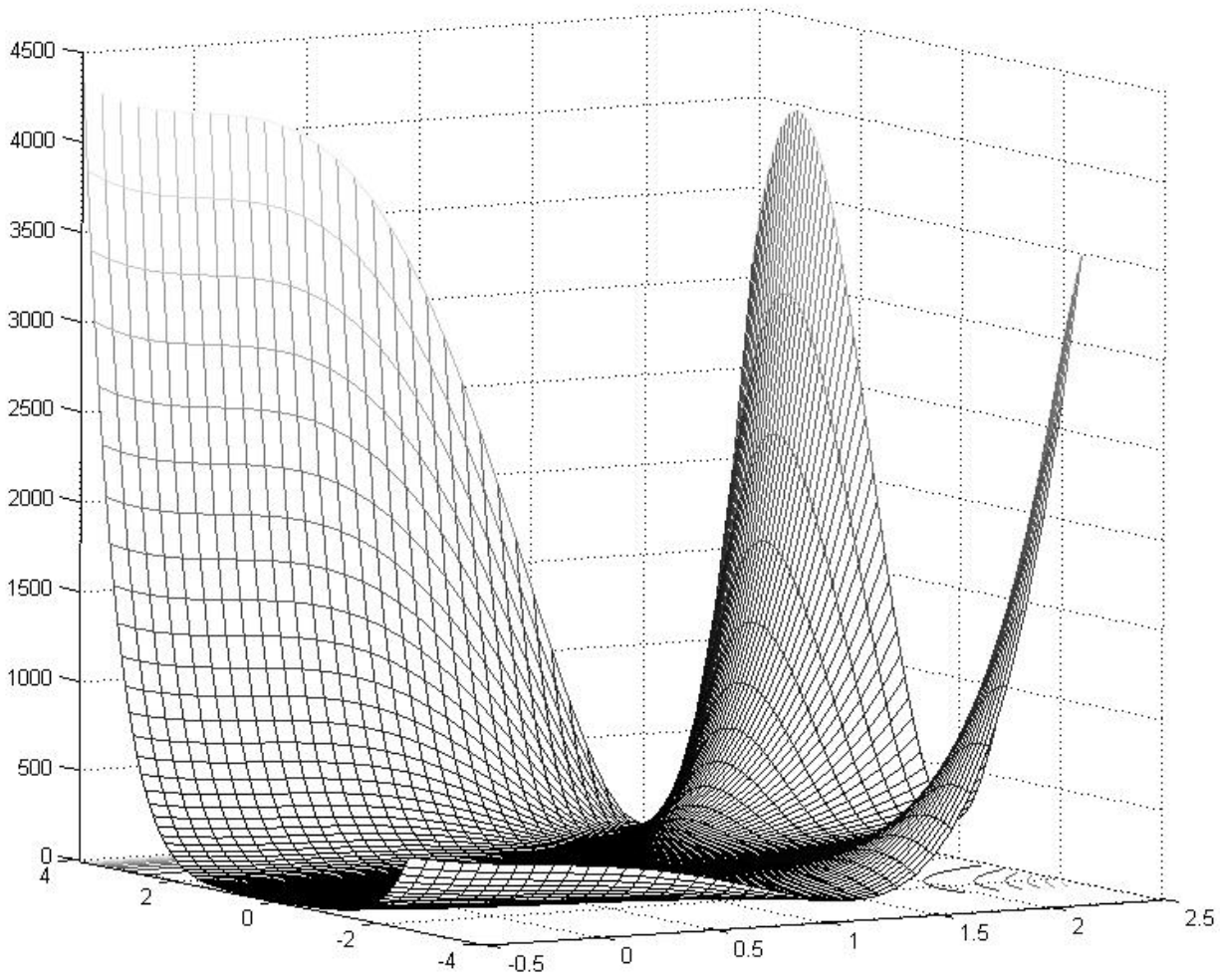


Рис. 1.4: График функции «Price 5» (P5) вблизи начала координат.

Действительно, если в формуле вычисления интервального расширения функции в среднезначной форме (1.27) обозначить интервальное расширение производной по i-той переменной как g_i:

$$\boldsymbol{f}_{mv}(\boldsymbol{x}) = f(c) + \sum_{i=1}^{n} \boldsymbol{g}_i(\boldsymbol{x}) \cdot (\boldsymbol{x}_i - c_i),$$

то можно показать [82, 107] справедливость следующего утверждения.

Если производные g_i для всех $i = 1, \ldots, n$ удовлетворяют условию Липшица (то есть $\operatorname{dist}\big(g_i(x), g_i(y)\big) \leqslant L \cdot \operatorname{dist}(x, y)$) и при стремлении размеров бруса к нулю для интервального расширения производной справедливо (1.39), то

$$\operatorname{dist}\big(\boldsymbol{f}_{mv}(\boldsymbol{x}),\, \operatorname{range}_{\boldsymbol{x}} f\big) \;=\; O(\operatorname{wid}^{\,2}(\boldsymbol{x})) \tag{1.40}$$

Если же для используемой интервальной оценки для производных условие (1.39) не выполняется, а верно лишь

$$\operatorname{wid} \boldsymbol{g}_i(\boldsymbol{x}) \leqslant c \qquad \text{для} \;\; i = \overline{1, n},$$

то вместо (1.40) выполняется соотношение

$$\operatorname{dist}\big(\boldsymbol{f}_{mv}(\boldsymbol{x}),\, \operatorname{range}_{\boldsymbol{x}} f\big) = O(\operatorname{wid}\,(\boldsymbol{x})).$$

Именно благодаря тому, что среднезначная форма интервального расширения имеет второй порядок точности, она значительно преобладает в современных интервальных вычислениях над естественным интервальным расширением. Наши эксперименты действительно подтверждают, что ширина среднезначной формы с уменьшением размера бруса уменьшается значительно быстрее ширины естественного интервального расширения. К сожалению, это ещё не означает, что его точность лучше на брусах конечных размеров, т. е. не в асимптотике.

На графике 1.5 показаны верхняя и нижняя оценки области значений функции

$$f = 4x^2 - 2.1x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4, \tag{1.41}$$

также известной под именем «Шестигорбый верблюд» (Six Hump Camel Back или «C10») [90]. Это весьма круто возрастающая за пределами бруса $[-1, 1] \times [-1, 1]$ функция (см. рис. 1.6) имеет шесть локальных минимумов, которым она и обязана своим названием. Вид функции вблизи начала координат приведен на рис. 1.7.

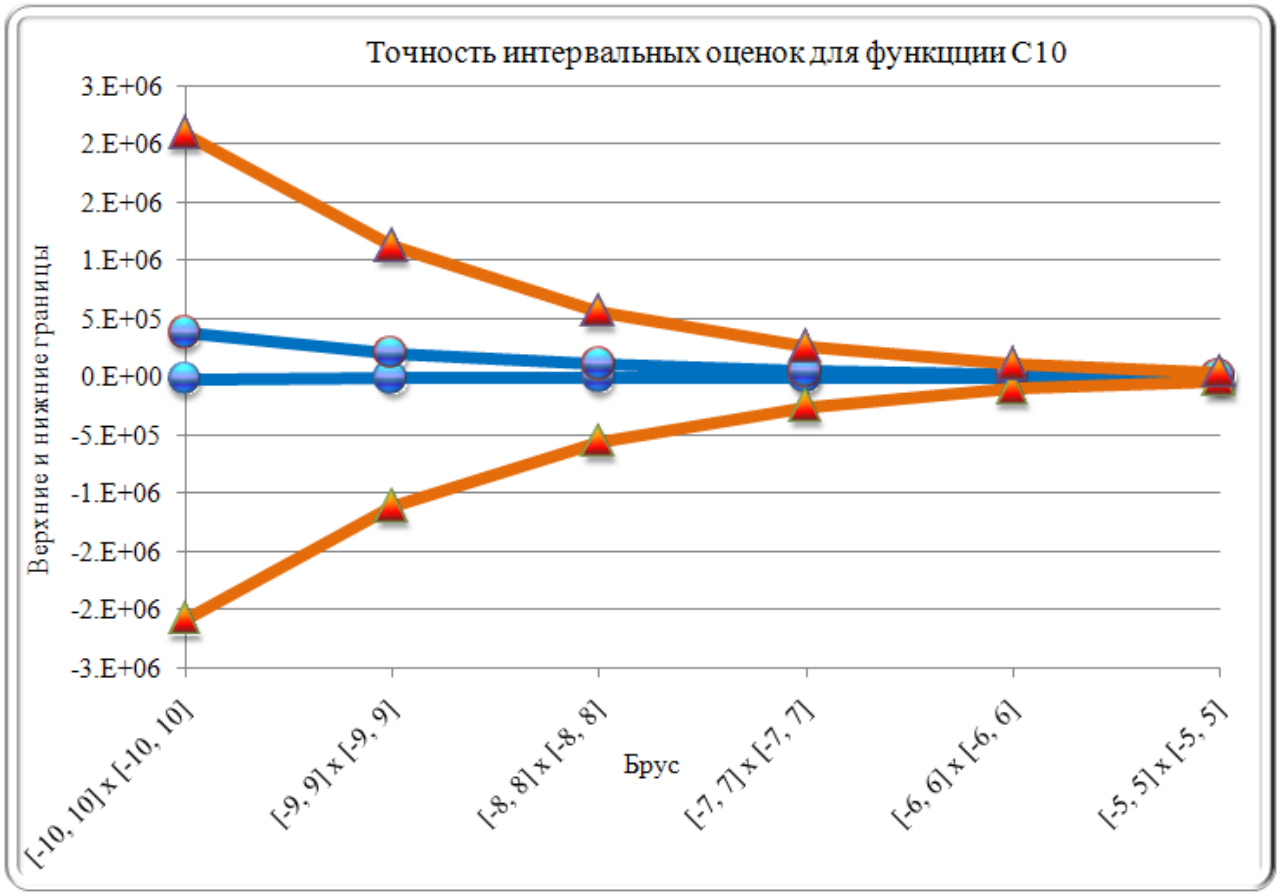


Рис. 1.5: Сравнение интервальных оценок

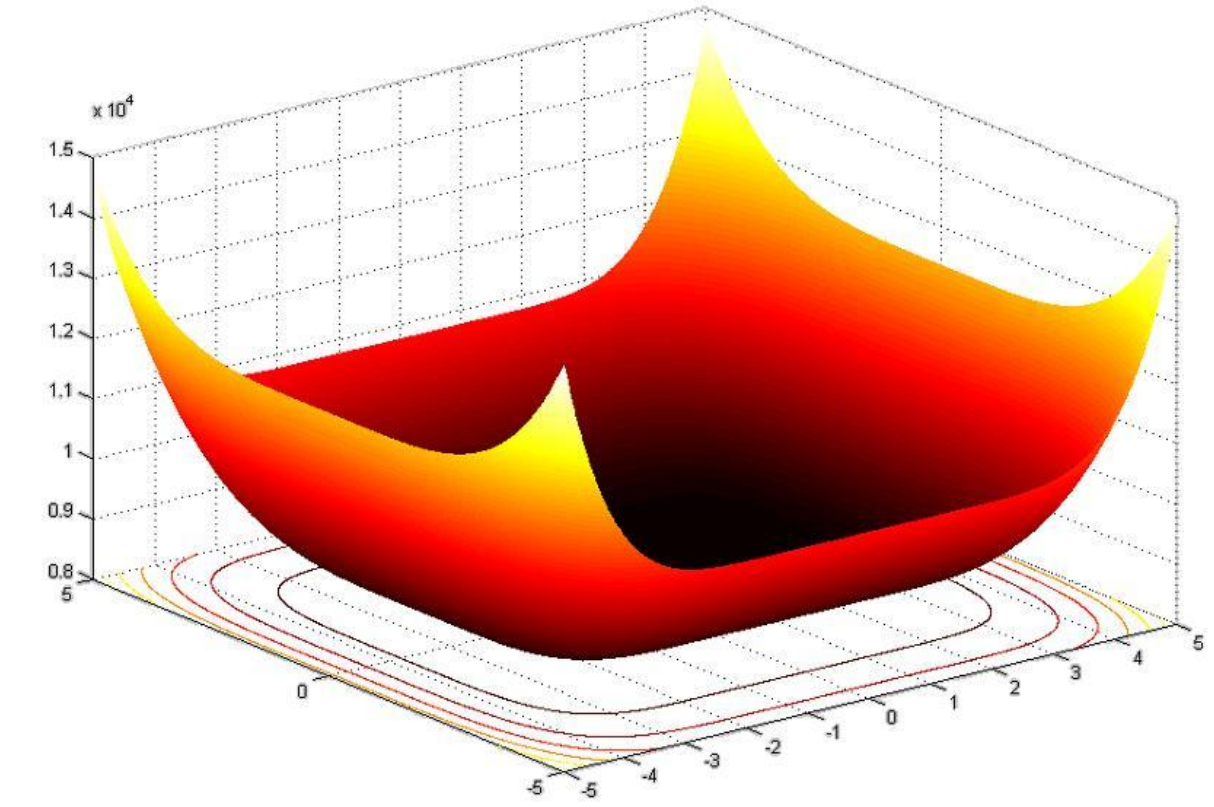


Рис. 1.6: График функции «Шестигорбый верблюд» (C10) — «конформация ванна»

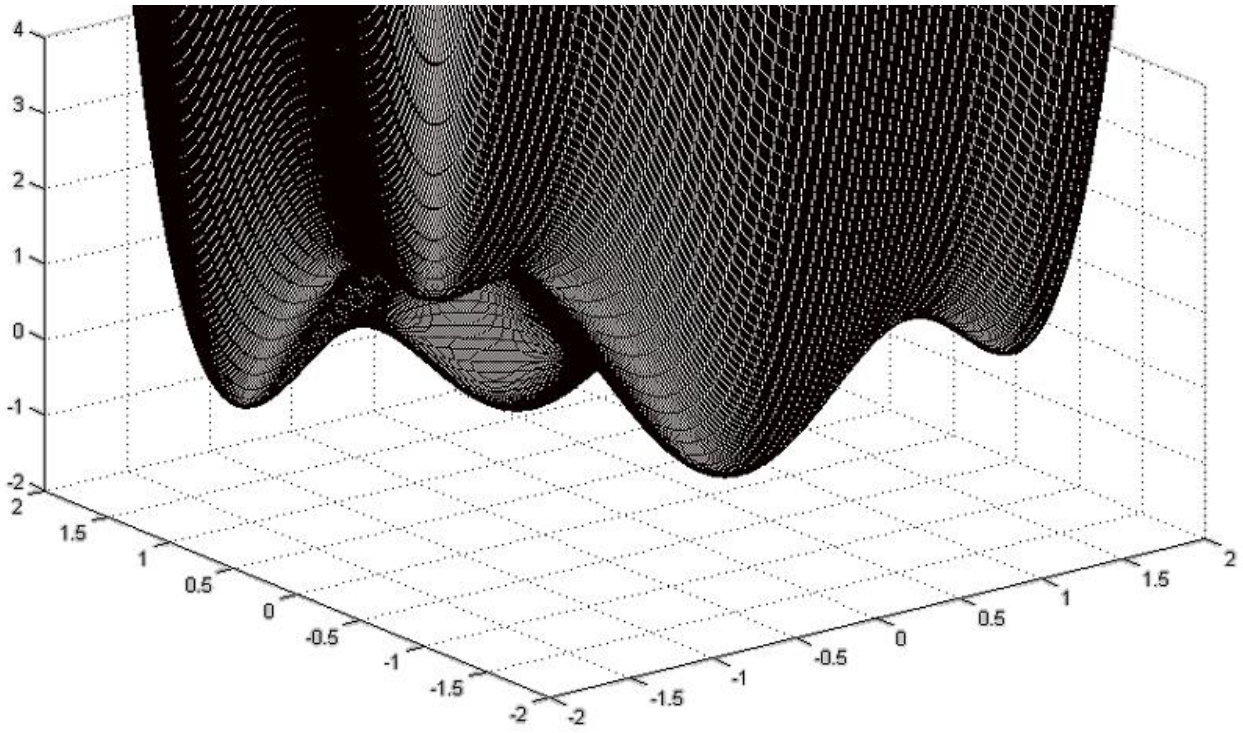


Рис. 1.7: Вид функции «Шестигорбый верблюд» (C10) вблизи начала координат

Результаты проведённых расчётов показались парадоксальными и стимулировали дальнейший поиск в этом направлении. Были проверены ещё несколько функций:

$$f_R = x^2 + y^2 - \cos(18x) - \cos(18y), \tag{1.42}$$

известная под именем десятой функции Растригина (Rastrigin10 или R10, вид её графика вблизи начала координат можно найти на стр. 99), а также

$$f_{G2} = \frac{x^2 + y^2}{200} - \cos(x) \cdot \cos\left(\frac{y}{\sqrt{2}}\right) + 1, \tag{1.43}$$

которую в дальнейшем в соответствии с [90] мы будем называть Griewank 2 или же «G2» (вид её графика показан на рис. 1.8).

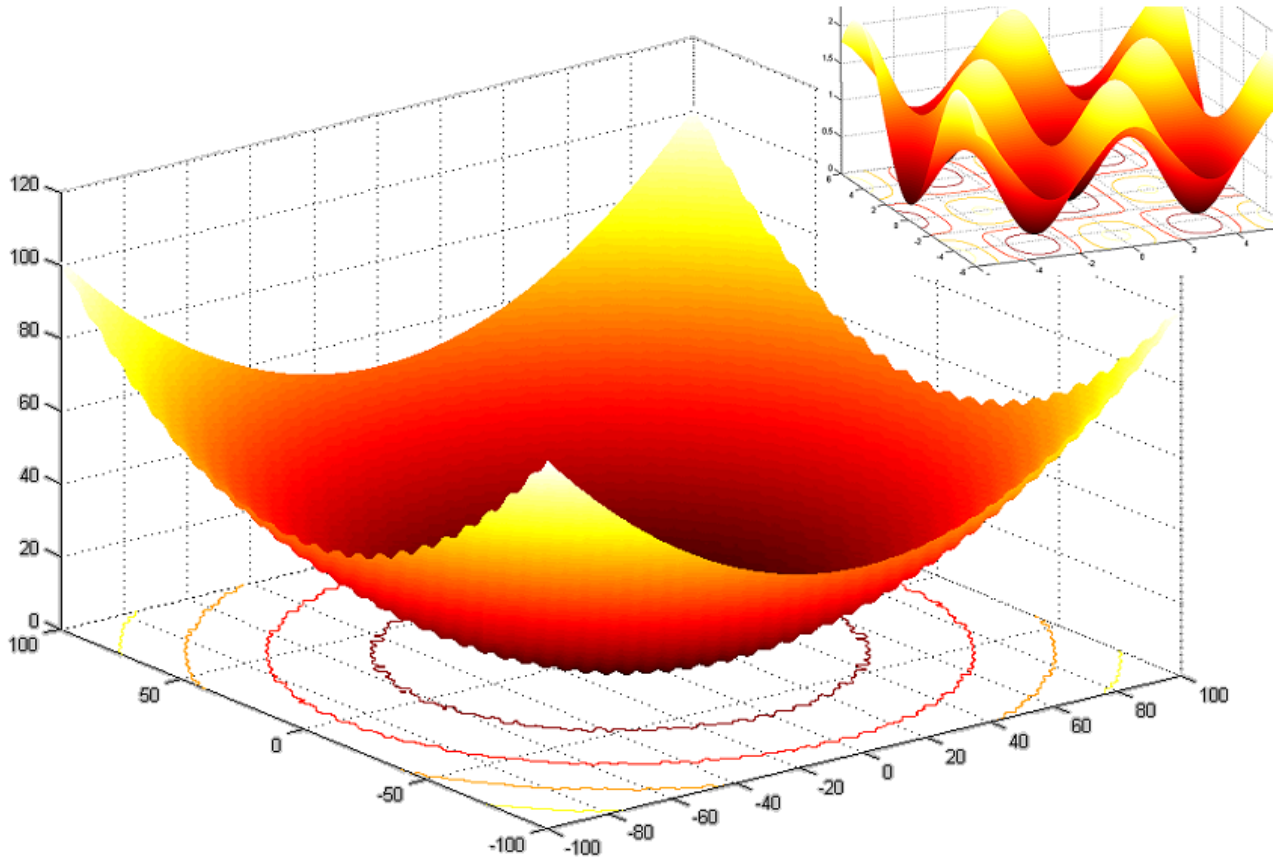


Рис. 1.8: Вид функции «Griewank 2» (G2)

Брус	«C10»		«R10»		«G2»		«RB4»	
	$IE_\natural$	IE_{mv}	$IE_\natural$	IE_{mv}	$IE_\natural$	IE_{mv}	$IE_\natural$	IE_{mv}
$[-10,10]$ × $[-10,10]$	395333	4169800	204	760	3	22	1000120	8000400
$[-9,9] \times [-9,9]$	217979.1	2237440	132	544	2.81	19.62	656200	5249130
$[-8,8] \times [-8,8]$	113006.6	1118540	132	544	2.64	17.28	409681	3277050
$[-7,7] \times [-7,7]$	54352.4	511814	102	448	2.49	14.98	240164	1920996
$[-6,6] \times [-6,6]$	23817.6	209044	76	360	2.36	12.72	129649	1036944
$[-5,5] \times [-5,5]$	9270.83	73450	54	280	2.25	10.5	62536	500100
$[-2,2] \times [-2,2]$	158.933	596.8	12	88	1.46	4.08	1609	12816
$[-1,1] \times [-1,1]$	16.4333	50	6	40	0.6	1.7	104	804
$[-0.5,0.5]$ × $[-0.5,0.5]$	2.8865	6.5	4.5	19	0.1792	0.4844	8.25	51.5
$[-0.1,0.1]$ × $[-0.1,0.1]$	0.1006	0.1832	2.4744	3.64	0.0076	0.0202	0.41	0.3
$[-0.05,0.05]$ × $[-0.05,0.05]$	0.025	0.0452	0.7618	1.42	0.0019	0.005	0.2006	0.11
$[-0.01,0.01]$ x $[-0.01,0.01]$	0.001	0.0018	0.0325	0.0647	7.6E-05	0.0002	0.04	0.0202

Для большей наглядности приводим график для четырех функций (тех, что могут быть построены в одном масштабе) на рис. 1.9.

Вывод из полученных результатов прежний — естественное интервальное расширение, как правило, позволяет получать более качественную оценку. Лишь на интервалах, чья ширина очень мала, среднезначная форма значительно чаще дает более точный результат.

Изучим этот эффект более подробно на основе функции «RB4», также

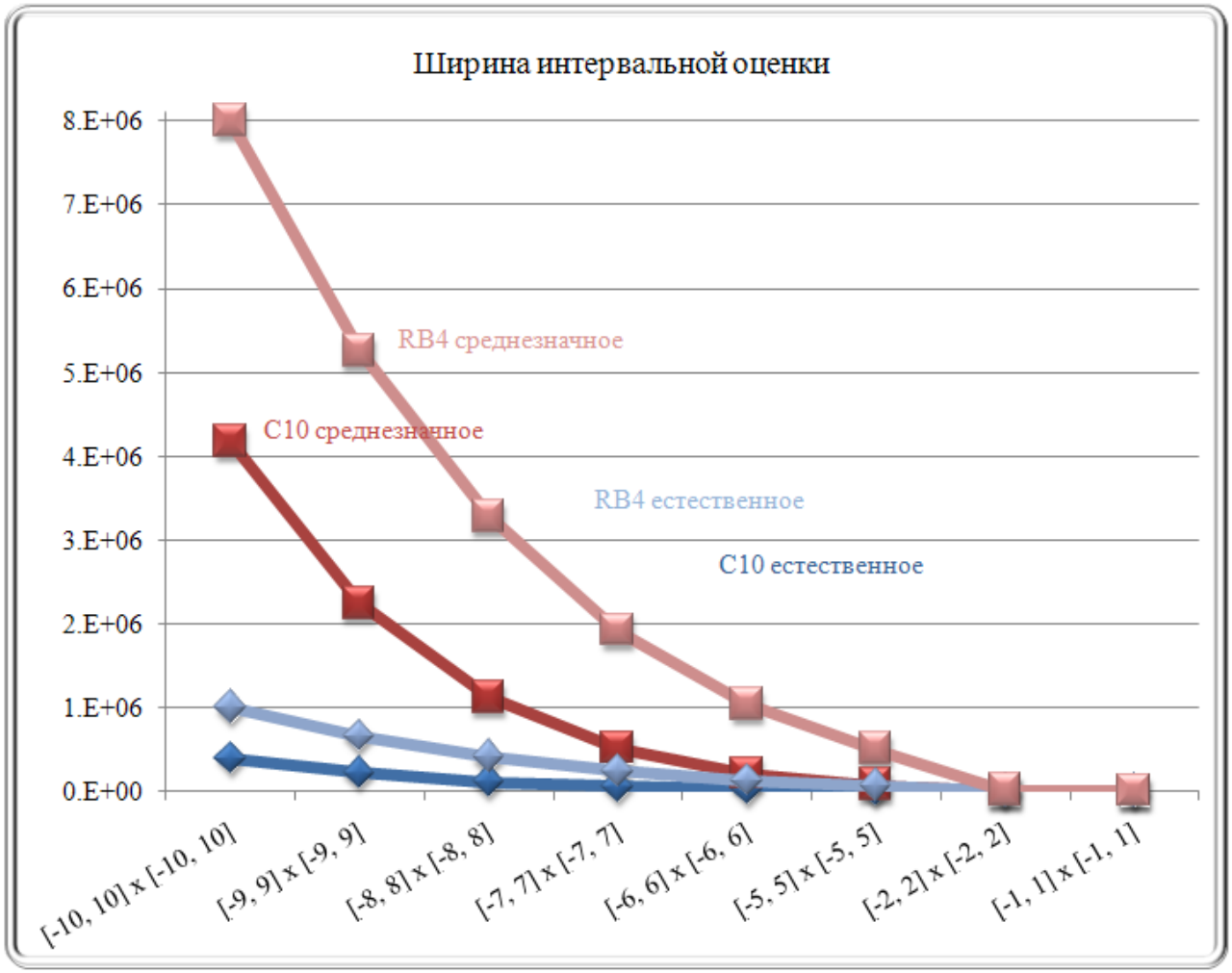


Рис. 1.9: Ширина естественной и среднезначной интервальных оценок на разных целевых функциях

известной под именем четвертой функции Розенброка (Rosenbrock4):

$$f_{RB4}(x) = 100 \cdot \left(x_1^2 - x_2^2\right)^2 + (x_1 - 1)^2. \tag{1.44}$$

Вид функции приведен на рис. 1.10.

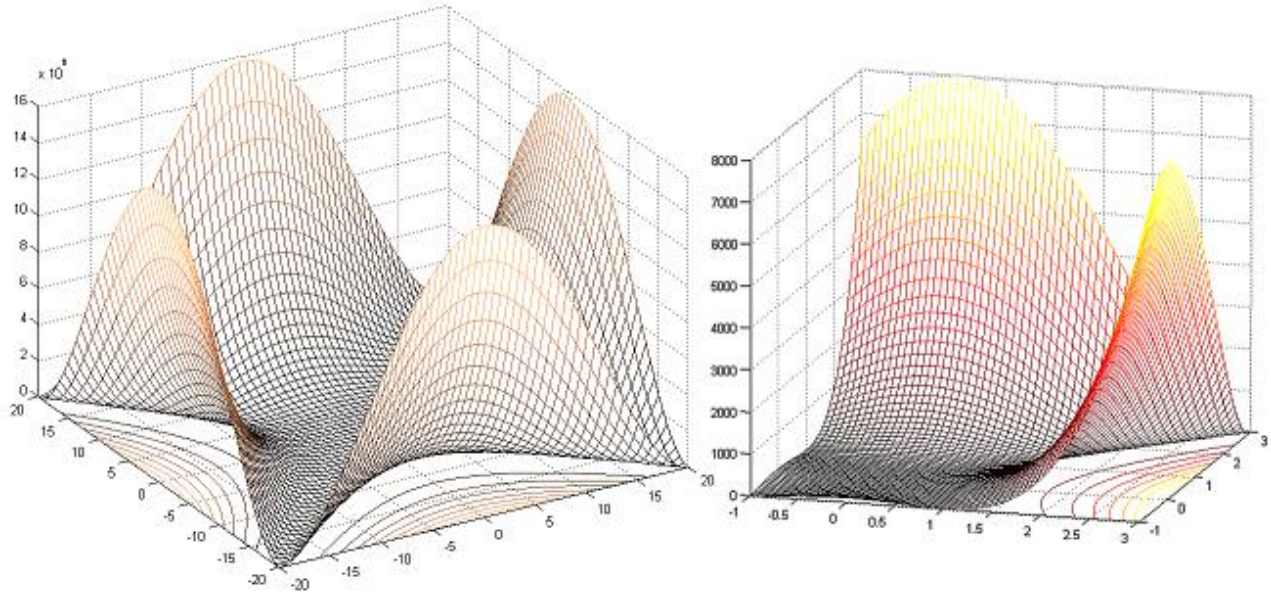


Рис. 1.10: Функция (1.44) «Розенброк4» (RB4).
Общий вид (слева) и поведение вблизи точки глобального минимума (справа)

Мы случайным образом разбили исходную область определения $[-10, 10] \times [-10, 10]$ на сто брусов, как это показано на рис. 1.11 и вычислили интервальные оценки используя естественное, а затем среднезначное интервальные расширения. Результаты приведены на рис. 1.12.

В целом, естественное интервальное расширение выглядит более точным. Такие статистические показатели как сумма ширины всех оценок по всем брусам и средняя ширина интервальной оценки меньше именно для естественного интервального расширения. Во многом это объясняется очень избыточной оценкой, полученной при использовании среднезначной формы для бруса $[0, 5] \times [-2.5, 0]$ (хорошо заметный пик на графике). Интересно, что для рассматриваемой функции среднезначная форма в основном дает более точную верхнюю границу, но при этом склонна занижать нижнюю оценку (см. графики на рис. 1.13)

Следующий эксперимент, проделанный нами, направлен на получение более полной картины и заключается в дроблении исходной области определения ($[-10, 10] \times [-10, 10]$) последовательно на $2, 4, 8, 16, \ldots, 2\,048$ одинаковых подбрусов, равномерно покрывающих начальную область определения и

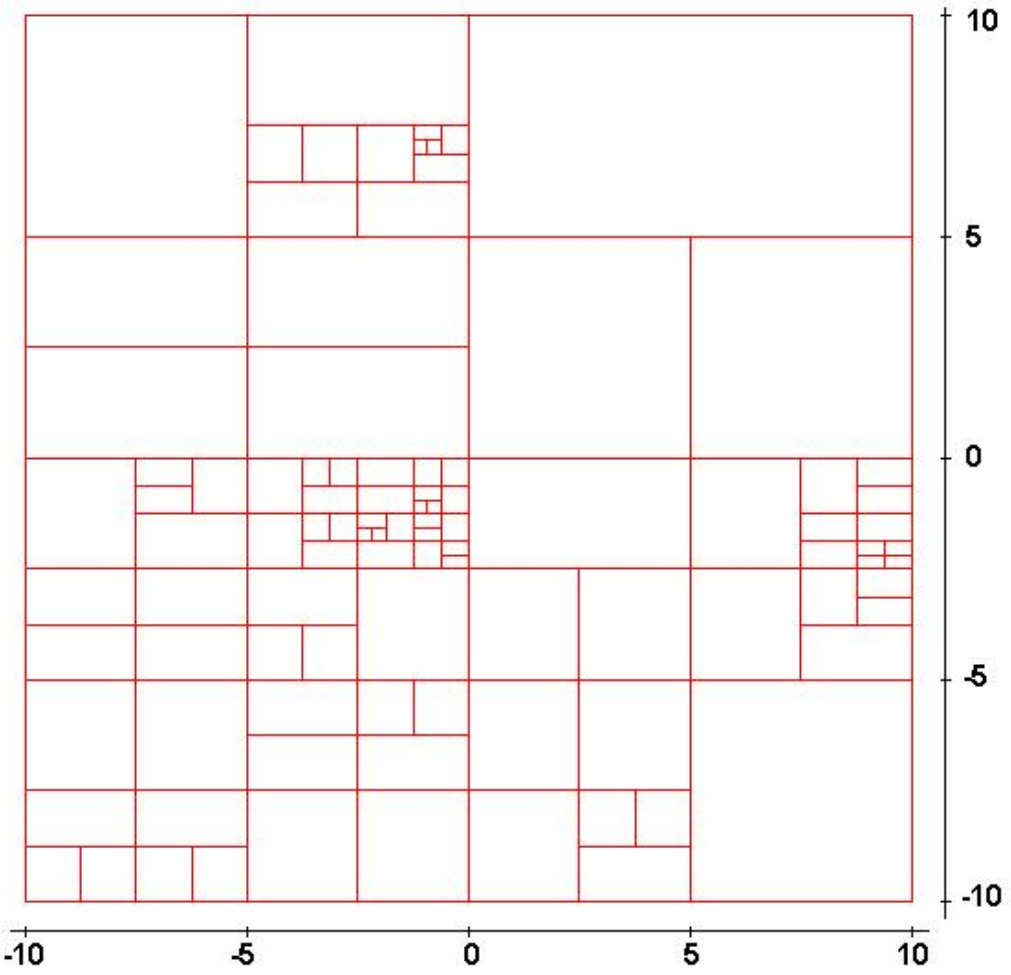


Рис. 1.11: Область определения функции «RB4» случайно разбита на подбрусы следующим образом.

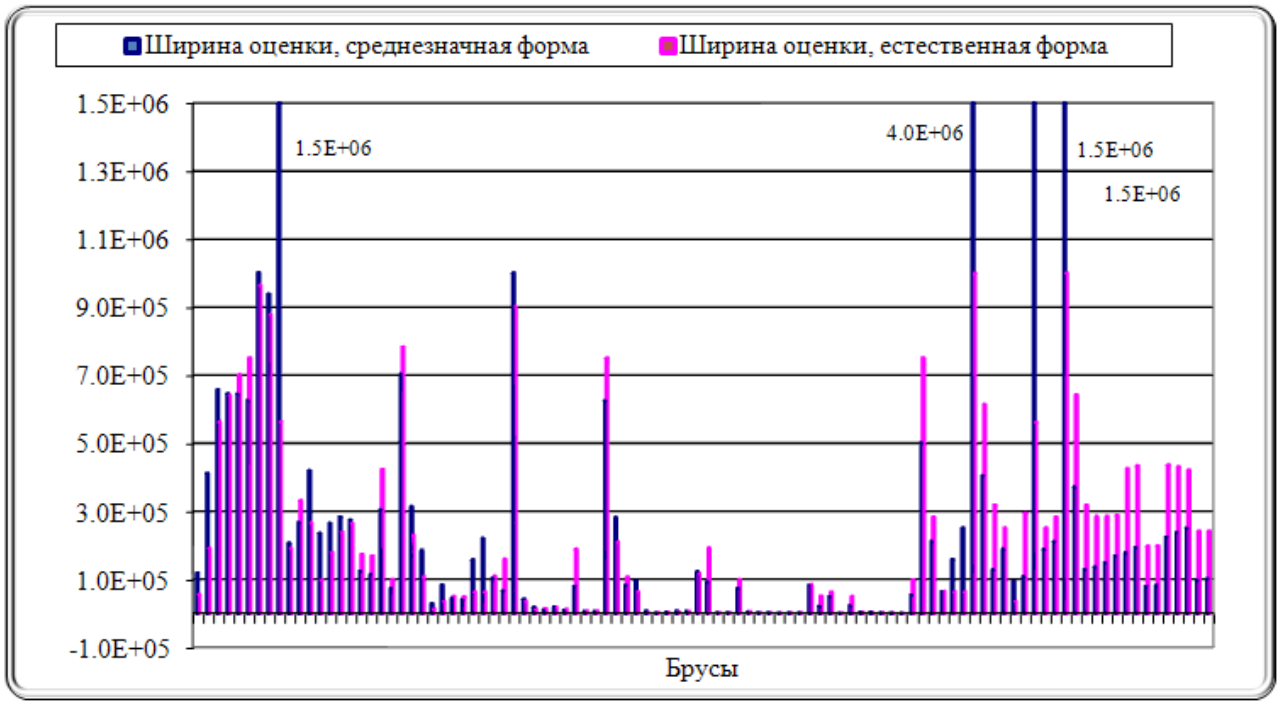


Рис. 1.12: Ширина интервальных оценок на подбрусах случайного разбиения, показанного на рис. 1.11. На оси x выстроены брусы. Способ их упорядочивания в данном случае не важен. Важна лишь ширина полученной интервальной оценки на каждом из брусов

сравнении таких статистических характеристик как суммарная ширина интервальных оценок по всем брусам, средняя ширина интервальных оценок, относительная избыточность оценок. В качестве целевых функций мы использовали уже знакомые «RB4» (1.44), «C10» (1.41), «R10» (1.42), «Tre4» (1.15) и «P5» (1.36).

Результаты приведены в виде графиков на рисунках 1.14 — 1.19. Видно, что на достаточно крупных брусах предпочтительнее использовать естественное интервальное расширение. При уменьшении характерного размера брусов для большинства (но далеко не для всех) целевых функций среднезначная форма позволяет получить более точные оценки.

Исходя из этих результатов, мы рекомендуем использовать следующее интервальное расширение:

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}_{\natural}(\boldsymbol{x}) \cap \boldsymbol{f}_{mv}(\boldsymbol{x}) \tag{1.45}$$

Необходимо отметить, что существуют еще другие, не рассмотренные в настоящей работе способы вычисления интервальной оценки, такие как наклонные и бицентрированные формы. С учётом доказанной в работе [97] следующей теоремы:

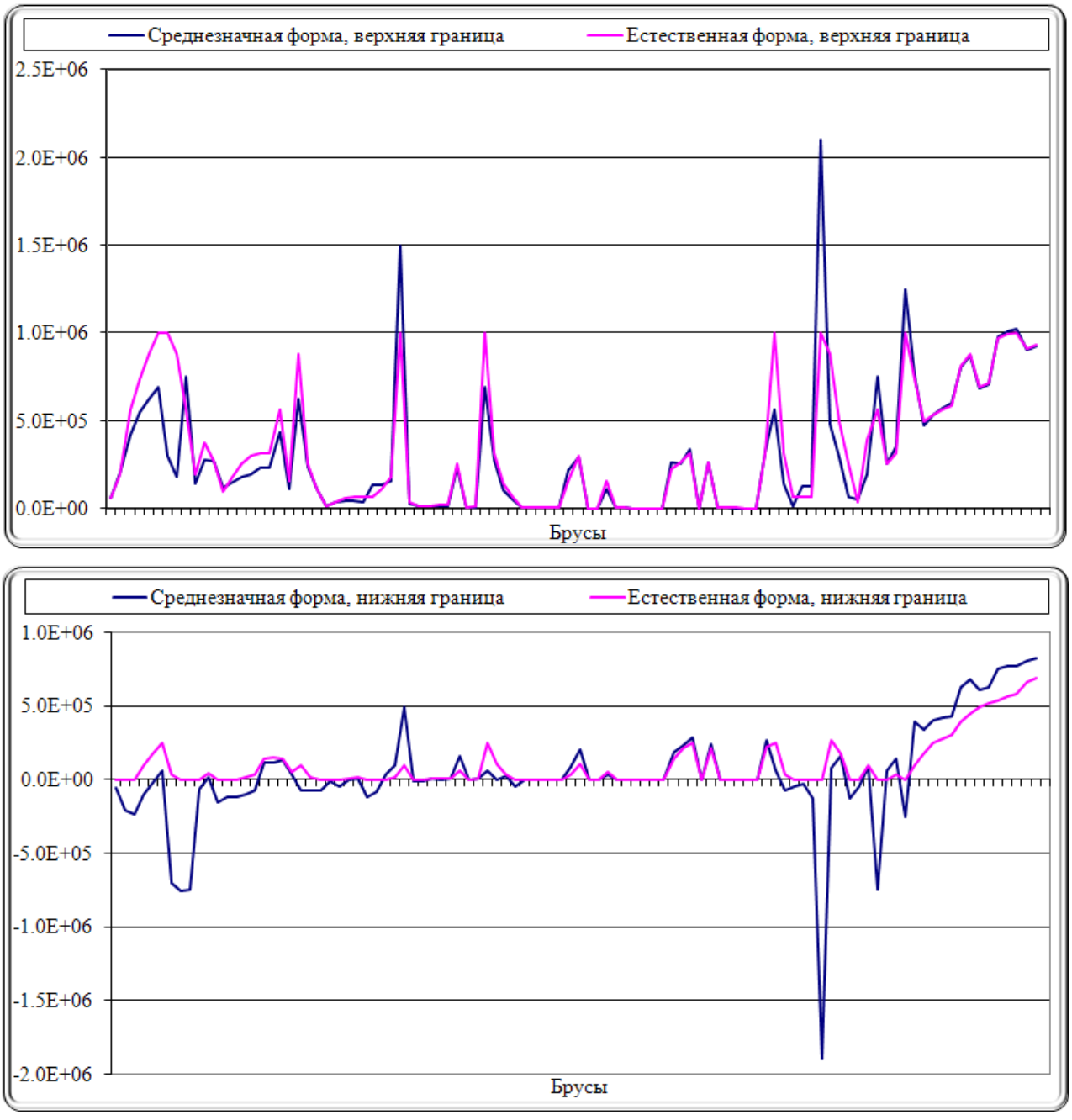


Рис. 1.13: Верхние и нижние оценки значений функции «RB4» на подбрусах случайного разбиения, показанного на рис. 1.11

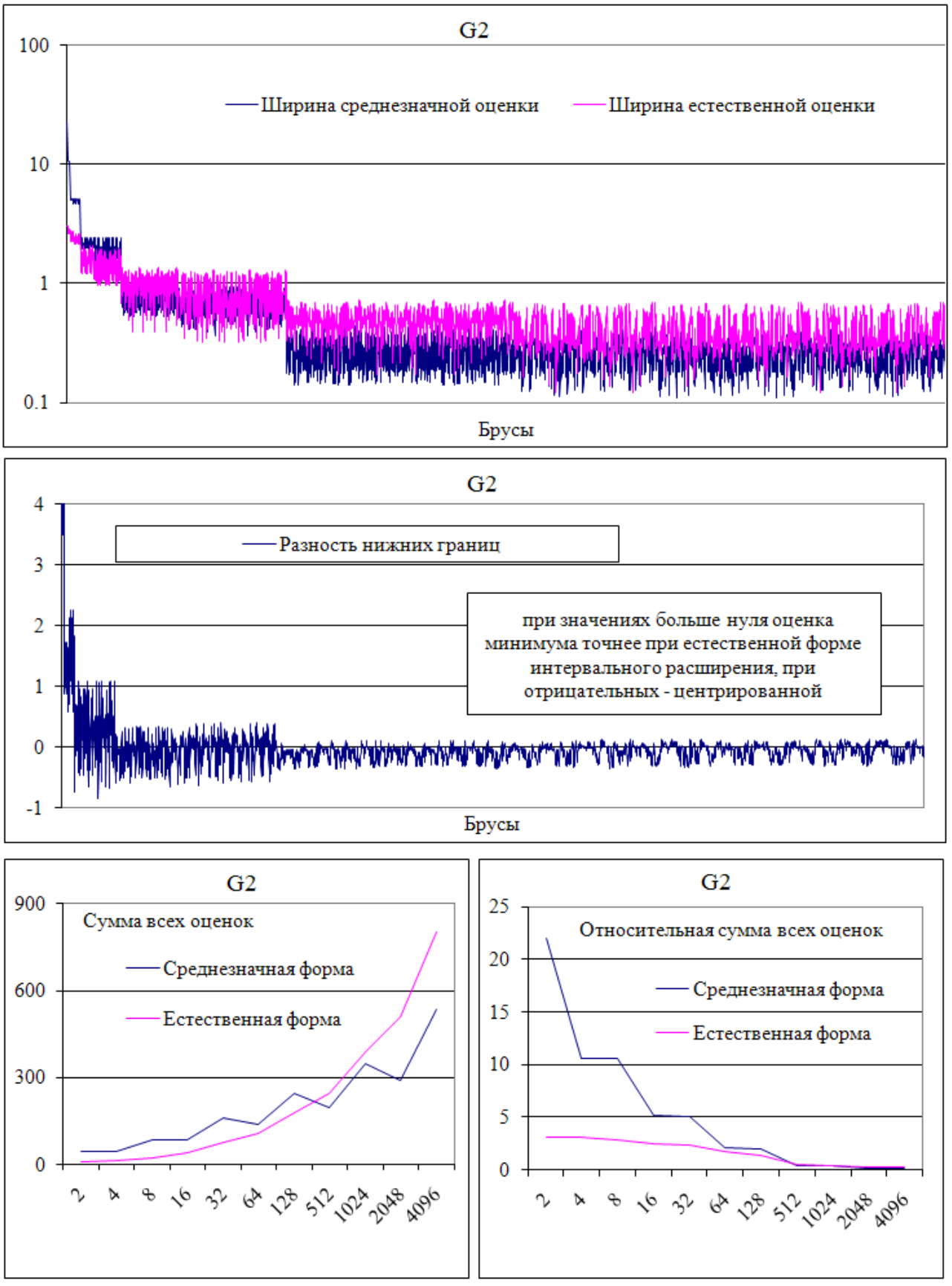


Рис. 1.14: Ширина, разность нижних границ и сумма всех оценок функции (1.43) «G2» в зависимости от количества брусов, на которое раздроблена область определения

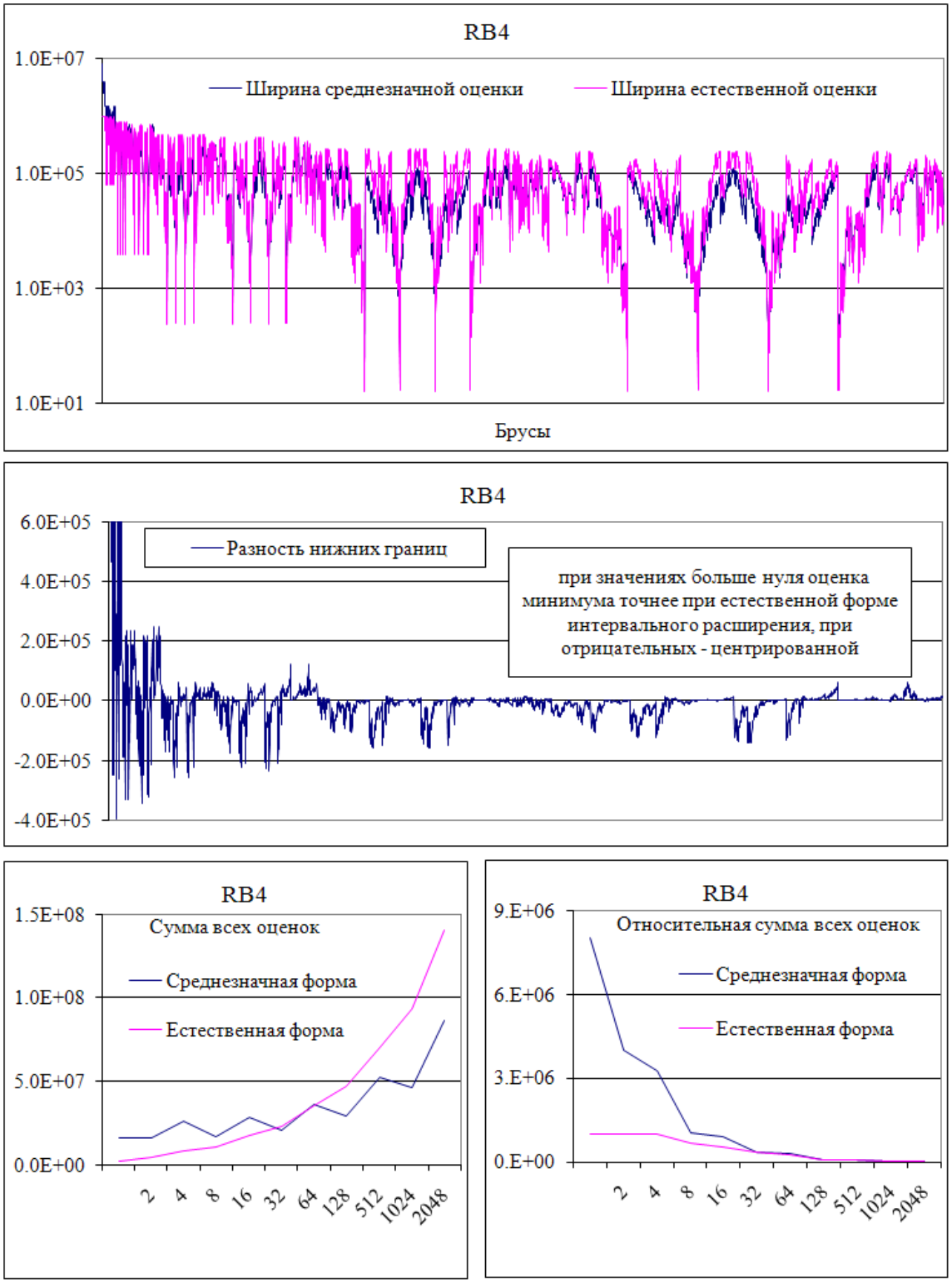
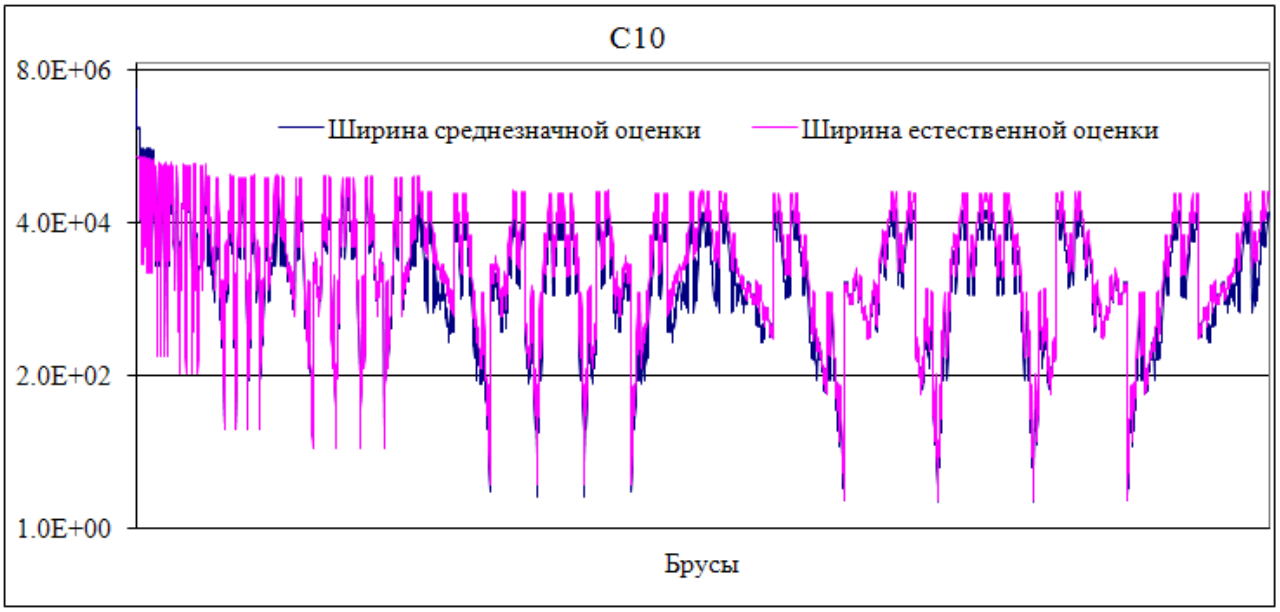
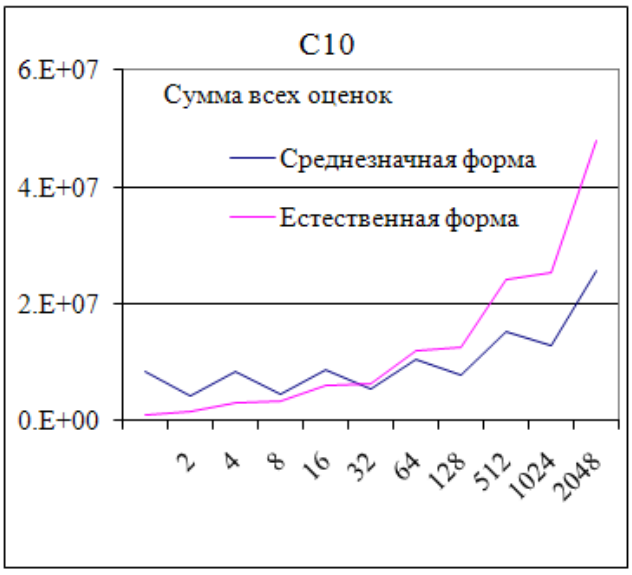


Рис. 1.15: Ширина, разность нижних границ и сумма всех оценок функции (1.44) «RB4» в зависимости от количества брусов, на которое раздроблена область определения





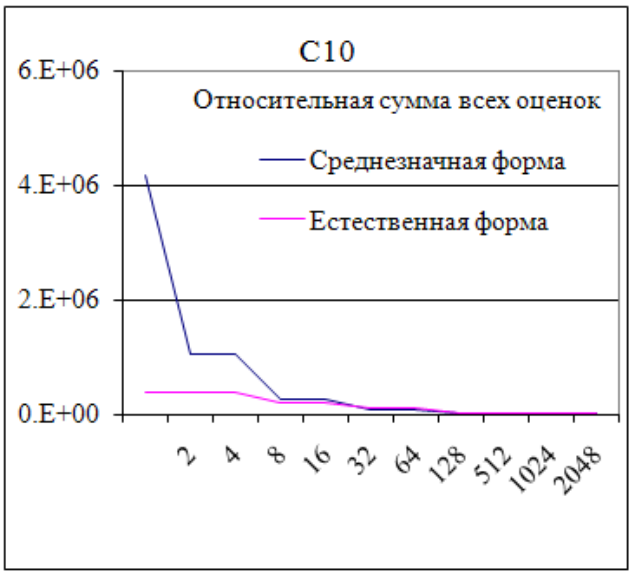
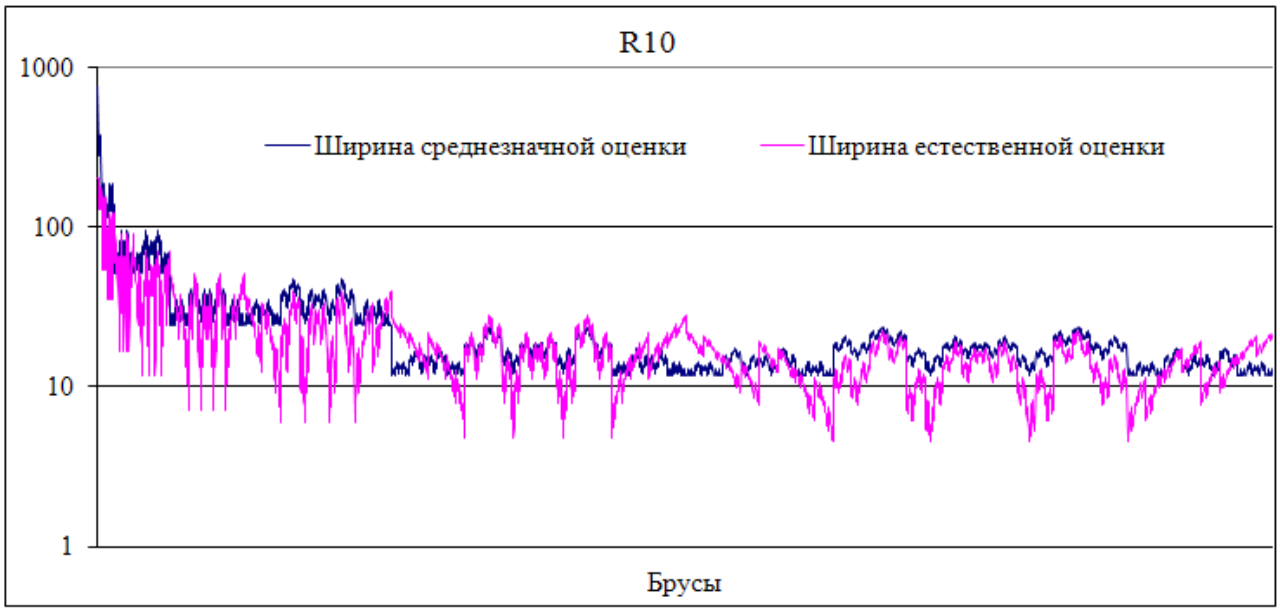
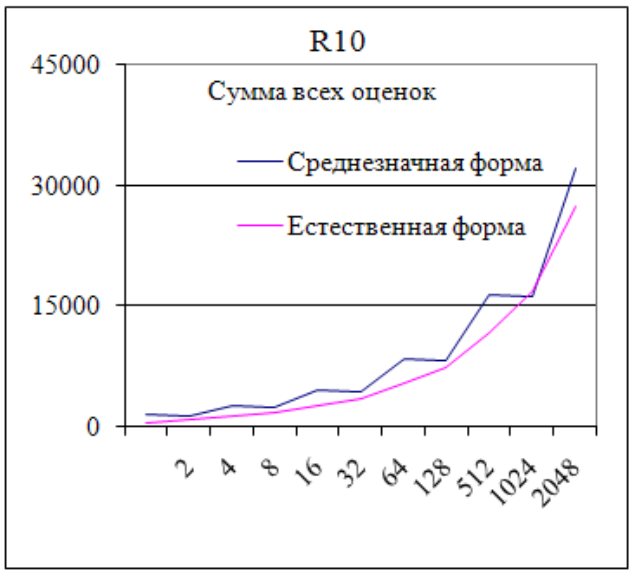


Рис. 1.16: Ширина, разность нижних границ и сумма всех оценок функции (1.41) «C10» в зависимости от количества брусов, на которое раздроблена область определения





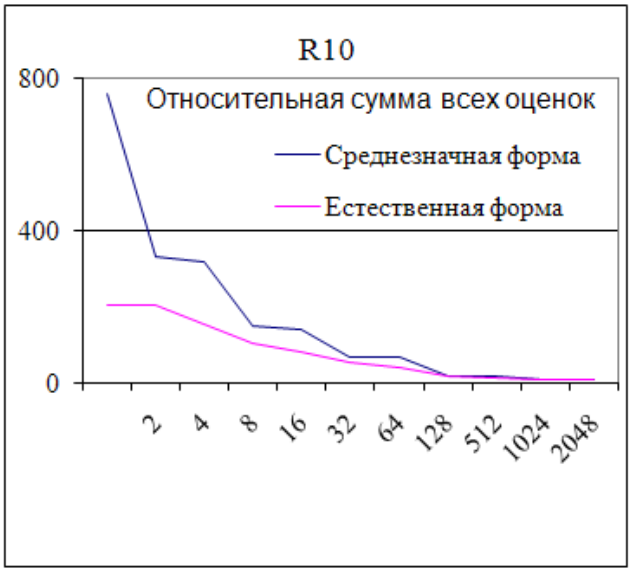
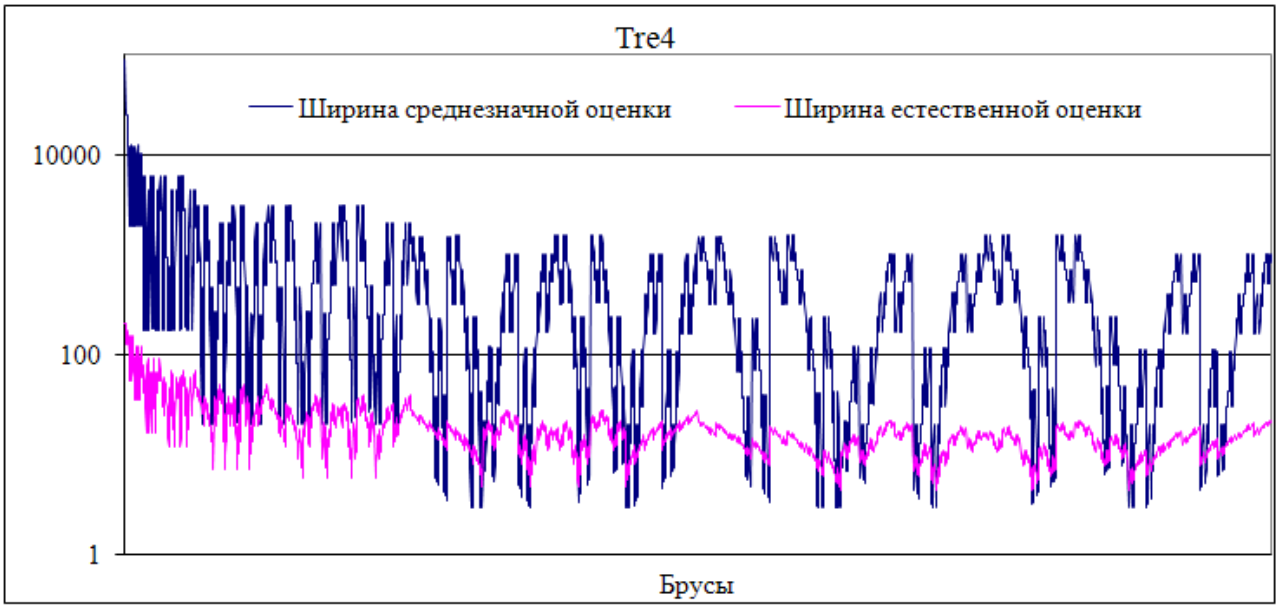
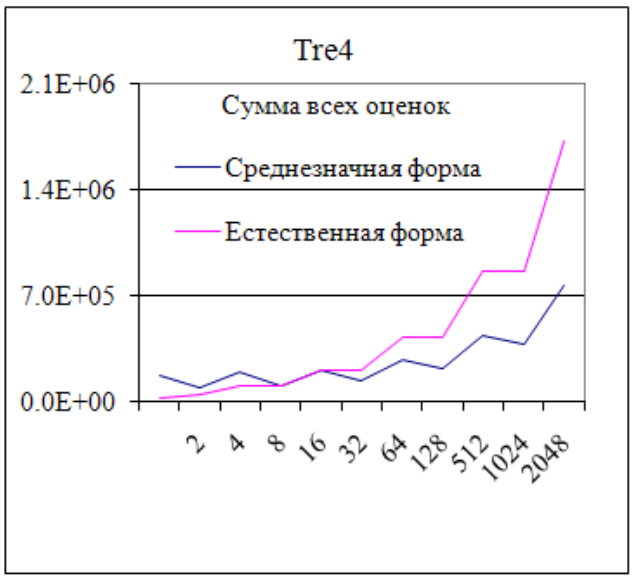


Рис. 1.17: Ширина, разность нижних границ и сумма всех оценок функции (1.42) «R10» в зависимости от количества брусов, на которое раздроблена область определения





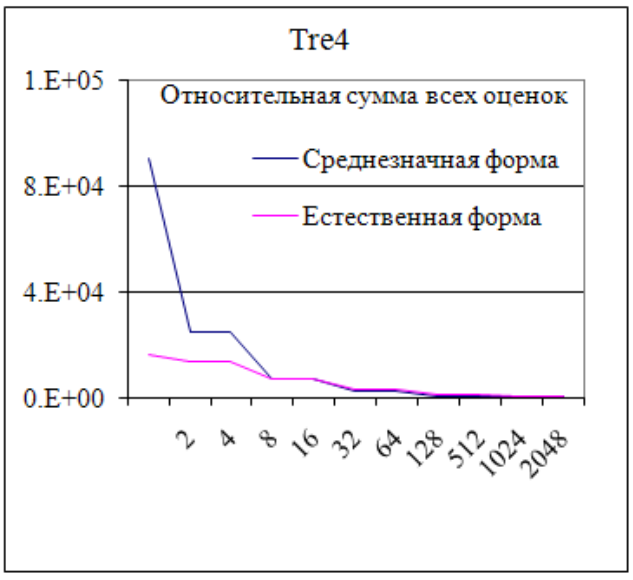
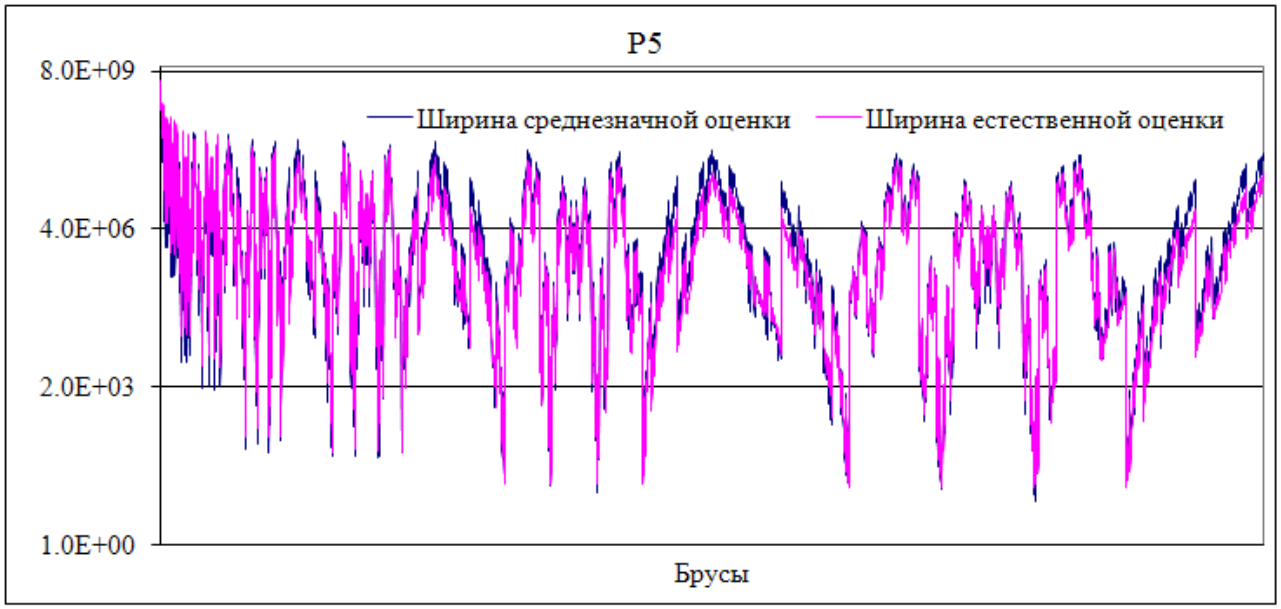
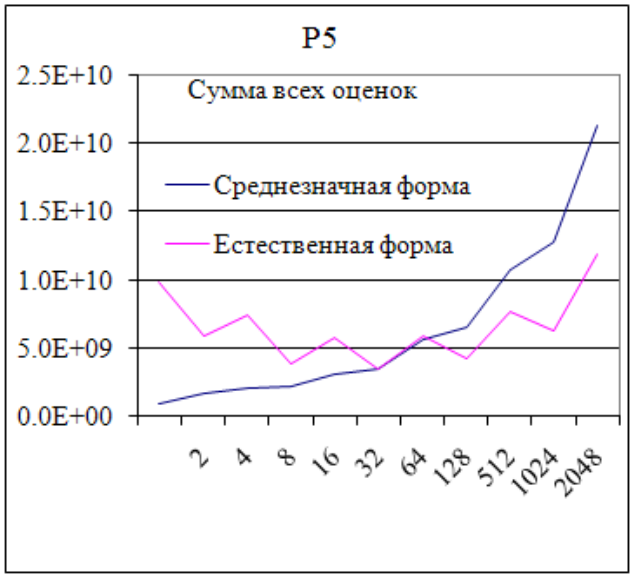


Рис. 1.18: Ширина, разность нижних границ и сумма всех оценок функции (1.15) «Tre4» в зависимости от количества брусов, на которое раздроблена область определения





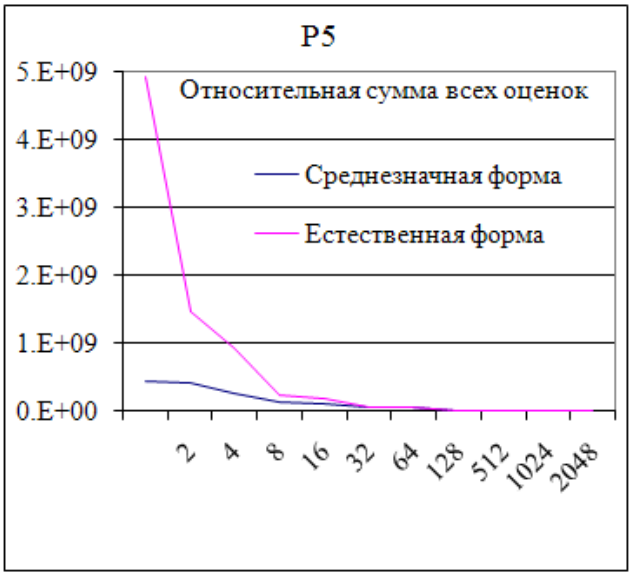


Рис. 1.19: Ширина, разность нижних границ и сумма всех оценок функции (1.36) «P5» в зависимости от количества брусов, на которое раздроблена область определения

Теорема 1.1 *Пусть* $f : \mathbb{R} \to \mathbb{R}$, $\boldsymbol{x} \in \mathbb{IR}$ *и* $\tilde{x} \in \boldsymbol{x}$. *Если при этом* $\boldsymbol{f'}$ *область значений производной* f', *то*

$$\lim_{\text{wid } \boldsymbol{x} \to 0} \frac{\text{wid } \boldsymbol{f'}(\boldsymbol{x})}{\text{wid } \boldsymbol{f}^{\angle}(\boldsymbol{x})} = 2,$$

полученные результаты будут справедливы и для этих способов вычисления интервальных расширений. Но ширина интервала, начиная с которой точность естественного интервального расширения будет уступать этим формам — увеличится.

Итоги главы 1

Основным исследовательским итогом главы 1 диссертации является экспериментальное исследование точности различных интервальных расширений. Теоретические оценки точности интервального оценивания областей значений функций с помощью различных интервальных расширений хорошо известны, но в настоящей работе выполнено широкомасштабное экспериментальное исследование, которое позволяет более аккуратно подходить к выбору тех или иных форм интервальных расширений при решении практических задач. Сделанные выводы являются неочевидными и свидетельствуют о том, что зачастую естественное интервальное расширение более предпочтительно, чем центрированные формы, имеющие более высокий асимптотический порядок точности.

Глава 2

Критический анализ интервальных методов глобальной оптимизации

В этой главе работы представлен критический обзор существующих на настоящее время интервальных методов глобальной оптимизации, выявлены их слабые стороны, намечены пути совершенствования. В нашем исследовании мы используем как теоретические (рациональные) рассуждения, так и экспериментальные методы, когда те или иные выводы делаются на основе анализа исполнения соответствующих алгоритмов. Материал главы является новым, и в рассматриваемом нами аспекте интервальные оптимизационные методы с достаточной полнотой практически никем ранее не изучались.

2.1 Обзор классических (точечных) методов

Напомним общую постановку задачи оптимизации. Пусть X — некоторое множество (которое может совпадать со всем пространством $\mathbb{R}^n$ или же быть задано какими-то ограничениями — равенствами, неравенствами и т. п.), f — заданная на X функция, называемая *целевой функцией*. Требуется найти приближение

$$f^* = \inf_{x \in X} f(x). \tag{2.1}$$

Так формулируется задача минимизации в классической (точечной) постановке. (В §2.2 можно будет сравнить её с постановкой интервальной задачи глобальной оптимизации.)

Методом (алгоритмом) решения задачи минимизации называют способ построения последовательности точек из X, сходящейся к некоторой точке, в которой значение (2.1) точно или приближенно достигается. Типы сходимости указанной последовательности могут быть различными — от сходимости по значению функции f до сходимости с некоторой вероятностью. Посколь-

ку на практике количество вычислительной работы всегда ограниченно, то ограничиваются конечным числом членов указанной последовательности и пытаются их построить так, чтобы с наименьшими (или ограниченными) вычислительными затратами достичь желаемой (соответственно, максимально возможной) точности. В зависимости от вида алгоритма глобального поиска для построения точек может использоваться априорная информация о целевой функции и множестве оптимизации, а также значения целевой функции и, возможно, её производных в предшествующих точках последовательности и, может быть в некоторых вспомогательных точках.

Сложность задачи оптимизации определяется свойствами целевой функции на интересующей нас области определения (допустимое множество определения). Имеет место определенная дуальность свойств области X и функции f. Задачу со сложной целевой функцией можно переформулировать так, что функция станет простой (например, линейной), но при этом все сложности будут перенесены на область определения. Возможна и противоположная трансформация. Чаще рассматривают задачи с достаточно простыми областями X, а все сложности переносят на целевую функцию. Иногда множество X задается с помощью различных ограничений типа равенств и неравенств и имеет сложную структуру. В этих случаях обычно используют различные методы штрафных функций, выпуклую аппроксимацию множества, операции проектирования и сводят исходную задачу к задаче (или последовательности задач) оптимизации на множестве (или множествах) более простой структуры. Одним из основных факторов, определяющих сложность задачи оптимизации является размерность множества X. Наиболее развиты методы решения одномерных задач, однако для большинства прикладных задач более важными являются методы многомерной глобальной оптимизации.

Идея большинства классических методов глобальной оптимизации заключается в том, чтобы тем или иным способом оценить значения целевой функции $f(\cdot)$ на некотором, по-возможности наиболее представительном, подмножестве точек из допустимого множества, и, фактически, различие методов заключается в способах выбора этих точек. Так как нам не известно, где именно на области определения можно найти глобальные минимумы (максимумы), необходимо использовать некоторую стратегию распределения этих точек. Любую такую стратегию принято называть *глобальной техникой* [114].

Далее, вполне вероятно, что в окрестности точек $x_1, \ldots x_n$, указанной глобальной техникой могут быть достигнуты лучшие значения целевой функции. Для этой цели почти все методы глобальной оптимизации к полученным точкам применяют алгоритмы локального поиска. Метод, осуществляющий та-

кой локальный спуск, часто называют *локальной техникой*.

Помимо глобальной и локальных техник поиска для любого численного алгоритма необходимо определить условия остановки. Это ключевой момент, так как без какой-либо дополнительной информации или предположений о задаче невозможно сделать выводы о точности решения, полученного за некоторое ограниченное число итераций алгоритма. И мы должны воспринимать полученное с помощью численного метода решение как приближённое.

Существует множество специальных классификаций методов глобальной оптимизации [55]. По большóму счету их можно разделить на две категории: детерминированные [10, 17, 18, 19, 21, 22, 85] и стохастические [78, 10, 21, 28, 91, 86].

Все методы многоэкстремальной оптимизации можно условно разделить на две группы. Для первых существуют какие-либо точные утверждения об их сходимости к глобальному оптимуму, для вторых приходится ограничиваться некоторыми правдоподобными рассуждениями об их разумном поведении в многоэкстремальной ситуации. Будем говорить в первом случае о *точных* методах, во втором — об *эвристических*.

Придумать точные методы нетрудно, однако их ценность, как правило, невелика. В [56] приводится характерный пример.

Теорема 2.1 *Пусть $f(x)$ — непрерывная функция на брусе $\boldsymbol{X} \subset \mathbb{R}^n$, x^k — последовательность независимых равномерно распределённых на $\boldsymbol{X}$ случайных векторов. Тогда* $\min_{l \leqslant i \leqslant k} f(x^i) \to \min_{x \in \boldsymbol{X}} f(x)$.

Доказательство, приводимое там же, несложно. По теореме Вейерштрасса существует точка x^* глобального минимума $f(x)$ на $\boldsymbol{X}$. Пусть $\epsilon > 0$ произвольно, в силу непрерывности $f(x)$ найдётся окрестность U точки x^*, для которой $f(x) < f(x^*) + \epsilon$ при $x \in U$. Пусть v — объём $U \cap \boldsymbol{X}$, V — объём $\boldsymbol{X}$, тогда $v > 0$ в силу открытости U. Вероятность попадания точки x^i в $U \cap \boldsymbol{X}$ равна v/V; вероятность, что хотя бы одна точка из точек $x^1, \ldots, x^k$ попадёт в $U \cap \boldsymbol{X}$ равна $p_k = 1 - (1 - v/V)^k$. Очевидно, что $p_k \to 1$ при $k \to \infty$. Это и означает сходимость по вероятности. Тем самым теорема доказана.

Доказанная теорема столь же проста и универсальна, сколь и бессодержательна. Оценим число вычислений функции, требуемое для отыскания решения с небольшой точностью, для одного примера. Пусть $x = (x_1, \ldots, x_{10}) \in \mathbb{R}^{10}$, $f(x) = \max_{0<j<11} x_j$, $\boldsymbol{X} = \{\, x \mid 0 \leqslant x_j \leqslant 1, j = 1, \ldots, 10 \,\}$, и зададимся точностью $\epsilon = 10^{-2}$. Тогда $x^* = 0$, $f(x^*) = 0$, $v = (10^{-2})^{10} = 10^{-20}$, $V = 1, p_k \sim k \cdot 10^{-20}$, то есть, чтобы вероятность отыскания x^* с точностью до 1% была равна хотя бы 10%, потребуется $\sim 10^{19}$ итераций. Иначе говоря,

метод случайного поиска (в форме, сформулированной в обсуждаемой теореме) абсолютно непригоден для нахождения глобального оптимума уже для размерностей порядка 10. Таким образом сама по себе теорема о сходимости отнюдь не гарантирует работоспособности метода.

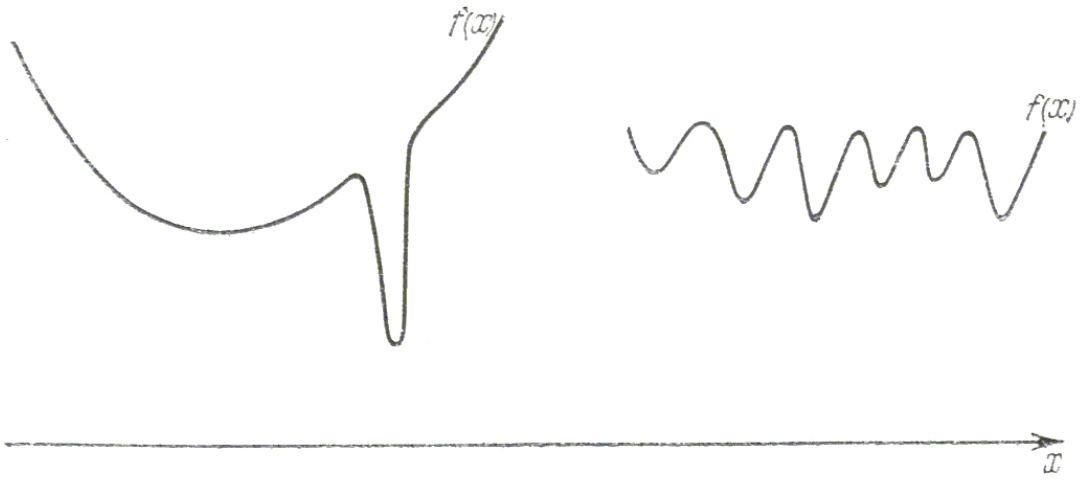


Рис. 2.1: Примеры функций, для которых трудно найти глобальный оптимум

В то же время, должно быть ясно, что для произвольных непрерывных или даже гладких функций невозможен метод существенно лучший, чем в ранее сформулированной теореме. На рис. 2.1 приведены примеры функций, для которых глобальный минимум нельзя найти иначе, чем путём перебора её значений на достаточно мелкой сетке. Один из возможных способов корректного математического обоснования получающихся при этом алгоритмов состоит в сужении класса рассматриваемых функций до так называемых *липшицевых функций*, удовлетворяющих *условию Липшица*

$$|f(x) - f(y)| < L\,\|x - y\|,$$

с некоторой известной константой L. При минимизации такого рода функций можно руководствоваться следующим соображением. Пусть уже найдено рекордное значение $f(x)$ по $k-1$ предыдущим итерациям, $\varphi_{k-1} = \min_{0<i<k} f(x^i)$, и вычислено $f(x^k)$. Тогда, если $f(x^k) < \varphi_{k-1}$, то улучшается значение рекорда: $\varphi_k = f(x^k)$, если же $f(x^k) > \varphi_{k-1}$, то в шаре $\{\, x \in \mathbb{R}^n \mid \|x - x^k\| < L^{-1}(f(x^k) - \varphi_{k-1})\}$ заведомо не может быть точки глобального минимума, что приводит к сокращению области возможной локализации минимума.

Таким образом отыскать оценку глобального оптимума детерминированным алгоритмом можно, сравнив значения целевой функции в разных точках области определения. При этом достоверность результата естественным образом зависит от количества и положения выбранных точек. В работах [17, 18, 19, 60] описывается детерминистские методы, которые условно можно назвать методами «неравномерных покрытий». Они основаны на том, что если мы каким-либо образом найдём константу Липшица интересующей нас целевой функции, то сможем оценивать возможные значения функции между узлами сетки. В свою очередь, это позволит оценивать точность получаемого результата, а также строить адаптивные сетки, тем самым снижая количество необходимых вычислений. Подытоживая, можно сказать, что основная идея детерминистских методов так или иначе состоит в получении глобального решения посредством исчерпывающего поиска на всем допустимом множестве. Вследствие этого большинство детерминированных методов теряют эффективность и надёжность с возрастанием размерности задачи. Более того, чтобы гарантировать успех, такие методы требуют выполнения дополнительных предположений, наложенных на целевую функцию.

Стохастические алгоритмы позволяют в какой-то степени уйти от проблем детерминированных алгоритмов. Одним из популярных принципов организации стохастических методов является оценивание значение функции цели в случайных точках допустимого множества с последующей обработкой получившейся выборки. Как следствие, стохастические методы не гарантируют, что будет найден именно глобальный оптимум.

Среди классических детерминированных методов можно выделить методы, основанные на *редукции размерности задачи*. При одном из подходов многомерная задача сводится к серии вложенных одномерных задач (*многошаговая схема* редукции размерности). Другие алгоритмы используют *отображение многомерной области на отрезок с помощью разверток* на основе аппроксимаций кривых Пеано [29, 101]. Последние результаты в этой области получены Р.Г. Стронгиным [62, 63, 64] и связаны с применением *множественных разверток*, позволяющих более точно передать близость точек измерений в многомерном пространстве при его отображении на отрезок. Вкратце, методы редукции этого типа так или иначе основываются на использовании заполняющих пространство кривых. Идея построения кривых Пеано представлена на рис. 2.2 слева. В пределе, когда размеры брусов, центрами которых являются вершины ломаной Пеано, стремятся к нулю, кривая Пеано проходит через все точки области. Однако, при этом вид целевой функции меняется кардинально. На рис. 2.2, заимствованном из книги [22], справа

для примера показано, что происходит с квадратичной функцией после её редукции к одномерной.

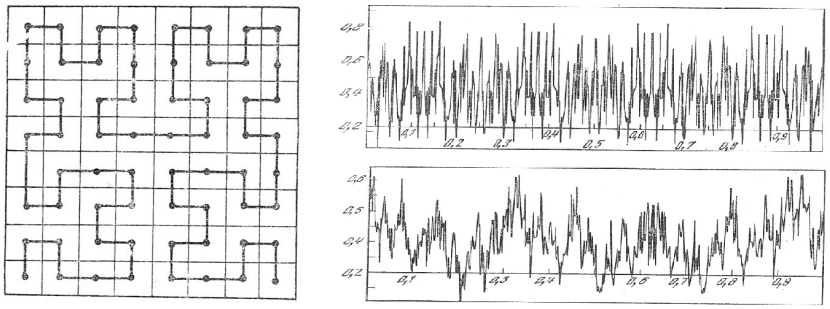


Рис. 2.2: Слева: Иллюстрация построения кривой Пеано.
Справа: Одномерный образ квадратичной функции пяти переменных, построенный для разных шагов как кусочно-линейная аппроксимация по 500 равномерно распределённым точкам.

Из других идей построения методов глобальной оптимизации функций отметим развиваемые А.С. Стрекаловским методы, основанные на представлении целевой функции в виде разности выпуклых [61].

Также существуют методы решения многомерных задач глобальной оптимизации, опирающиеся на нахождение кусочно-линейных выпуклых и вогнутых опорных функций, развиваемые О.В. Хамисовым [20]. Суть метода сводится к отысканию для целевой функции $f(\boldsymbol{x})$ выпуклой опорной функции-мажоранты и вогнутой опорной функции-миноранты $f^+(\boldsymbol{x},\boldsymbol{y})$ и $f^-(\boldsymbol{x},\boldsymbol{y})$ таких, что выполняются следующие требования:

1) $f^+(\boldsymbol{x},\boldsymbol{y})$ выпукла и непрерывна по $\boldsymbol{x}$ при любом фиксированном $\boldsymbol{y}$;
2) $f^-(\boldsymbol{x},\boldsymbol{y})$ вогнута и непрерывна по $\boldsymbol{x}$ при любом фиксированном $\boldsymbol{y}$.

Доказывается, что если для целевой функции $f(\boldsymbol{x})$ удаётся найти такие опорные функции, то она является непрерывной и удовлетворяет условию Липшица. Следовательно, в этом случае исходную задачу глобальной оптимизации можно свести к задаче липшицевой оптимизации, только вместо константы Липшица главную роль в дальнейшем играют введенные выше опорные функции.

Случайный поиск возник не на пустом месте: его появлению предшествовал период предыстории, когда инструмент случайности оттачивался на решении различного рода прикладных и теоретических задач. Рассмотрим их.

I. Наиболее очевидной и всем известной схемой, эксплуатирующей идею случайного поиска, является способ, обычно называемый *методом проб и ошибок.*

Как видно из названия, метод состоит из серии случайных экспериментов (проб) с некоторыми объектами, причем допустим неудачный результат экспериментов, то есть неудачи не имеют фатального значения, а их последствия легко устранимы. Иначе говоря, стоимость исправления допущенной ошибки невелика. Метод напрямую реализует идею случайности — пробы (эксперименты) всегда случайны, для их организации нет никаких иных соображений.

II. *Метод Монте-Карло* появился в 1943 г. и с тех пор стал, одним из наиболее популярных вычислительных методов. Этот численный метод основан на получении большóго числа реализаций стохастического процесса, который формируется таким специальным образом, чтобы его вероятностные характеристики совпадали с реальными величинами решаемой задачи. Методы этого типа нашли чрезвычайно широкое применение для оценки многомерных интегралов, решения интегральных уравнений, являются основой для моделирования многих сложных процессов и т. д.

Идея метода состоит в том, чтобы представить интересующую нас величину (решение уравнения и т. п.), как вероятностную характеристику какого-то стохастического процесса, а затем выполнить прямое статистическое моделирование некоторого ансамбля его реализаций. На основании обработки результатов этого моделирования (вычисление выборочного среднего, дисперсии и т. п.) и будет получен ответ к поставленной задаче. В ряде простых случаев, локальные свойства построенного процесса удается описать в виде регулярных дифференциальных соотношений и получить дифференциальное уравнение, интегрирование которого позволяет определить глобальные свойства процесса. Но зачастую для сложных процессов в лучшем случае получается лишь стохастическое дифференциальное уравнение, интегрирование которого представляет собой весьма трудную задачу, а в худшем — лишь правила перехода процесса из одного состояния в другое. Эти правила носят стохастический характер, то есть можно «разыграть» с помощью прямого численного моделирования компьютером конкретные случайные реализации переходов. Тем самым удастся построить разные возможные траектории процесса.

Как видно, суть метода Монте-Карло состоит в многократном моделировании («разыгрывании») случайного поведения процесса и принятии каких-то решений на этой основе. Для разыгрывания нужно уметь многократно моделировать локальное поведение процесса с целью опреде-

ления его интегральных, глобальных характеристик.

III. *Метод рандомизации* предложен в 30-х годах Фишером, заложившим основы математической теории эксперимента, и по сей день широко используется в планировании эксперимента. Этот метод также опирается на специально создаваемую случайность. Суть подхода заключается в следующем.

Пусть необходимо построить модель изучаемого явления, зависящего от ряда факторов. Для простоты положим, что таких факторов два — x_1 и x_2. Оба находятся в распоряжении исследователя, причем интерес представляет влияние лишь первого (x_1), а второй (x_2) является досадной помехой, которую следует исключить. Таким образом, исследователя интересует модель $y = f(x_1)$, а в эксперименте он наблюдает отдельные реализации зависимости $y = f_0(x_1, x_2)$, где f_0 — оператор объекта. Как видно, задача заключается в синтезе модельного оператора f. Прежде всего исследователь строит план эксперимента, т. е. фиксирует конкретные значения обоих факторов, при которых необходимо поставить эксперимент и определить реакцию исследуемого явления. План по первому фактору должен быть таким, чтобы как можно точнее определить модель f. А что делать со вторым фактором, который не входит в модель? Его следует исключить. Если оператор f_0 объекта известен, то исключение реализуется путем обычного осреднения по исключаемому фактору:

$$f(x_1) = \int f_0(x_1, x_2) p_2(x_2) dx_2,$$

где $p_2(x_2)$ — плотность распределения фактора x_2.

2.2 Интервальные методы

Итак, перед нами стоит задача найти глобальный оптимум вещественнозначной функции $f : \mathbb{R}^n \supset X \to \mathbb{R}$ на некоторой подобласти $\boldsymbol{x}$ её области определения X. При этом $\boldsymbol{x}$ представляет собой прямоугольный брус, с параллельными координатным осям сторонами:

$$\boxed{\text{найти} \quad \min_{x \in \boldsymbol{x}} f(x)} \tag{2.2}$$

В случае, когда нам априори не известен характер поведения целевой функции f и структура её локальных экстремумов, решение задачи (2.2) неизбеж-

но потребует в том или ином виде перебор и сравнение «всех точек» области определения.

Наилучшим решением будет найти некие x' и y', наиболее точно приближающие точный результат. То есть такие, что расстояния $\mid x'-x^* \mid$ и $\mid y'-y^* \mid$ будут «малы». Таким образом, мы получаем следующую постановку задачи.

Для функции $f(x)$, заданной на некоторой области определения $x_0 \le x \le x_1$, надо найти $\boldsymbol{X} := [\underline{x}, \overline{x}]$ и $\boldsymbol{Y} := [\underline{y}, \overline{y}]$, такие что:

(i) $x' \in \boldsymbol{X}$ и $y' \in \boldsymbol{Y}$ и

(ii) $\boldsymbol{X}$ и $\boldsymbol{Y}$ настолько узки, насколько это нужно для данной задачи.

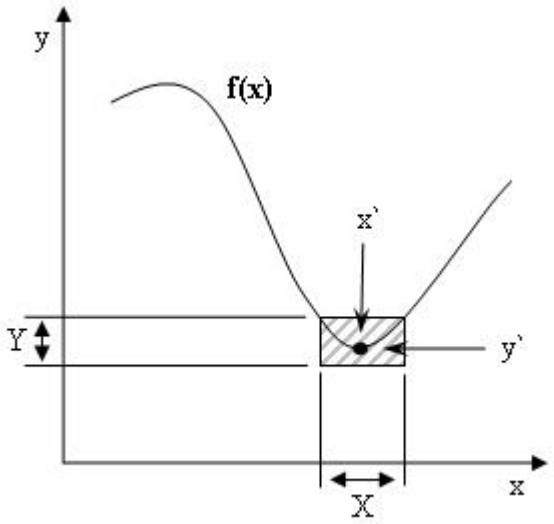


Рис. 2.3: Иллюстрация интервальной постановки задачи.

Ещё раз обрисуем основной принцип итерационных интервальных методов глобальной оптимизации. Так или иначе, все методы этого типа существенно эксплуатируют тот факт, что большинство интервальных оценок области значений функции — асимптотически точные:

$$\operatorname{dist}\left(\boldsymbol{f}(\boldsymbol{x}), \operatorname{range}_{\boldsymbol{x}} f\right) \longrightarrow 0 \qquad \text{при} \quad \operatorname{wid}\left(\boldsymbol{x}\right) \longrightarrow 0. \tag{2.3}$$

Это означает, что при уменьшении размеров области определения точность интервального расширения функции увеличивается. К сожалению, при этом не следует ожидать, что при уменьшении размеров области определения, скажем, вдвое, и точность интервальной оценки улучшится на столько же, или хотя бы пропорционально. Это лишь означает, что при принудительном дроблении бруса области определения точность интервальной оценки будет возрастать.

В самом деле, если разбить исходный брус $\boldsymbol{x}$ на два подбруса $\boldsymbol{x}'$ и $\boldsymbol{x}''$, дающие в объединении весь $\boldsymbol{x}$, то есть такие, что $\boldsymbol{x}' \cup \boldsymbol{x}'' = \boldsymbol{x}$, то

$$\{\, f(x) \mid x \in \boldsymbol{x} \,\} = \{\, f(x) \mid x \in \boldsymbol{x}' \,\} \cup \{\, f(x) \mid x \in \boldsymbol{x}'' \,\}.$$

Соответственно, можно вычислить интервальные расширение на каждом подбрусе и в качестве новой оценки минимума целевой функции на $\boldsymbol{x}$ взять

$$\min\{\underline{\boldsymbol{f}(\boldsymbol{x}')}\,,\ \underline{\boldsymbol{f}(\boldsymbol{x}'')}\}$$

и она будет, вообще говоря, более точна, чем исходная оценка $\boldsymbol{f}(\boldsymbol{x})$, так как у брусов $\boldsymbol{x}'$ и $\boldsymbol{x}''$ размеры меньше, чем у исходного $\boldsymbol{x}$. Брусы-потомки $\boldsymbol{x}'$ и $\boldsymbol{x}''$ можно, в свою очередь, опять разбить на более мелкие части, найти для них интервальные расширения и далее уточнить оценку для минимума, потом снова повторить процедуру и так далее, примерно как на рис. 2.4.

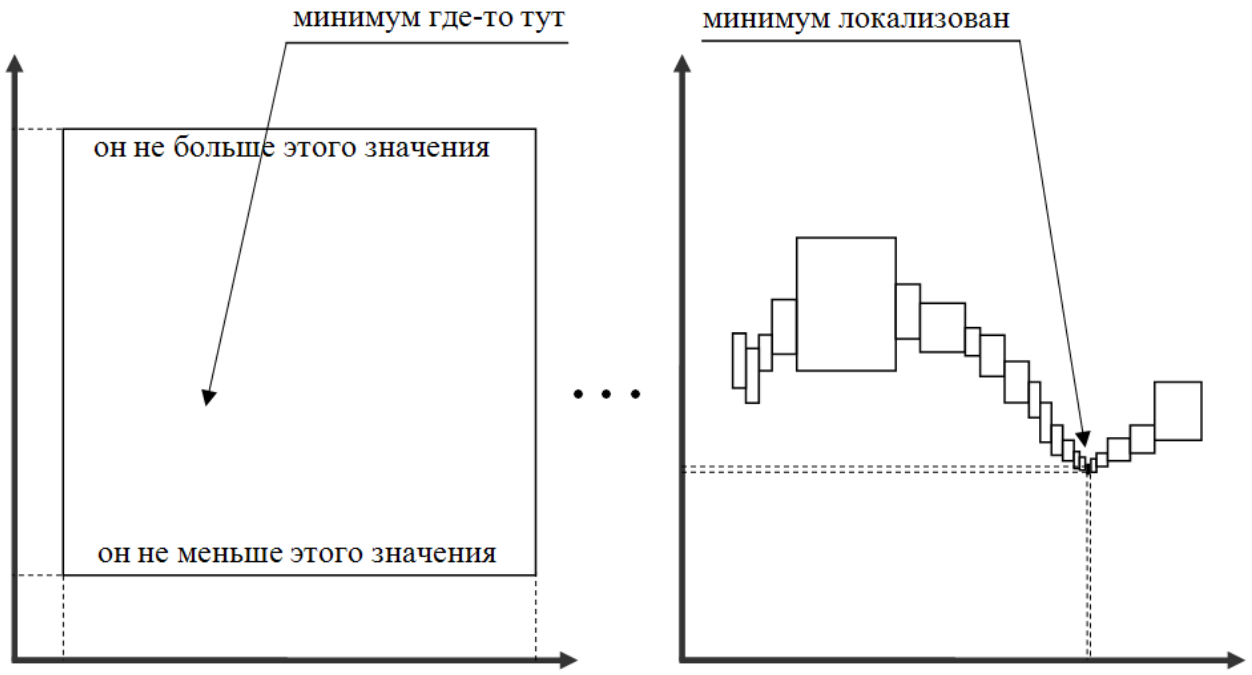


Рис. 2.4: Поиск оптимума последовательным дроблением области определения

Целесообразно внести в этот процесс последовательного дробления-оценивания некоторый порядок, и прежде всего мы обсудим возможную стратегию дробления, т. е. алгоритм, в соответствии с которым на той или иной итерации выбирается дробимый брус.

Исходя из того, что чем шире брус, на котором происходит оценивание, тем выше погрешность интервальной оценки, можно попробовать на каждой итерации находить самый широкий брус и дробить его. Ниже приведён псевдокод такого алгоритма, где `_boxContainer` это некий список всех брусов,

изначально содержащий всего один брус — всю интересующую нас область. Очевидная простота этого алгоритма, пожалуй, единственное его достоинство. Ведь нам, скорее всего, придется последовательно перебрать все брусы, сделав $(\mathrm{wid}\,(\boldsymbol{x})/\epsilon)$ итераций, где ϵ — нужная нам точность.

Листинг 4. Один из возможных алгоритмов последовательного улучшения оценки оптимума на языке C++

```
. . .

 _boxContainer.add(X);          // Занести в рабочий список исходный брус.
 do {
       Box workBox = _boxContainer.getWidest();
                                              // Извлечь самый широкий брус.

       if ( workBox.size() < min_size )       // Выход, если его размер
              break;                          //  меньше требуемой точности.

       _temporaryLine = algorithm->splitBox(workBox); // Раздробить
                                              //брус каким-либо способом.

       _boxContainer.merge(_temporaryLine); // Добавить в рабочий список
                                              //  новые подбрусы.

    }  while ( step_counter++ < max_step )  // Алгоритм заканчивается,
                                              //  если достигнуто
                                              //  максимальное количество
                                              //  итераций.

. . .
```

Более разумный выбор — дробить брус с наименьшей нижней границей, (с наибольшей верхней границей, если нас интересует максимум функции). На рис. 2.5 приведена блок-схема широко распространенного метода «адаптивного интервального дробления».

Алгоритм также оперирует понятием «рабочий список» — это список (реализуемый или как список или стек или куча), где хранятся все подбрусы, порождённые алгоритмом. Ведущий или наиболее перспективный брус — это брус, на котором в настоящий момент достигается наибольшая (наименьшая) оценка значения функции. Критерием остановки может быть максимальное

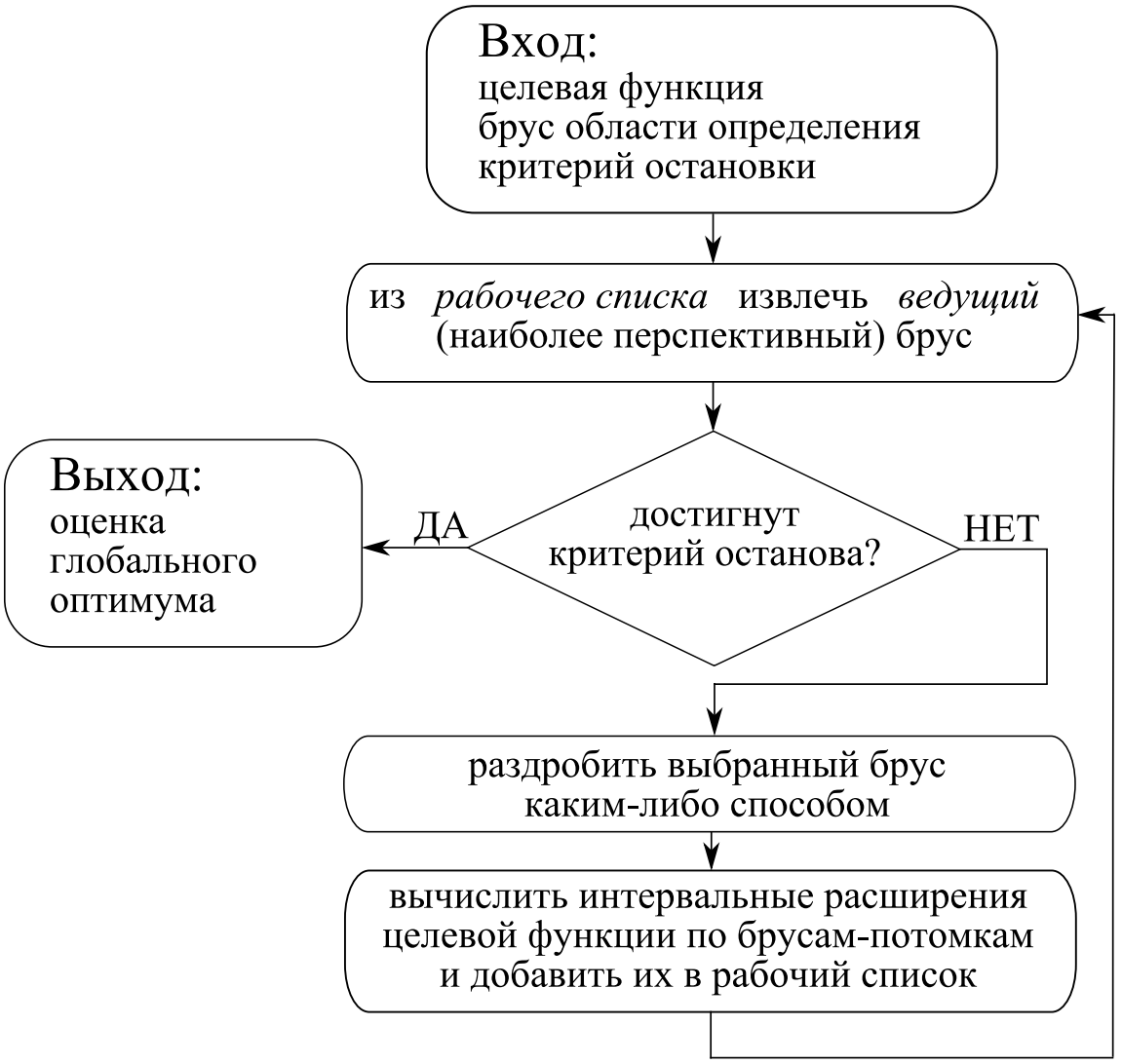


Рис. 2.5: Блок-схема оптимизационного алгоритма адаптивного интервального дробления.

количество итераций, достаточная ширина оценки оптимума и т. п.

Этот алгоритм, часто называют базовым интервальным алгоритмом глобальной оптимизации. К его достоинствам следует отнести очевидную простоту реализации и доказательность (гарантированность) результатов. Она следует из того ценного свойства интервального расширения функции, которое гарантирует, что на данной области определения ни одно значение функции не выйдет за вычисленные границы. Таким образом гарантируется, что найденное значение (значения) — это действительно глобальный оптимум, и ни один оптимум не пропущен.

Теперь поговорим о недостатках метода. Для гарантии того, что ни один оптимум не пропущен, алгоритм вынужден хранить все подбрусы, что при решении «достаточно сложной» задачи на «достаточно большой» области

определения рано или поздно может вылиться в проблему нехватки оперативной памяти и резкому замедлению скорости вычислений. Подробнее это освещается в §2.3.

Кроме простого хранения брусов, алгоритм вынужден выполнять определенные действия по поддержанию рабочего списка упорядоченным либо же по поиску наиболее перспективного бруса при неупорядоченном хранении (подробнее способы организации рабочего списка мы рассмотрим в §2.4). Кроме того, имея какие-то критерии отбраковки брусов или их частей, мы, возможно, будем способны сэкономить время, не дробя заведомо бесперспективные брусы.

2.3 Особенности современных машинных вычислений

Чтобы выполнить на компьютере какую-нибудь операцию, например, умножение $m = a \cdot b$ необходимо сначала загрузить значения переменных a и b из памяти, исполнить инструкцию умножения, и затем сохранить результат. Таким образом, время выполнения одной вычислительной операции можно оценить как их сумму:

$$t_\Sigma \leqslant t_{load} + t_{comput} + t_{store}$$

t_{comput} — время, которое потребуется процессору на то, чтобы перемножить два числа, когда они уже загружены; t_{load} — время на загрузку данных из памяти, t_{store} — время на сохранение результата. Приведенное выше выражение общего времени выполнения операции является оценкой сверху, и нестрогое неравенство использовано не случайно. Внутренняя логика современных процессоров и оптимизирующие компиляторы стараются избавляться от влияний подобных факторов, что позволяет в какой-то степени уменьшить общее время исполнения программы.

Распространено убеждение, что для более быстрых вычислений нужен более быстрый процессор. Это было действительно так на ранних этапах развития вычислительной техники. Затем, видимо, этот миф укоренился благодаря деятельности служб маркетинга компаний – производителей компьютеров. На самом деле, важны все этапы. Более того, на данный момент $t_{comput} \ll t_{load} + t_{store}$.

Скорость работы с памятью определяется двумя величинами. Первая — это пропускная способностью канала (в англоязычной литературе называемая bandwidth), показывающая, сколько данных, инструкций и т. п. можно доставить за единицу времени. И вторая — это скорость доставки (latency),

определяющая время доставки. Для лучшего понимания этих двух характеристик можно воспользоваться аналогией с нефтепроводом. Пример вполне нагляден: можно прокачивать через нефтепровод немалое количество нефти в минуту, но каждый конкретная частица нефти идёт до места назначения несколько дней.

Практически важный момент в том, что пропускную способность увеличивать легко. Можно поставить два процессора, брать из памяти за один раз в два раза больше данных, поставить два компьютера, в конце концов. Время доставки же гораздо дороже. Хуже того, его нельзя улучшить экстенсивно. Если операция занимает 100 тактов, то поставив два устройства, можно выполнять две операции за такт (тем самым увеличив пропускную способность), но время выполнения одной операции останется неизменным. Остаётся лишь улучшать технологию, уменьшать размеры элементов, увеличивать частоты, и так далее.

Опыт последних двадцати с лишним лет показывает, что время обработки растёт гораздо медленней, чем пропускная способность, и это особенно актуально для памяти.

	1980 VAX-11/750	2007 Персональный компьютер	улучшение с 1980-го года
Частота процессора (MHz)	6	3000	в 500 раз лучше
Объем памяти (RAM, Mb)	2	2000	в 1000 раз лучше
Пропускная способность памяти (bandwdth, Mb/s)	13	7000 (чтение) 2000 (запись)	в 540 раз лучше в 150 раз лучше
Скорость работы с памятью (latency, ns)	225	~ 70	в 3 раза лучше
Относительная скорость работы с памятью (latency, такты)	1.4	210	**в 150 раз хуже**

Из этой таблицы видно, что за последние десятилетия тактовая частота процессора существенно выросла, размер памяти вырос ещё быстрее, пропускная способность памяти — тоже весьма значительно, а вот скорость доступа со времен VAX стала всего в три раза лучше. А в пересчёте на такты

(последняя строка) — ухудшилось в 150 раз. Это означает, что прямое обращение к памяти в ситуации, когда к моменты вычислений данных не было в кэше (быстрой маленькой памяти, расположенной непосредственно на процессоре) стоит на порядки дороже даже самых «тяжелых» инструкций процессора.

В 80-х годах время доступа к памяти было вполне сравнимо, а то и меньше, чем время выполнения вычислительных инструкций. Для программиста существовали процессор, диск и память, которыми он мог оперировать непосредственно. Код выполнялся прозрачно и был предсказуем до такта.

Сейчас же всё изменилось. Доступ к памяти составляет сотни тактов. Производители борются с этим, в основном, за счёт улучшения механизма кэширования. Логика и иерархия кэша всё усложняются и являются «ноу-хау» различных производителей процессоров. Кроме этого в процессорах обязательно реализуются механизмы внеочередного выполнения команд (out of order execution), что позволяет уменьшить время простоя вычислительных устройств. Так, если следующая команда использует результат предыдущей, то первая не может начать выполняться до завершения последней. В это время можно попытаться исполнить независимые цепочки команд. Реализуется мощный механизм предсказания переходов (branch prediction) и аппаратная упреждающая выборка данных (hardware prefetch), для того, чтобы начать подгружать данные как можно раньше, ещё задолго до того, как они понадобятся.

На процессоре Itanium2 фирмы Intel кэш занимает более половины площади процессора. На последних процессорах Core 2 Duo примерно столько же. Ещё чуть больше десяти процентов — логика «out of order», «branch prediction» и тому подобных механизмов. Остаются считанные проценты на собственно арифметико-логическое устройство процессора (ALU), которое что-то реально считает.

Не будет слишком большим преувеличением сказать, что сегодня компьютер из электронно-вычислительной машины уже давно превратился в своеобразный микроконтроллер, управляющий вводом-выводом. Современный процессор — это не вычислитель, а гигантский аппаратный эмулятор x86-инструкций. Вся это нужно для того, чтобы спрятать от программиста то самое пресловутое время исполнения или доставки — latency. Для того, чтобы можно было продолжать программировать в стиле 80-х годов — когда есть только процессор и память, причем обращаться к памяти можно сколько угодно недорого. Чтобы продолжать запускать старый код всё лучше, чтобы новый можно было писать всё так же, как и ранее.

Получается, что современный прогресс вычислительной техники направлен, в частности, на то, что бы скрыть падение скорости в 150 раз, причём сделать это незаметно для программиста, т. е. не изменяя его структур данных и так, чтобы он не заметил изменения порядка выполнения инструкций. Разумеется, это совершенно непродуктивная деятельность. К сожалению, многие программисты по-прежнему не задумываются об этом.

Но вернемся к задаче интервальной глобальной оптимизации. Всё вышесказанное имеет к ней непосредственное отношение.

Одно из важных свойств современных интервальных методов — гарантированность (доказательность) результатов. В приложении к задачи глобальной оптимизации это выливается в необходимость хранить все подбрусы, для которых не установлено наверняка, что глобальный оптимум на них не достигается (см. §2.5). Оставим пока в стороне вопрос о внутренней реализации рабочего списка (а значит, и времени работы таких операций как сортировка, поиск или добавление новых элементов). Пока рассмотрим лишь весьма модельный пример — обращение к какому-нибудь (произвольному) брусу из рабочего списка.

На рис. 2.6 и 2.7 приведены результаты реальных измерений пропускной способности шины памяти и времени ожидания данных. Замеры проводились на не самой лучшей, но весьма неплохой (на момент написания настоящего текста) системе: Intel Pentium M 1.8 GHz, 266 MHz DDR2 SDRAM. Из графиков хорошо видно, что если размер данных, которыми мы оперируем, превысит 32 (тридцать два) килобайта, производительность программы упадет примерно в два раза. При превышении двухмегабайтной границы (на рассматриваемой системе кэш второго уровня — два мегабайта) падение производительности превысит много более десяти раз.

Оценим, насколько много или мало для нас 32 Kb.

Для большинства вычислительных систем одна переменная двойной точности занимает в памяти 8 байт. Следовательно, мы можем позволить себе хранить в этих 32 Kb всего $\frac{32 \cdot 1024}{8} = 4096$ чисел двойной точности. Это величина кажется небольшой. В пространстве размерности n такое количество чисел позволит задать $4096/n$ точек. Многомерный интервал в n-мерном пространстве представляет собой прямоугольный параллелепипед (брус) с 2^n вершинами, для описания которого необходимо хранить две n-мерных точки — нижнюю и верхнюю границы бруса. Таким образом, в трехмерном пространстве мы можем оперировать с $\frac{4096}{3 \cdot 2} = 682$ брусами. Это оценка сверху, так как мы не учитывали прочие переменные, а также неявно исходили из предположения, что отсутствуют дополнительные расходы памяти на хране-

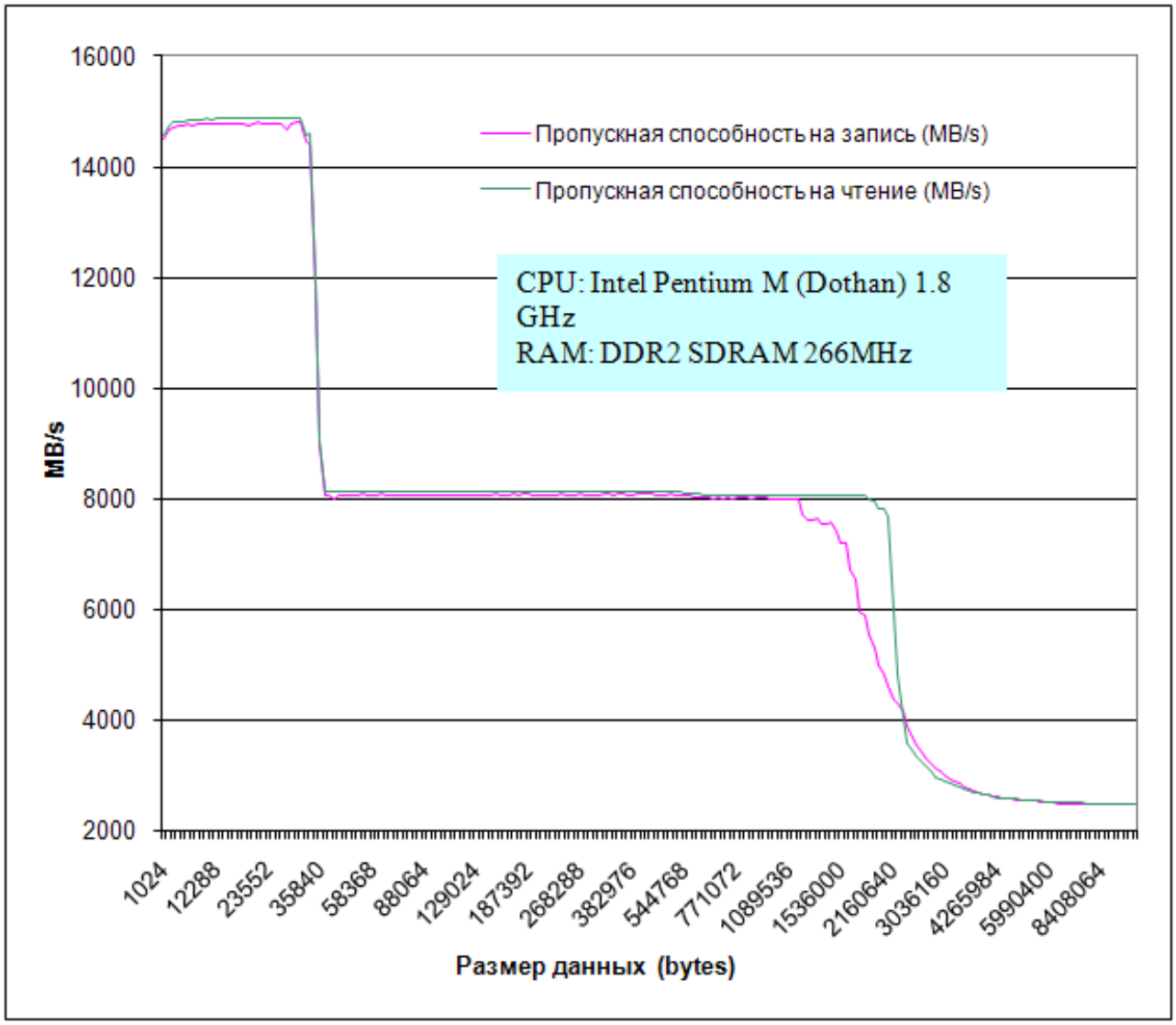


Рис. 2.6: Время ожидания данных

ние брусов, как это имеет место, например, при использовании контейнера «`list`» и вообще больше никакой работы с памятью не происходит.

Необходимо сделать весьма важное замечание. Полученную только что оценку следует воспринимать не как ограничение на размер всего рабочего списка, а как объём данных, с которыми мы можем работать быстро. То есть, если размер всего рабочего списка — тысячи брусов, но мы постоянно работаем лишь с первыми двумя, все нужные нам данные всегда будут в кэше и операции будут исполняться быстро. Это и, в частности, так называемый «приём Панкова», будут обсуждаться в §2.4.

Хранение и управление списком брусов может потребовать дополнительных расходов. Так как планируются частые вставки и удаления элементов последовательности, то лучше предпочесть контейнер, который выполняет

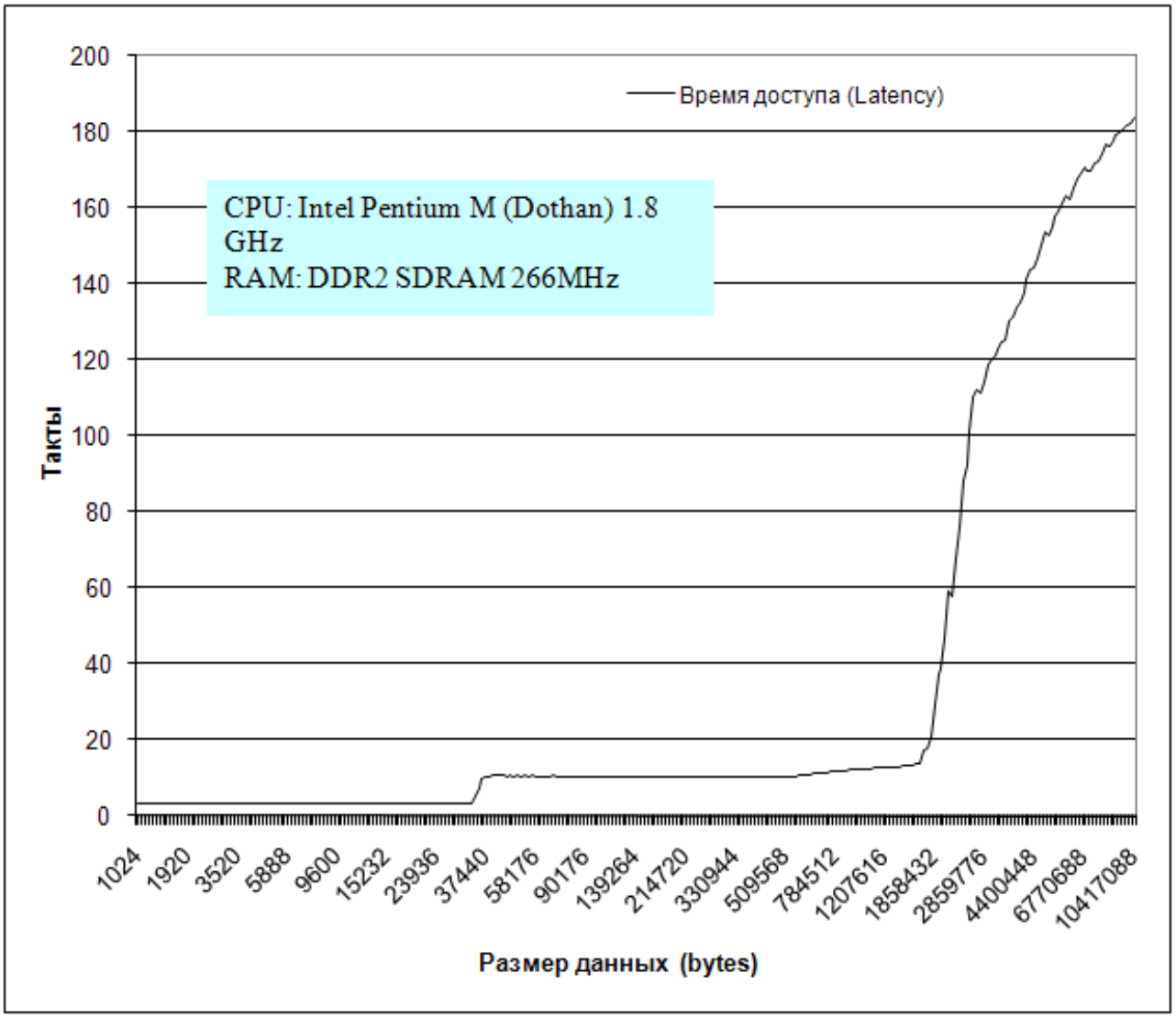


Рис. 2.7: Пропускная способность шины памяти

такие действия за постоянное время, то есть такой, для которого время на выполнение вставки или удаления не увеличивается с ростом числа элементов (N) в последовательности. Однако, такой контейнер, скорее всего, не поможет вам обнаружить нужный элемент последовательности за постоянное время.

Но и в этом случае можно попробовать найти определенный компромисс. Для контейнера можно обеспечить вставку, удаление и поиск за время, пропорциональное $\log N$. Для такого контейнера время работы операций, конечно, будет зависеть от размера контролируемой последовательности, но увеличиваться оно будет гораздо медленнее, чем растет N. Такая временная сложность может оказаться вполне приемлемой. Однако и за это приходится платить.

Чтобы обеспечить логарифмическую зависимость, контейнер должен представлять собой управляемую последовательность в виде древовидной структуры. При таком способе хранения данных в каждом элементе последовательности наряду с самим объектом содержаться ещё три дополнительных указателя — для обозначения родительского элемента и двух дочерних. Некомпактность представления данных дополнительно сказывается на производительности. В случаях, когда хранение значения элемента само по себе требует значительного объема памяти, такие дополнительные издержки могут быть и не столь критичными. В нашем же случае размер памяти, выделяемой для поддержания структуры (накладные расходы), практически, в полтора раза превышает полезный.

В этом случае наилучшим выбором можно считать наиболее простые контейнеры. Такой контейнер требует меньшего количества указателей на элемент (а может быть и не требует вовсе), однако за это придётся расплачиваться увеличением временно̀й сложности некоторых операций над контролируемой последовательностью.

Шаблонные классы-контейнеры в STL (Standard Template Library) [113] обычно представлены следующим набором:

`vector` — массив элементов, расположенных в памяти подряд.

`list` — двунаправленный связанный список.

`deque` — двусторонняя очередь.

`set` — черно-белое дерево, элементы упорядочены с помощью некого предиката, который применяется к парам элементов, никакая пара элементов не имеет эквивалентного упорядочивания. Его разновидность, `multiset`, допускает элементы с эквивалентным упорядочиванием.

`map` — набор пар (ключ, значение), упорядоченный по некоторому предикату, примененному к паре ключей. Структура multimap допускает пары ключей, имеющих равное упорядочивание.

В следующей таблице приведены временна́я и пространственная сложность контейнеров.

	`vector`	`deque`	`list`	`set/map`
Вставка, удаление	N	N	const	$\log N$
Добавление в начало	N^*	const	const	$\log N^*$
Поиск	N^*	N^*	N^*	$\log N$
Доступ к n-ному элементу — x[n]	const	const	N^*	N^*
Кол-во дополнительно хранимых указателей (на элемент)	0	1	2	3
* — операции не поддерживаются напрямую.				

2.4 Структура рабочего списка

При практической реализации интервальных методов глобальной оптимизации весьма важен вопрос о конкретной структуре рабочего списка. Мур [106, 107], Скелбоу [124], Рачек [116], Кирфотт [97] в своих работах поддерживают рабочий список упорядоченным по возрастанию нижней границы интервальной оценки. Хансен [94] же реализует его без какого-либо упорядочения, в виде так называемой кучи. В последнем случае экономится время при добавлении элементов в рабочий список, но зато последующий поиск ведущего бруса в неупорядоченном списке наоборот отнимает всё возрастающее время.

Наши исследования показывают, что с точки зрения производительности программы выгоднее поддерживать структуру упорядоченной. Отметим, что это возможно не во всех реализациях, так как некоторые системы программирования (к примеру, MATLAB и ему подобные, такие как Scilab, Octave и др.) имея в своём арсенале динамические структуры данных, не поддерживают, тем не менее, ссылки (указатели).

Некоторое ускорение обработки списка $\mathcal{L}$ может быть достигнуто с помощью следующего приёма, предложенного П.С. Панковым [38]. В его основе — задание и корректировка по текущим нижней и верхней оценкам глобального минимума, $\Omega(\boldsymbol{Q})$ и ω, соответственно, вспомогательной «пороговой константы» γ, такой что

$$\Omega(\boldsymbol{Q}) < \gamma < \omega,$$

и «подсписка активных записей»

$$\mathcal{L}_\gamma = \left\{ \ (\boldsymbol{P}, \Omega(\boldsymbol{P})) \in \mathcal{L} \mid \Omega(\boldsymbol{P}) < \gamma \ \right\} \subseteq \mathcal{L}.$$

В случае неупорядоченного рабочего списка (кучи записей) ясно, что именно в $\mathcal{L}_\gamma$ (при $\mathcal{L}_\gamma \neq \varnothing$) находится ведущая запись всего $\mathcal{L}$, и потому при её поиске нам достаточно, сэкономив машинное время, ограничиться лишь просмотром $\mathcal{L}_\gamma$. Если же мы придерживаемся варианта упорядоченного рабочего списка $\mathcal{L}$, то по аналогичным причинам эту упорядоченность достаточно поддерживать только в $\mathcal{L}_\gamma$, организовав дополнение $\mathcal{L} \setminus \mathcal{L}_\gamma$ в виде кучи. В процессе работы алгоритма подмножество $\mathcal{L}_\gamma$ не возрастает, и если на некотором шаге оно сделается пустым, то тогда же перевычисляется пороговая константа γ, и из $\mathcal{L}$ заново выделяется $\mathcal{L}_\gamma$.

Отметим также, что, будучи реализованным, приём П.С. Панкова не позволяет производить «чистку» всего списка $\mathcal{L}$ от бесперспективных записей в промежутках между перевычислениями «пороговой константы» γ, так что определённый выигрыш в быстродействии достигается им ценой дополнительной оперативной памяти.

2.5 Критерии отбраковки

Помимо уточнения интервальной оценки на перспективных подбрусах, полезно уметь выделять бесперспективные подбрусы, то есть такие, для которых может быть достоверно показано, что глобальный оптимум на них недостижим.

Отбраковка по значению

Допустим, в результате работы алгоритма оптимизации у нас получилась конфигурация, подобная показанной на рис. 2.8.

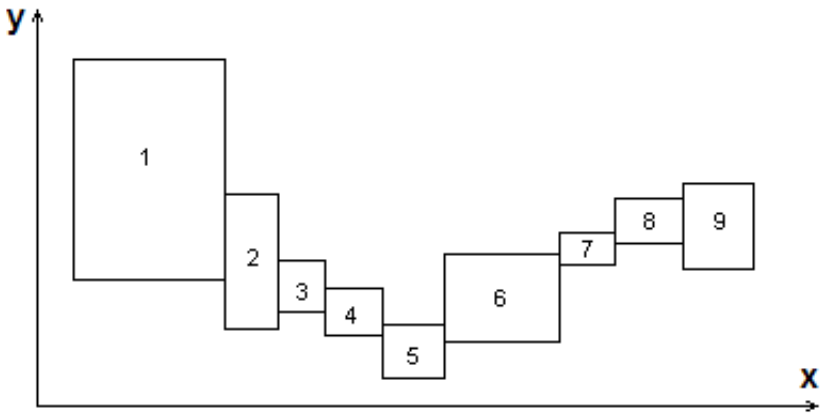


Рис. 2.8: Пример конфигурации брусов в процессе оптимизации

Можем ли мы исходя из этой информации указать, в каких брусах точно не может быть, скажем, глобального минимума? Ответ на этот вопрос положителен, так как интервальное расширение функции *гарантирует*, что значения функции на данном интервале не выйдут за обозначенные границы. Таким образом, брусы с номерами 1, 3, 7, 8, 9 гарантированно не содержат глобальный минимум, поскольку нижняя грань их значений находится выше верхней грани одного из брусов. А это значит, что нижняя оценка значений функции на этих брусах больше, чем верхняя оценка значений функции на брусе номер 5. То есть, какие бы значения не принимала целевая функция внутри бруса номер 5, она всё равно будет меньше любого своего значения на брусах 7, 8, 9, 1, 3.

Использование в качестве порогового значения верхней границы ведущего бруса (то есть бруса с наименьшей нижней границей, текущего кандидата на оптимум), является очевидным, но далеко не всегда лучшим выбором. Преимущество такого выбора состоит лишь в отсутствии каких-либо дополнительных вычислений: верхняя граница уже известна. Дело в том, что, как уже не раз говорилось, оценка верхней границы интервального расширения функции может быть избыточной. Повысить эффективность метода можно, вычислив точное значение целевой функции, скажем, в середине ведущего бруса.

Можно попытаться несколько улучшить этот метод. Первое предложение, которое было опробовано — использовать не среднюю точку, а случайно выбранную. На нашем тестовом примере этому методу удалось отбраковать 4812 бруса, что оказалось на десять брусов больше, чем при использовании средней точки. Для сравнения, использование верхней границы ведущего бруса позволило отбросить всего лишь 2291 брус. Использование двух случайных точек не улучшило полученный результат, и аналогичная ситуация наблюдалась для четырех, восьми, двадцати и пятисот случайных точек. Таким образом, дальнейшее улучшение, по-видимому, является невозможным, и было решено использовать шеститочечный тест. В нём выбиралось минимальное (максимальное) значение из четырех точек «в углах» бруса и двух внутри. Такой выбор позволил отсеять 4834 бруса. Но, как следует из экспериментов со случайными точками, ценность внутренних точек невысока. Это подтверждается и экспериментом: количество отбракованных брусов при четырёхточечном тесте такое же, как при использовании шести точек. Использование же классических методов поиска локального оптимума, таких как метод спуска, по нашему мнению, на небольших брусах неоправданно. Реализация их непроста, а дополнительного преимущества они не дают. На больших же брусах их

применение может значительно улучшить значение обсуждаемого критерия отбраковки.

Оговоримся, что не стоит ждать ощутимого выигрыша в производительности всего метода глобальной оптимизации от уменьшения количества вычисляемых точек, так как эти расчеты занимают весьма малую часть от общего времени работы программы.

И ещё одно замечание. Важно не просто сравнивать значение целевой функции в выбранных точках между собой, но ещё и хранить информацию о минимальном (максимальном) найденным таким образом значении функции r и сравнивать также и с ним.

Остался последний вопрос — когда применять этот критерий. На наш взгляд, имеет смысл это делать всегда, то есть на каждой итерации алгоритма оптимизации применять этот метод для всех полученных подбрусов. В случае, если при этом произошло уточнение рекорда r, придется пересматривать весь рабочий список.

Можно также слегка изменить этот подход. При изменении значения рекорда r, мы продолжаем обычные итерации алгоритма, не пересматривая весь список до тех пор, пока мы не решим по каким-либо соображениям, что размер рабочего списка слишком велик и время, потраченное на его чистку окупится за счет уменьшения времени последующих операций с ним (см. §2.3).

Тест на монотонность

В предыдущей главе нам удалось отсечь часть брусов, опираясь на значение интервального расширения функции $\boldsymbol{f}(\boldsymbol{x})$. Попробуем улучшить наш результат, используя информацию об интервальной оценке производной $\boldsymbol{f}'(\boldsymbol{x})$, которую мы можем легко получить, используя описанную в §1.2 технику автоматического дифференцирования.

В точке экстремума производная по каждой переменной должна обращаться в нуль. Таким образом, если на каком-либо подбрусе интервальное расширение первой производной целевой функции по какой-либо координате не содержит нуля, то такой брус гарантированно не содержит точек экстремума, а следовательно, может быть безболезненно удален из рассмотрения. Двумерный случай проиллюстрирован на листинге 5. В n-мерном случае тре-

буется анализировать градиент функции, равный

$$\bigtriangledown f(x) = f'(x)^T = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}.$$

Единственная, но важная оговорка состоит в том, что тест на монотонность нельзя применять к граничным брусам, так как оптимум может достигаться на границе области определения, и при этом совсем не обязательно, чтобы первая производная целевой функции занулялась.

Листинг 5. Схема алгоритма отбраковки

```
if ( workBox->isBorderOne() )
else if ( ! workBox->dFx()->isItIn(0) || ! workBox->dFy()->isItIn(0) )
    _boxQueue.remove(workBox);
```

Тест на выпуклость-вогнутость

Для выделения подбрусов, наверняка не содержащих оптимума, мы уже использовали информацию об интервальном расширении самой функции и её первой производной. Как известно, вторая производная функции содержит информацию о форме её графика — выпуклости или вогнутости функции вблизи этой точки, а интервальное расширение второй производной позволяет узнать вид функции на целом брусе. Соответственно, мы можем отсеивать брусы «не той» кривизны, то есть вогнутые, если нас интересует глобальный минимум и выпуклые, если наша цель — глобальный максимум.

Другим общеизвестным фактом является то, что достаточным условием существования экстремума функции f в стационарной точке x (точке зануления градиента) является знакоопределённость гессиана [31].

Исторически термин «гессиан» был введён Сильвестером для обозначения определителя матрицы Гессе, то есть матрицы, образованной вторыми частными производными функции. Сейчас этим термином часто обозначают

просто матрицу вторых производных

$$H(f) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}.$$

Тогда,

— если гессиан положительно определён и неособен, то x — точка локального минимума функции f;

— если гессиан отрицательно определён и неособен, то x — точка локального максимума функции f;

— если гессиан принимает как положительные, так и отрицательные значения, то x — седловая точка функции f.

Таким образом, при поиске глобального минимума (максимума) могут быть исключены брусы, на которых гессиан не является неположительно (неотрицательно) определенным.

Интервальный метод Ньютона

Метод Ньютона — это итерационный численный метод решения нелинейных уравнений, в котором поиск решения осуществляется путём построения последовательных приближений. Метод обладает квадратичной сходимостью. Его интервальная версия – *интервальный метод Ньютона* — позволяет локализовать корни, в том числе нелинейных уравнений. В задачах глобальной оптимизации это может быть использовано при использовании методов распространения ограничений, а также при прямом поиске нулей первой производной целевой функции. В точке экстремума частные производные должны обращаться в нуль. Таким образом, если на каком-либо негранично подбрусе интервальное расширение первой производной целевой функции по какой-либо переменной не содержит нуля, то он гарантированно не содержит точек экстремума, и, следовательно, может быть удалён из рассмотрения.

Рассмотрим случай одномерного уравнения $f(x) = 0$. Если разложить функцию f по формуле Тейлора в окрестности некоторой точки x_0

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + O(x^2) = 0,$$

то

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) = 0.$$

При решении этого уравнения получаем новое приближение $x = \frac{x_0 - f(x_0)}{f'(x_0)}$, если $f'(x_0) \neq 0$. Таким образом можно реализовать следующий итерационный алгоритм:

0. Выбираем $x^{(0)}$;

1. Итерируем:

$$x^{(k+1)} = x^{(k)} - \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)}, \qquad k = 0, 1, 2, \ldots .$$

Он называется (одномерным) методом Ньютона. Сходимость метода Ньютона зависит от выбора x_0, так что при неудачном начальном значении данный метод может, вообще говоря, и не сойтись. Перейдём теперь к интервальному случаю и также начнём с одномерной версии.

Пусть $\boldsymbol{x}$ — интервал на нём рассматриваем уравнение $f(x) = 0$, где f — непрерывно дифференцируемая функция на $\boldsymbol{x}$. Если $\tilde{x}$ — некоторая точка из интервала $\boldsymbol{x}$, то в силу теоремы Лагранжа о среднем получаем:

$$f(x) - f(\tilde{x}) = f'(\xi)(x - \tilde{x}), \tag{2.4}$$

где $\xi \in \boldsymbol{x}$. Если же при $x = x^*$ справедливо $f(x^*) = 0$, т. е. x^* — решение искомого уравнения, то путём простых преобразований из (2.4) получаем

$$x = \tilde{x} - \frac{f(\tilde{x})}{f'(\xi)}. \tag{2.5}$$

Пусть $\boldsymbol{f'}$ — интервальное расширение для f'. Тогда, интервализуя по $\xi \in \boldsymbol{x}$, получим из (2.5) включение

$$x \in \tilde{x} - \frac{f(\tilde{x})}{\boldsymbol{f'}(\boldsymbol{x})}$$

при условии, что данное выражение определено.

Отображение $\mathbb{R} \times \mathbb{IR} \to \mathbb{IR}$, определяемое как $\mathcal{N}(\tilde{x}, \boldsymbol{x}) = \tilde{x} - \frac{f(\tilde{x})}{\boldsymbol{f'}(\boldsymbol{x})}$ для заданной функции $f : \mathbb{R} \to \mathbb{R}$ с интервальным расширением её первой производной $\boldsymbol{f'}$, называется (одномерным) *интервальным оператором Ньютона.*

Получается, что если заданы $f, \boldsymbol{x}$, то $\mathcal{N}(\tilde{x}, \boldsymbol{x})$ определяет новый интервал, содержащий решение x^* уравнения $f(x) = 0$. Следовательно, более точные границы для x^* можно взять в виде $\boldsymbol{x} \cap \mathcal{N}(\tilde{x}, \boldsymbol{x})$. Итерационный процесс

$$\boldsymbol{x}^{(k+1)} = x^{(k)} \cap \mathcal{N}(\tilde{x}, \boldsymbol{x}^{(k)}), \qquad k = 0, 1, 2, \ldots ,$$

является простейшим одномерным интервальным методом Ньютона. Часто в качестве $\tilde{x}$ используют середину интервала $\boldsymbol{x}$.

Работая с этим методом, можно получить интервал внешней оценки производной, содержащий нуль. В этом случае в операторе Ньютона возникает деление на нуль, и мы получаем интервал от минус бесконечности до плюс бесконечности. Частично избежать этого помогает пересечение нового интервала с предыдущим.

В реальных задачах очень популярен многомерный метод Ньютона, который применяется к системам уравнений

$$f(x) = 0, \quad \text{где } f(x) = \big(f_1(x), f_2(x), \dots, f_n(x)\big)^\top.$$

Аналогично одномерному случаю после линеаризации относительно некоторой точки x_0 из области определения функции получаем следующее приближённое уравнение

$$f'(x_0)(x - x_0) = -f(x_0),$$

где $f'(x_0)$ — матрица Якоби. Если она неособенна, то система линейных уравнений разрешима и путём простых преобразований получим новое приближение как

$$x = x_0 - \big(f'(x_0)\big)^{-1} f(x_0).$$

Итерационный процесс в многомерном случае будет выглядеть следующим образом:

$$x^{(k+1)} = x^{(k)} - \big(f'(x^k)\big)^{-1} f(x^k), \qquad k = 0, 1, 2, \dots .$$

Теперь мы можем построить интервальную версию многомерного метода Ньютона. Пусть $\boldsymbol{x} \in \mathbb{IR}^n$ — многомерный интервал, на котором ищем решение, и пусть $x, \tilde{x} \in \boldsymbol{x}$. Аналогично одномерному интервальному случаю получим для многомерной функции $f(x)$, определённой на интервале $\boldsymbol{x}$

$$f(x) - f(\tilde{x}) = f'(\xi)(x - \tilde{x}), \qquad \text{где } \xi \in (x, \tilde{x}) \text{ в } \mathbb{R}^n$$

Возьмём $x = x^*$, где x^* — решение системы, так что $f(x^*) = 0$, и пусть известно интервальное расширение $\boldsymbol{f'}(\boldsymbol{x})$ для $f'(x)$ на интервале $\boldsymbol{x}$. Тогда получим:

$$0 \in f(\tilde{x}) + \boldsymbol{f'}(\boldsymbol{x})(x^* - \tilde{x}).$$

Воспользуемся известным фактом из теории интервальных линейных систем уравнений, называемым характеризацией Бека (см., к примеру, [72]). Пусть $\boldsymbol{A}x = \boldsymbol{b}$ — интервальная система линейных алгебраических уравнений (ИСЛАУ), и $\Xi(\boldsymbol{A}, \boldsymbol{b})$ — её множество решений. Тогда принадлежность $x \in \Xi(\boldsymbol{A}, \boldsymbol{b})$ равносильна $\boldsymbol{A}x \cap \boldsymbol{b} \neq \varnothing$ или же $0 \in \boldsymbol{A}x - \boldsymbol{b}$.

Из характеризации Бека следует, что $x^* - \tilde{x}$ принадлежит множеству решений интервальной системы $\boldsymbol{f}'(\boldsymbol{x}) \cdot y = -F(\tilde{x})$. Это справедливо тогда и только тогда, когда

$$x^* \in \tilde{x} + \Big(\text{ множество решений ИСЛАУ } \boldsymbol{f}'(\boldsymbol{x}) \cdot y = -F(\tilde{x}) \Big).$$

Следовательно, если обозначить посредством $Encl$ внешнюю интервальную оценку множества решений ИСЛАУ, полученную каким-либо методом, то

$$x^* \in \tilde{x} + Encl\Big(\Xi\big(\boldsymbol{f}'(\boldsymbol{x}), -f(\tilde{x})\big)\Big).$$

Оператор Ньютона для многомерной функции $f(x)$ и интервального расширения её первой производной $\boldsymbol{f}'(\boldsymbol{x})$ определяется следующим образом:

$$\mathcal{N}(\tilde{x}, \boldsymbol{x}) := \tilde{x} - Encl\Big(\Xi\big(\boldsymbol{f}'(\boldsymbol{x}), f(\tilde{x})\big)\Big).$$

Здесь, как и в одномерном варианте интервального метода Ньютона, $\mathcal{N}(\tilde{x}, \boldsymbol{x})$ определяет новый интервал, содержащий решение x^* уравнения $F(x) = 0$. Точно так же более точные границы для x^* можно взять в виде $\boldsymbol{x} \cap \mathcal{N}(\tilde{x}, \boldsymbol{x})$. Для дальнейшего уточнения решения можно запустить итерирование. В целом алгоритм многомерного интервального метода Ньютона таков:

Полагаем $\boldsymbol{x}^{(0)} = \boldsymbol{x}$.

Для $k = 0, 1, 2, \ldots$. вычисляем $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} \cap \mathcal{N}(\tilde{x}, \boldsymbol{x}^{(k)})$.

В качестве $\tilde{x}$ можно взять вектор скалярную матрицу, состоящую из середин интервалов компонент бруса $\boldsymbol{x}$.

Многомерная версия интервального метода Ньютона имеет свои недостатки. Так, по ходу работы с этим методом постоянно нужно решать задачу внешнего оценивания множества решений интервальных систем алгебраических уравнений. Кроме того, интервальная линейная система, которую необходимо решать при реализации интервального метода Ньютона, может иметь особенную матрицу, и тогда её множество решений будет неограничено.

Существуют различные модификации исходной схемы интервального метода Ньютона, преодолевающие эти недостатки. Таковы метод Кравчика, метод Хансена-Сенгупты и другие. Описанные методы и их модификации могут быть использованы при поиске глобального оптимума функций для нахождения нулей градиента. Это, как было указано выше, можно использовать для дополнительного повышения эффективности решения задачи глобальной оптимизации [37].

Введение критериев отбраковки существенно улучшает алгоритм интервальной оптимизации. Как можно добиться дальнейших улучшений?

2.6 Способы дробления брусов

Задумаемся теперь над тем, как дробить очередной брус. Очевидно, что это можно сделать множеством способов. С одной стороны, выбранный метод должен обеспечивать стремление размеров ведущих брусов к нулю в процессе работы алгоритма, а, с другой стороны, не приводить к излишним трудозатратам, например, вызванным чрезмерным «раздутием» рабочего списка.

Простейшую процедуру дробления брусов описал Р.Е. Мур в своей книге «Interval Analysis» в 1966 году [106]. Он установил также, что точность естественного интервального расширения имеет первый порядок в зависимости от ширины брусов, а следовательно, для её улучшения надо делать интервалы мельче. На основании этого им был предложен итеративный метод поиска глобального оптимума. Суть предложенного метода сводилась к тому, чтобы на каждой итерации дробить все брусы по всем измерениям. При этом автор признавал существенную трудоёмкость такого метода и оговаривался, что метод представляет скорее теоретическую ценность.

Лишь через 8 лет датчанин С. Скелбоу предложил существенное улучшение приёма Мура — удачную вычислительную схему, подстраивающуюся под задачу, и основанную на заимствованном из комбинаторной оптимизации «методе ветвей и границ». Новая вычислительная схема позволяла дробить лишь один, наиболее перспективный на данном этапе брус, существенно экономя трудозатраты.

К сожалению, С. Скелбоу рассмотрел в своей работе [124] лишь одномерный случай. Далее Р.Е. Мур в ряде статей и книге [107] обобщил этот подход на многомерный случай и предложил дробить брус циклически каждый раз по следующей координате. Но оказалось, что при таком подходе ширина ведущего бруса может не стремиться к нулю, и, следовательно, алгоритм может не сходиться. На это обратили внимание Рачек и Рокне и предложили более совершенный способ измельчения брусов. Более того, они доказали [116], что при таком дроблении алгоритм глобальной оптимизации действительно сходится. Предложенный ими метод заключается в дроблении бруса всегда по самой широкой координате.

Дробление бóльшей стороны

За одну итерацию алгоритма выполняется одно деление, при этом делится наиболее широкая сторона бруса. Не делая никаких предположений о конфигурации функции, алгоритм последовательно уточняет интервальное рас-

ширение, деля наибольшую и, вероятно, дающую наименее точную оценку сторону.

Например, брус, показаный на рис. 2.9 а будет поделен таким (рис. 2.9 б) образом.

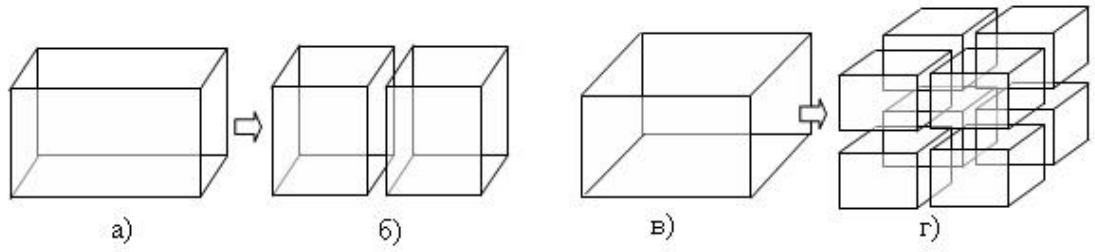


Рис. 2.9: Различные способы дробления брусов

Дробление стороны, на которой максимизируется оценка изменения функции

В 1992 году С.П. Шарым (в применении к методам дробления параметров для решения интервальных линейных систем) и затем Д. Ратцем было предложено дробить ту компоненту ведущего бруса, на которой достигается максимум произведения ширины на интервальную оценку частной производной. В силу известной из математического анализа теоремы Лагранжа о среднем значении нетрудно понять, что мы тем самым дробим ту компоненту бруса, по которой ожидается наибольшее изменение целевой функций. Этот метод дробления хорошо зарекомендовал себя на практике, и, кроме того, при некоторых естественных допущениях для него была показана теоретическая оптимальность [119].

Дробление всех сторон

Брус дробится сразу по всем координатам. К примеру, для функции от двух переменных за одну итерацию алгоритма брус будет разделён сразу на четыре части. Иллюстрация подобного дробления по всем сторонам представлена на рис. 2.9 в) и г).

Достоинство этого способа дробления в том, что мы за одну итерацию существенно уточняем интервальное расширение функции на подозрительном брусе. К сожалению, зачастую это может привести к чрезмерному усложнению одного шага алгоритма. Возможна нерациональная трата времени на совершение лишних дроблений, дополнительных оцениваний целевой функции

на полученных подобластях и применение процедур отбраковки. Наконец, при этом рабочий список может «замусориваться» лишними подбрусами, которые не уточняют значение глобального экстремума и которых можно было бы избежать.

Правда, на более узких брусах процедуры отбраковки работают зачастую эффективнее. Что же касается собственно точности интервальных оценок, то мы сравнили, как изменялась точность интервальной оценки оптимума при использовании этого метода дробления по сравнению с дроблением только лишь большей стороны (см. рис. 2.10 и 2.11). Разница, на наш взгляд, непринципиальная.

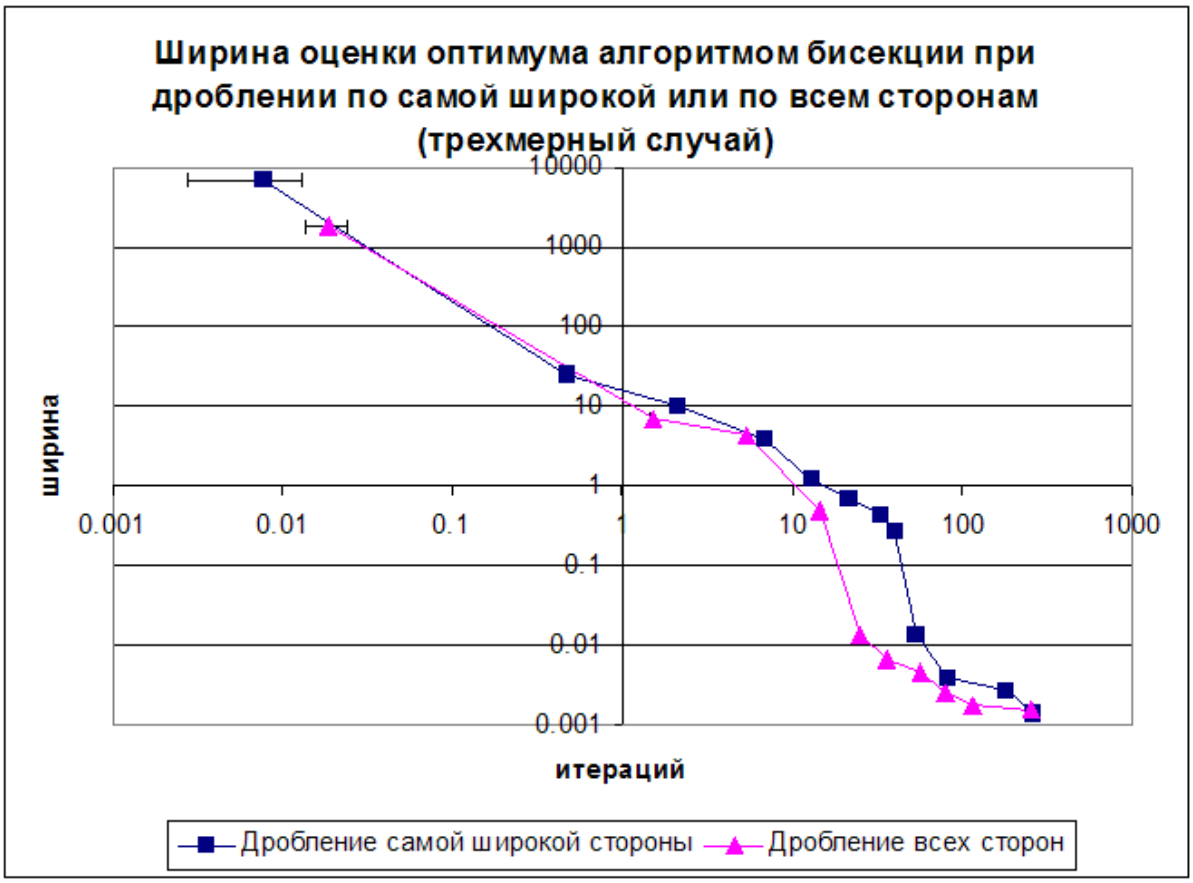


Рис. 2.10: Точность интервальной оценки в зависимости от способа дробления брусов

Эвристическое дробление

Этим метод похож на алгоритм деления большей стороны. Суть метода в том, что сторона для очередного дробления выбирается исходя из результативности предыдущих дроблений. Каждый раз, производя деление очередного бруса, алгоритм анализирует, насколько улучшилась интервальная оценка

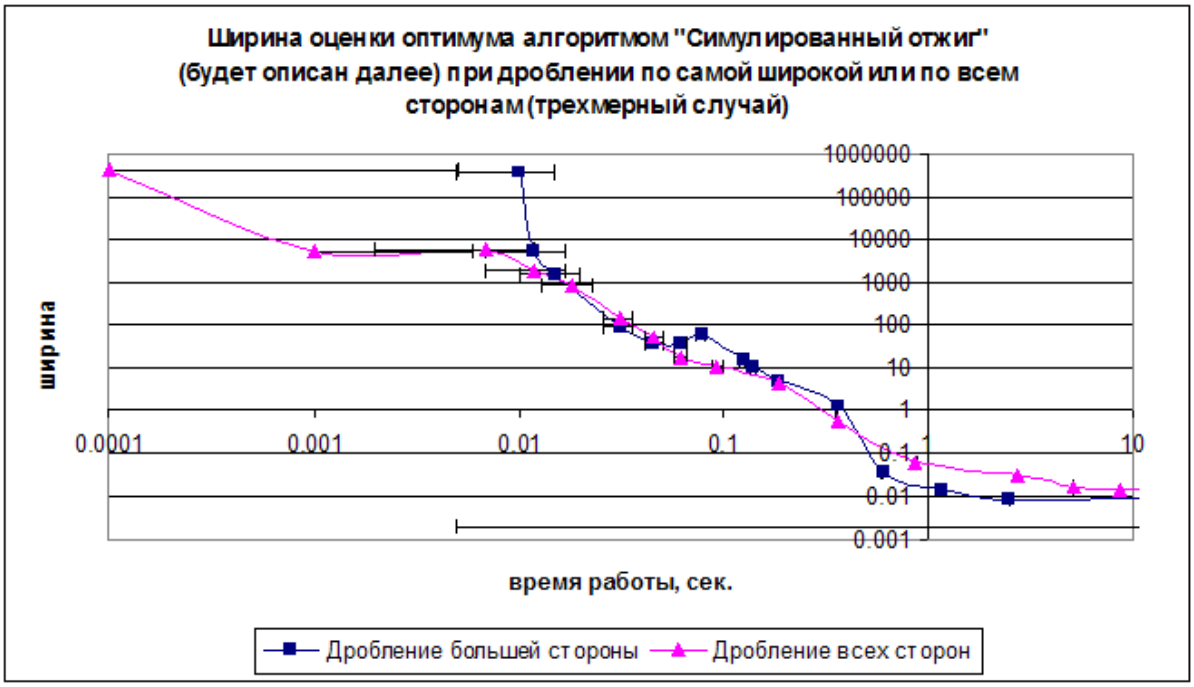


Рис. 2.11: Точность интервальной оценки в зависимости от способа дробления брусов

при делении именно по этой координате.

Осталось ли ещё что-то для улучшения? Мы рассмотрели уже все шаги алгоритма, за исключением процедуры выбора ведущего бруса.

2.7 Процедура выбора ведущего бруса

На первый взгляд, что-либо улучшить в существующей процедуре выбора ведущего бруса уже нельзя. Интервальное расширение функции даёт нам оценки минимального и максимального значения целевой функции на интересующем интервале. Более того, в силу фундаментальных свойств интервального расширения функции, гарантируется, что значения функции не выйдут за вычисленные границы на всём интервале. То есть, если мы нашли интервальное расширение $\boldsymbol{f}(\boldsymbol{x})$ целевой функции $f(x)$ на интересующей нас области $\boldsymbol{x}$, то $\underline{\boldsymbol{f}(\boldsymbol{x})}$ — это оценка искомого глобального минимума снизу.

Таким образом, хотя интервальное расширение дает избыточную оценку, но, первое — мы уже знаем оценку оптимума, и, второе — мы знаем, что она асимптотически точна. Далее, дробя интересующую нас область на всё более мелкие подобласти и, соответственно, всё увеличивая точность оценивания, мы в принципе можем найти значение как глобального оптимума, так и аргументов, его доставляющих, с любой наперед заданной точностью.

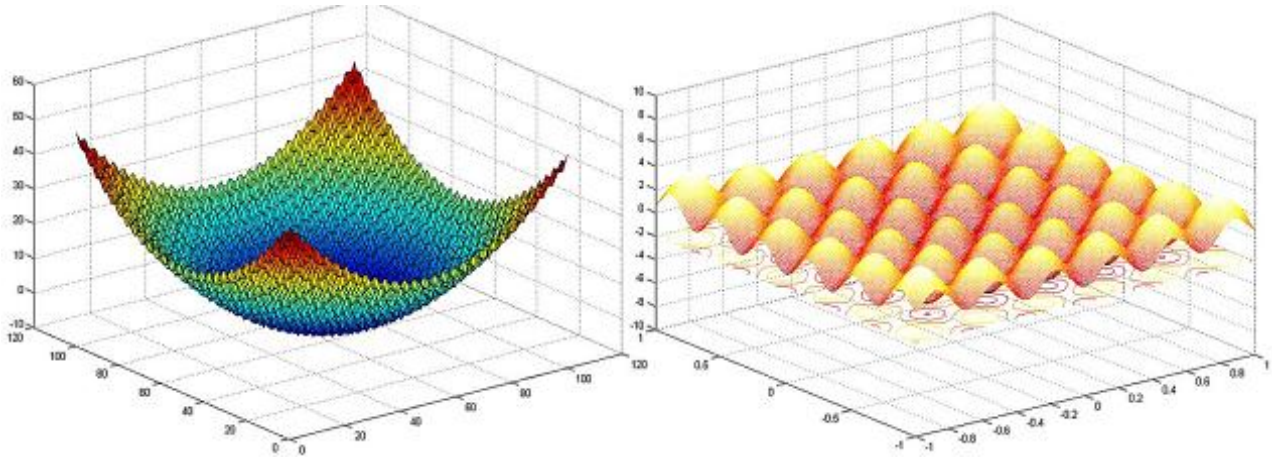


Рис. 2.12: Вид графика десятой функции Растригина (R10):
справа — крупномасштабная структура графика,
слева — увеличенная окрестность минимума

Но грубость интервальной оценки, а также возможное её застаивание, могут существенно ухудшить производительность нашего метода. На графике на рис. 2.13 сравнивается работа классического алгоритма адаптивного интервального дробления на двух целевых функциях.

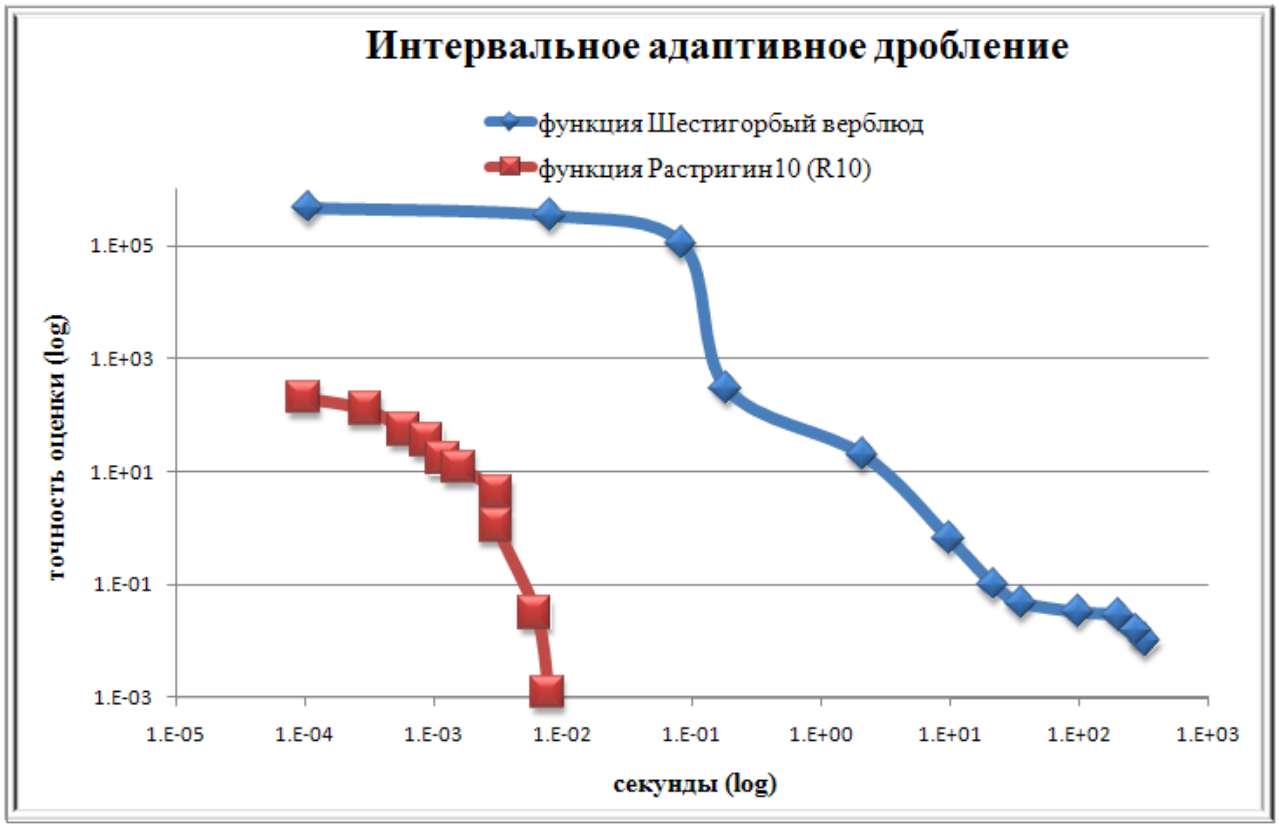


Рис. 2.13: График зависимости точности оценки глобального минимума двух целевых функций от времени для алгоритма адаптивного дробления. По обеим осям графика взят логарифмический масштаб. Время работы измерялось с точностью до 0.005 секунды, что отражено на графике в виде погрешности

Функция «Растригин10» (R10) [90], с её множеством локальных минимумов и максимумов (вид функции вблизи начала координат можно найти на рис. 2.12) неожиданно оказалась «лёгкой» для алгоритма — глобальный минимум был локализован за доли секунды, в то время как функция «Шестигорбый верблюд» (C10), см. рис. 1.6 и 1.7 на стр. 52 оказалась для данного алгоритма чрезвычайно сложной. Для нахождения минимума с высокой точностью пришлось затратить более трёхсот секунд, что делает метод совершенно неприемлемым.[1]

Прежде чем мы сможем улучшить метод, необходимо понять, в чём он неудовлетворителен. Почему на одной целевой функции алгоритм работает

[1] Ясно, что абсолютное время работы программы зависит от использовавшегося аппаратного обеспечения, и приводится здесь лишь для сравнения алгоритмов между собой.

превосходно, но не справляется с другой?

На синем графике на рис. 2.13 хорошо видны проявления эффекта застаивания интервальной оценки, когда за большой промежуток времени (на графике приведены логарифмические шкалы по обеим координатам) и, соответственно, и за большóе количество дроблений, точность интервальной оценки улучшилась лишь незначительно. Но это лишь одна из причин. Другая становится очевидной, если рассмотреть работу алгоритма на этой целевой функции шаг за шагом.

В процессе разработки алгоритмов мы встретились с тем, что при работе над достаточно сложными задачами, зачастую приходится тратить много сил, чтобы понять, что происходит на той или иной итерации алгоритма и оптимально ли его поведение. Также бывает сложно понять пути улучшения алгоритма, не представляя всю картину в целом. Вычислительные алгоритмы сохраняют свое состояние на каждой итерации (см. листинг 6), но ценность этой информации стремительно падает с увеличением размера файла сохраняемого файла протокола. Из-за достаточно большого времени работы и немалого количества получающихся подбрусов, делать это напрямую оказалось совершенно неэффективно, а порой и невозможно.

Так встала задача наглядного представления работы алгоритмов и для этих целей была разработана специальная программа, по своей логике являющейся развитием предыдущей работы авторов «Программный комплекс для графического представления процесса и результатов работы интервальных алгоритмов» [47]. Но требования, предъявляемые к программе, существенно изменились, появились новые технологии. Описываемая программа представляет совершенно новую, самостоятельную разработку. К сожалению, описание этой программы не укладывается в формат и направленность настоящего текста. Скажем лишь, что программа позволяет наглядно демонстрировать процесс работы интервальных оптимизационных алгоритмов для целевых функций, действующих из $\mathbb{R}^2$ в $\mathbb{R}$, отображая все изменения конфигурации брусов в трехмерном пространстве. Есть возможность вращать и приближать изображение, сравнивать поведение нескольких алгоритм одновременно. Программа написана на языке C++. В отличие от алгоритмической части, реализованной таким образом, чтобы она могла быть собрана для любой платформы, программа-визуализатор изначально предполагалась для работы под управлением операционной системой Windows. Поэтому для работы с трехмерной графикой выбрана технология Microsoft DirectX.

Возвращаясь к рассматриваемому вопросу, обсудим рис. 2.14, где приведены несколько кадров, показывающих, как менялась конфигурация рабочего

Листинг 6. Пример сохраненных сырых данных алгоритма

```
{ [-0.328125, -0.3125], [0.9375, 0.953125], [1.13776, 1.22863] }
{ [0.4375, 0.445313], [0.03125, 0.0390625], [1.1382, 1.18729] }
{ [-0.253906, -0.25], [0.974609, 0.976563], [1.13829, 1.1532] }
{ [0.4375, 0.445313], [0.0390625, 0.0429688], [1.13875, 1.18761] }
{ [-0.257813, -0.25], [0.976563, 0.984375], [1.13904, 1.17678] }
{ [0.4375, 0.445313], [0.0429688, 0.046875], [1.13907, 1.18796] }
{ [-0.3125, -0.28125], [0.96875, 0.984375], [1.13907, 1.28017] }
{ [-0.890625, -0.875], [-0.578125, -0.5625], [1.13927, 1.45658] }
{ [0.4375, 0.439453], [0.046875, 0.0478516], [1.13942, 1.15153] }
{ [0.4375, 0.439453], [0.0478516, 0.0488281], [1.13951, 1.15162] }
{ [-0.257813, -0.253906], [0.972656, 0.976563], [1.13955, 1.15849] }
{ [0.4375, 0.441406], [0.0488281, 0.0507813], [1.13961, 1.16391] }
{ [-0.375, -0.367188], [0.890625, 0.894531], [1.13976, 1.18443] }
{ [0.4375, 0.439453], [0.0507813, 0.0527344], [1.1398, 1.15202] }
{ [0.4375, 0.438477], [0.0527344, 0.0537109], [1.14, 1.14611] }
{ [0.4375, 0.438477], [0.0537109, 0.0546875], [1.14011, 1.14621] }
{ [0.4375, 0.441406], [0.0546875, 0.0585938], [1.14021, 1.16477] }
{ [0.4375, 0.441406], [0.0585938, 0.0625], [1.14066, 1.16524] }
{ [0.4375, 0.5], [-0.125, -0.0625], [1.14113, 1.57813] }
{ [0.4375, 0.46875], [0.0625, 0.09375], [1.14113, 1.34796] }
{ [-0.90625, -0.890625], [-0.570313, -0.5625], [1.14127, 1.45911] }
{ [-0.265625, -0.257813], [0.972656, 0.976563], [1.14138, 1.17233] }
{ [-0.9375, -0.921875], [-0.578125, -0.5625], [1.14217, 1.48849] }
{ [-0.921875, -0.90625], [-0.578125, -0.5625], [1.14224, 1.47872] }
{ [-0.328125, -0.320313], [0.929688, 0.9375], [1.14393, 1.18971] }
```

списка при поиске глобального оптимума десятой функции Растригина (R10). Хорошо видно, что алгоритм ни разу «не ошибся» — каждое дробление дей-

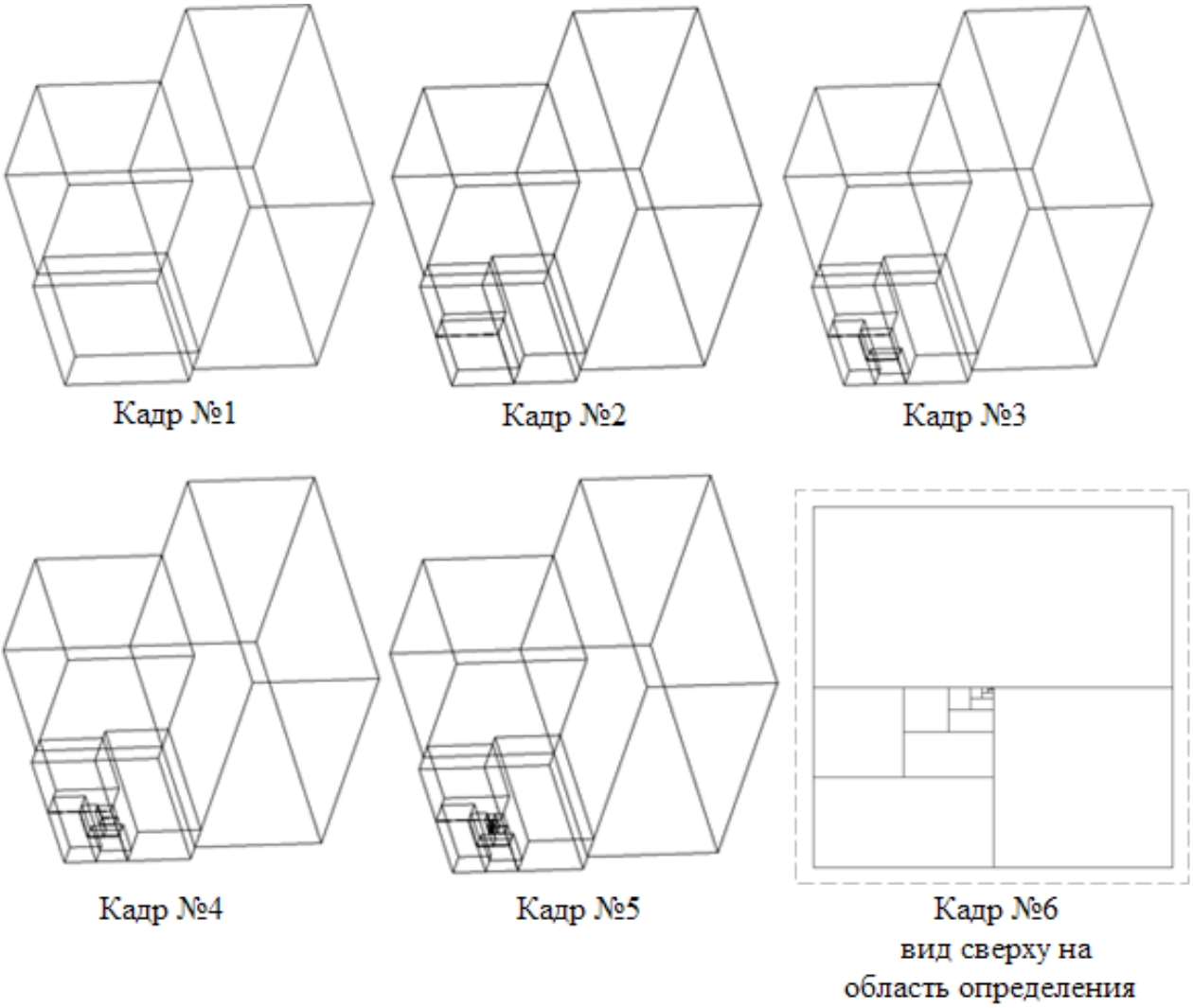


Рис. 2.14: «Визуализация» работы алгоритма интервального адаптивного дробления на целевой функции «Растригин10» (R10)

ствительно уточняло оценку глобального оптимума. Это отчетливо видно на последней иллюстрации на рис. 2.14. На ней приведен «вид сверху» — показано, как дробилась исходная область определения.

На рис. 2.15 приведена аналогичная конфигурация для функции «Шестигорбый верблюд» (C10). К сожалению, из статичной картинки трудно что-то понять, так что обратимся к двухмерным проекциям, представленным на рис. 2.16. На нем проиллюстрировано, как дробилась исходная область определения при поиске глобального оптимума функции «Шестигорбый верблюд» (C10).

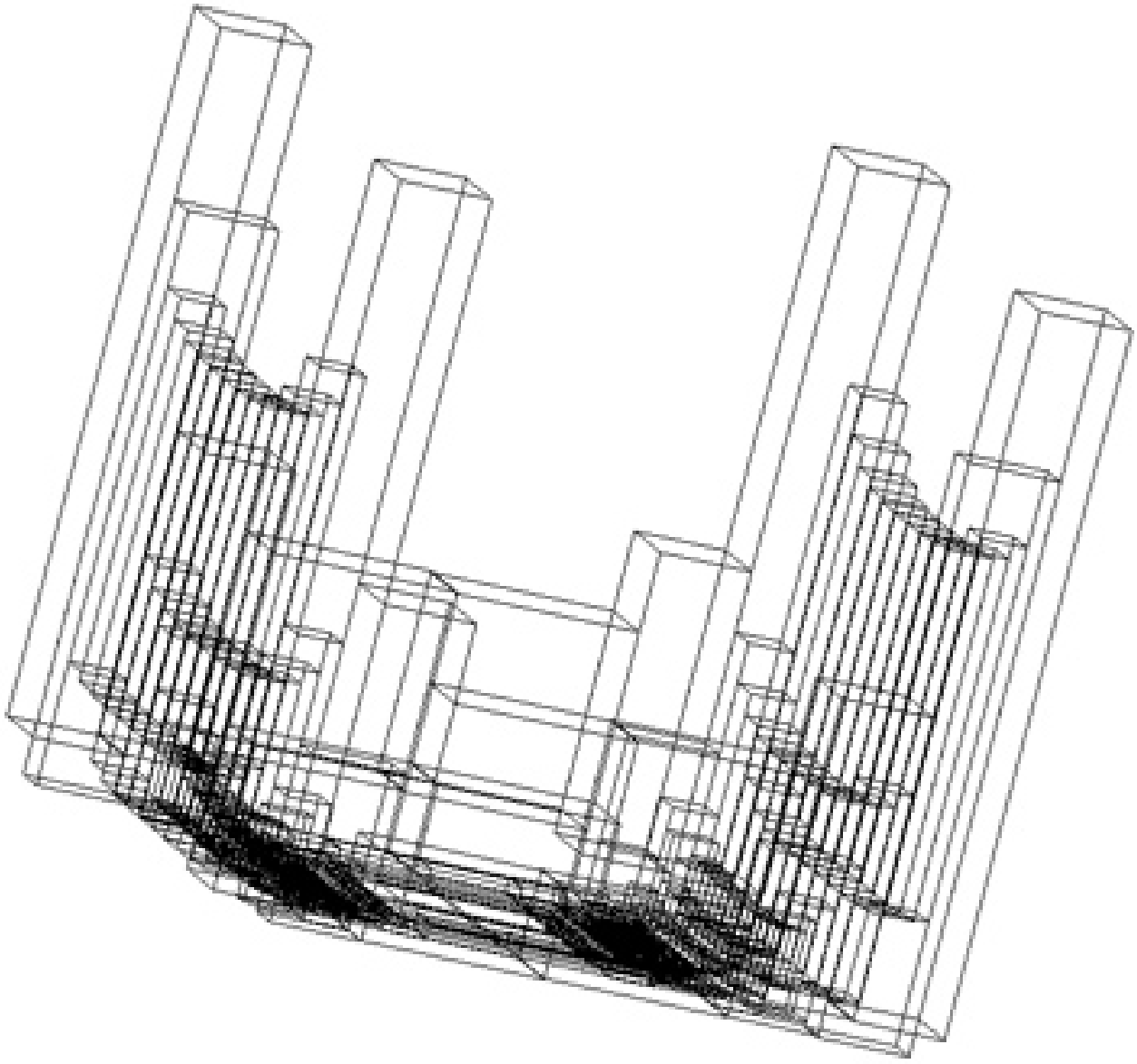


Рис. 2.15: Трехмерная конфигурация брусов при поиске глобального минимума целевой функции «Шестигорбый верблюд» (C10) алгоритмом интервального адаптивного дробления

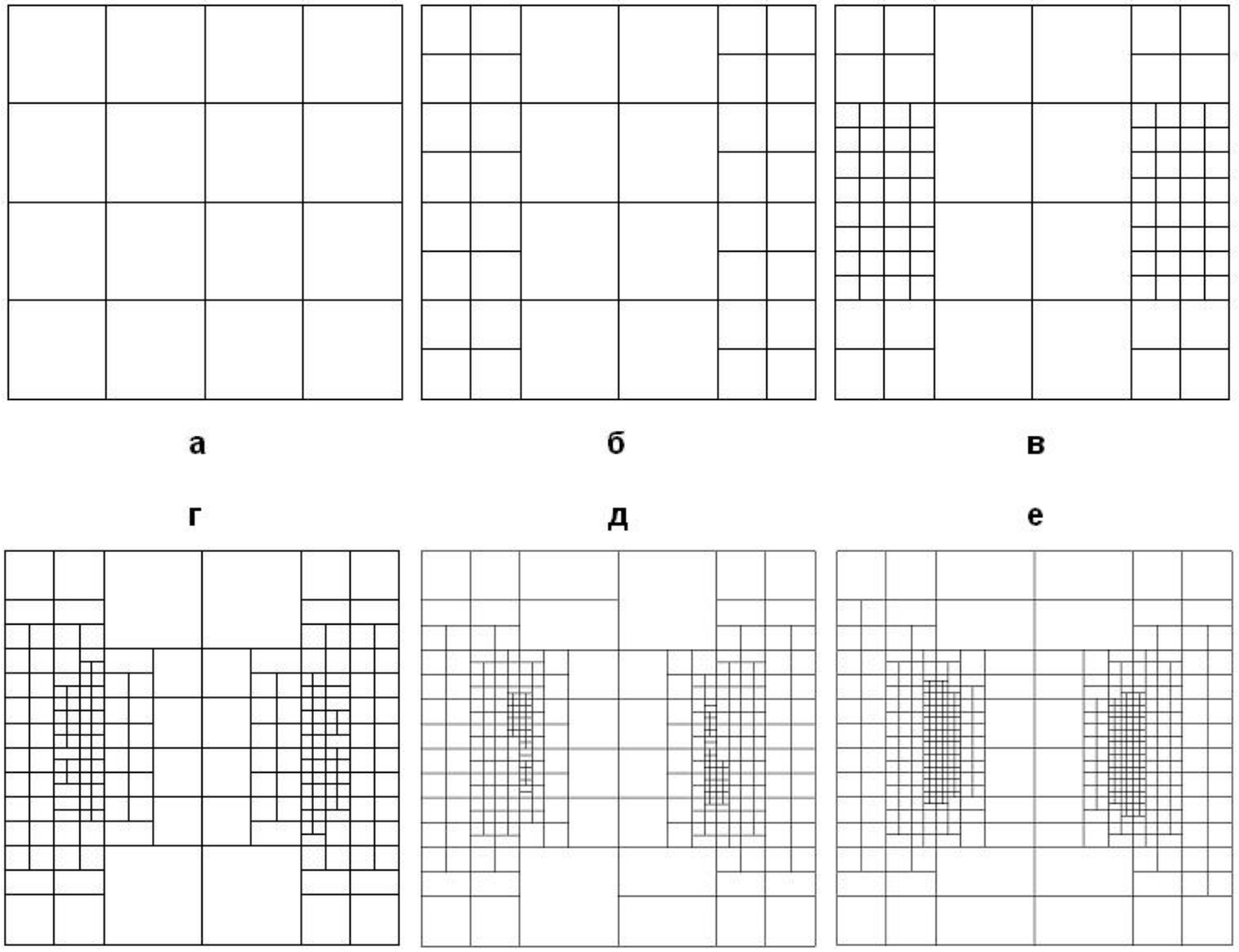


Рис. 2.16: Дробление исходной области определения алгоритмом интервального адаптивного дробления при поиске глобального минимума целевой функции «Шестигорбый верблюд» (С10)

Важно отметить, что для большей наглядности процедуры отбраковки были отключены.

Нами был проделан достаточно подробный анализ поведения алгоритма при оптимизации различных функций, здесь же приведем лишь краткие выводы. В отличии от процесса на рис. 2.14 «ведущий брус» постоянно меняется, алгоритм последовательно мельчит соседние брусы. При этом точность оценивания улучшается лишь незначительно — имеет место эффект «застаивания интервальной оценки». То есть ситуация, когда достаточно большое количество последовательных дроблений не привело к значимому улучшению точности интервальной оценки (см. рис. 2.17). Дополнительно усугубляют дело «ложные оптимумы», то есть брусы, признаваемые ведущими, хотя реально и не содержащие глобального оптимума. Всё это вместе взятое приводит к раздутию рабочего списка. (Ещё раз напомним, что иллюстрации на рис. 2.16 для большей наглядности получены при отключенных процедурах отбраковки).

Итак, обозначим основные причины неуспешности алгоритма:

— Получаемые интервальные оценки области значений функции весьма избыточны.

— Зачастую границы интервальных оценок целевой функции на подбрусах не коррелировали с действительными областями значений. То есть, несмотря на то, что $\min\limits_{\boldsymbol{a}} f(x) > \min\limits_{\boldsymbol{b}} f(x)$, величина $\underline{\boldsymbol{f}(\boldsymbol{a})}$ часто оказывалась меньше, чем $\underline{\boldsymbol{f}(\boldsymbol{b})}$, где $\boldsymbol{a}$ и $\boldsymbol{b}$ — два каких-то подбруса, не вложенных друг в друга. (См. рис. 2.17).

— Эффект застаивания интервальной оценки вместе с двумя указанными выше явлениями заставляет алгоритм делать очень много дроблений впустую, фактически не приближаясь к глобальному оптимуму.

— Растущий размер рабочего списка, в свою очередь (см. §2.3), дополнительно замедлял процесс оптимизации.

Из рис. 2.17 должно быть понятно, что алгоритмы рассматриваемого нами типа основанные на адаптивном дроблении области определения и интервальном оценивании по получающимся подобластям, запрограммированы на неудачу в задачах такого рода. В соответствии со своей внутренней логикой они будет последовательно мельчить ложные ведущие брусы, лишь незначительно улучшая точность интервальной оценки.

Один из возможных выходов из создавшегося положения — изменение процедуры выбора бруса для дробления. Естественным, хотя и несколько

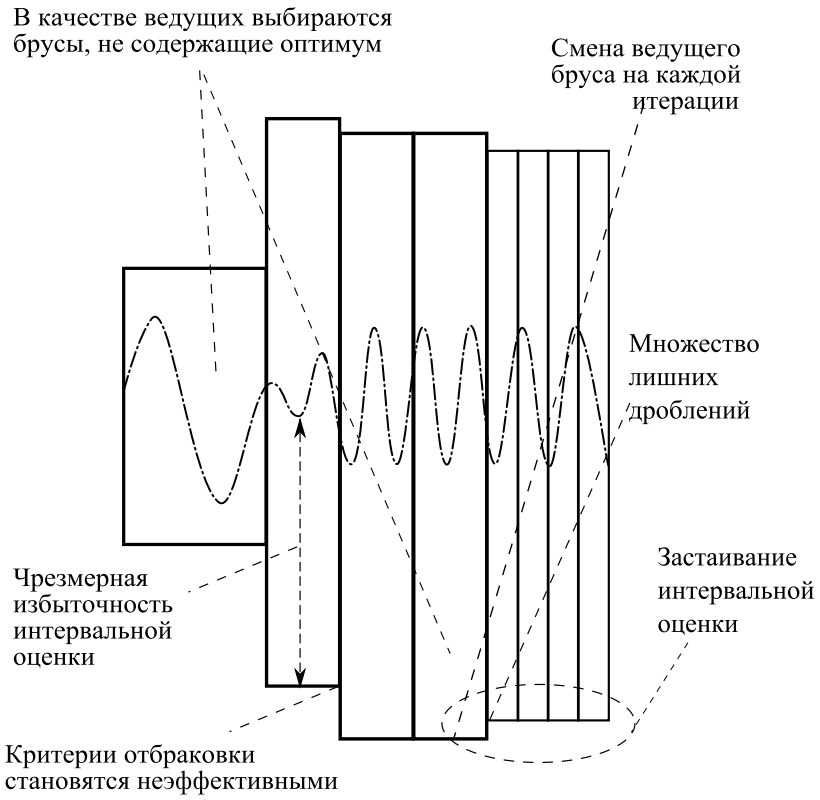


Рис. 2.17: Детерминистский алгоритм последовательно дробит область определения на всё более мелкие подбрусы, при этом практически не уточняя границы глобального оптимума

необычным шагом является введение случайности (рандомизации) в этот процесс. Впервые подобная идея — отказаться от жёсткого детерминизма классических методов интервальной оптимизации с целью получения принципиально новых алгоритмов — была предложена С.П. Шарым в публикациях [70, 71].

Первым на этом пути получается алгоритм «случайного интервального дробления» [70, 71], блок-схема которого изображена на рис. 3.1.

Итоги главы 2

Основной итог главы 2 диссертации — тщательный критический анализ вычислительной схемы традиционных интервальных методов глобальной оптимизации, позволивший понять их узкие места и наметить пути их преодоления. Новыми являются также выводы о сравнительной эффективности различных модификаций интервальных методов глобальной оптимизации, процедур отбраковки и т. д. Приведены рекомендации по наиболее оптимальной программной реализации соответствующих алгоритмов на различных вычислительных системах.

Отметим, что для выполнения этой работы потребовалось создать специальный программный инструмент визуализации, который в наглядной форме представляет процесс работы интервальных методов глобальной оптимизации и позволяет анализировать их производительность в реальном времени.

Глава 3

Стохастические интервальные методы глобальной оптимизации

Эта глава является центральной в нашей работе и посвящена разработке, реализации, тестированию и сравнительному анализу интервальных алгоритмов глобальной оптимизации, основанных на использующих стохастики и рандомизации. Мы рассматриваем и анализируем ряд существующих алгоритмов этого типа, предложенных С.П. Шарым, предлагаем их возможные модификации, а также развиваем на этом пути новые подходы — интервальный генетический алгоритм и мультиметодный алгоритм, сочетающий в себе положительные стороны большинства из составляющих его вычислительных схем.

3.1 Случайное интервальное дробление

Алгоритм случайного интервального дробления, блок-схема которого представлена на рис. 3.1, является простейшим из методов, сочетающих интервальную технику оценивания значений функций со стохастическим управлением. Он был предложен в работах [70, 71], в которых также обосновывалась необходимость именно такого способа работы с областью определения — дробления вкупе с организацией списка, если для получения информации о целевой функции используются её интервальные расширения. При этом вероятности извлечения брусов из рабочего списка полагаются равными.

Мы применили алгоритм случайного интервального дробления для нахождения глобального минимума функции (1.15):

$$z(x, y) = x^4 + 4x^3 + 4x^2 + y^2$$

на начальном брусе $[-1, 1] \times [-1, 1]$.

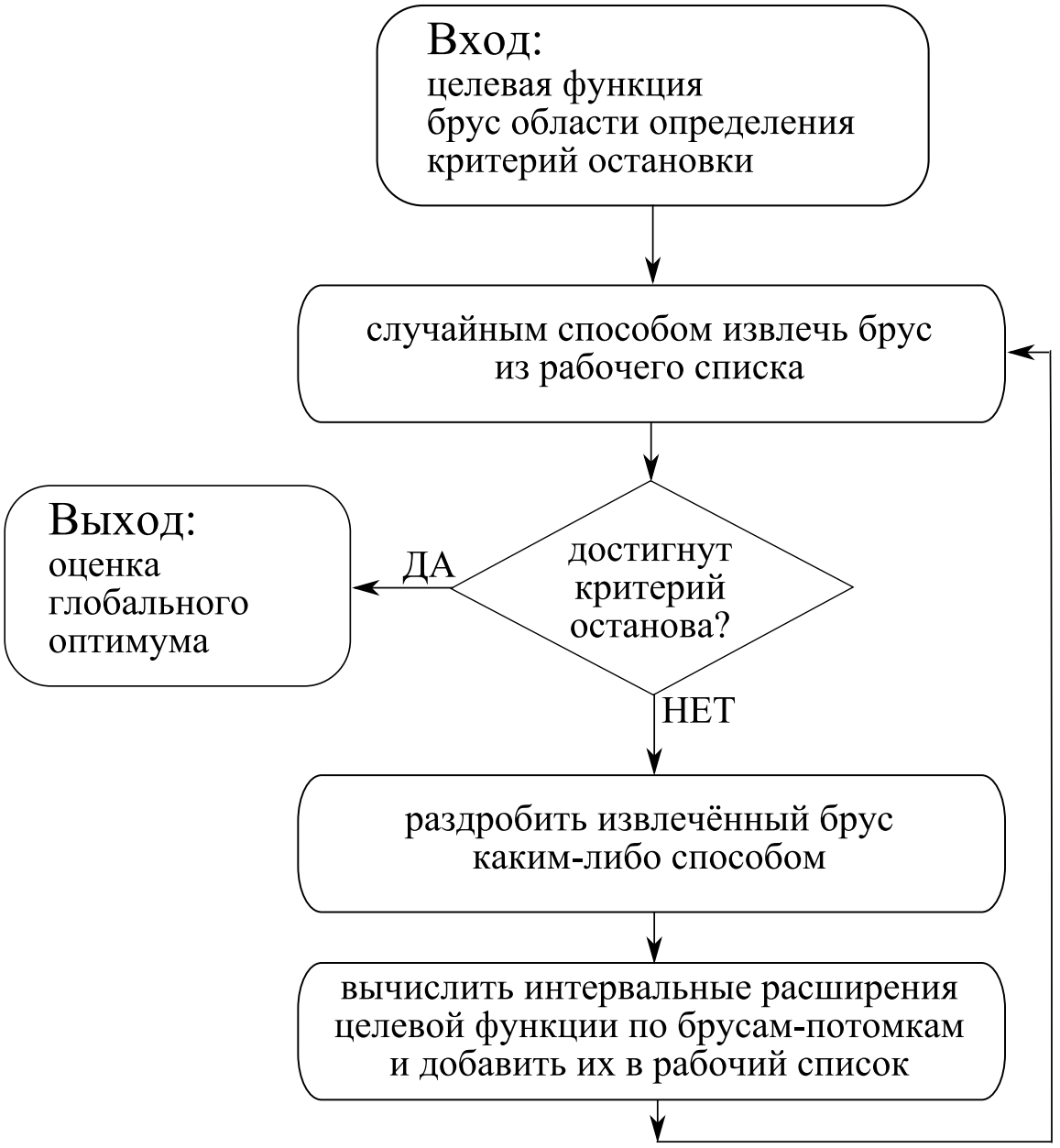


Рис. 3.1: Блок-схема случайного интервального дробления

Через 0.01 секунды минимум был локализован как находящийся на брусе $[-1, 0] \times [-0.125, 0]$, а соответствующие значение целевой функции — в интервале $[-1.4, 3.3]$. При этом список брусов имел вид, показанный на диаграмме на рис. 3.3.

Практически в соответствии с этими свойствами распределялась и точность интервальной оценки функции.

Буквой «D» на диаграммах обозначена ширина брусов. Видно, что основной процент составляют достаточно мелкие брусы, но есть и несколько крупных. Несмотря на достаточно хорошее распределение, эти несколько нераз-

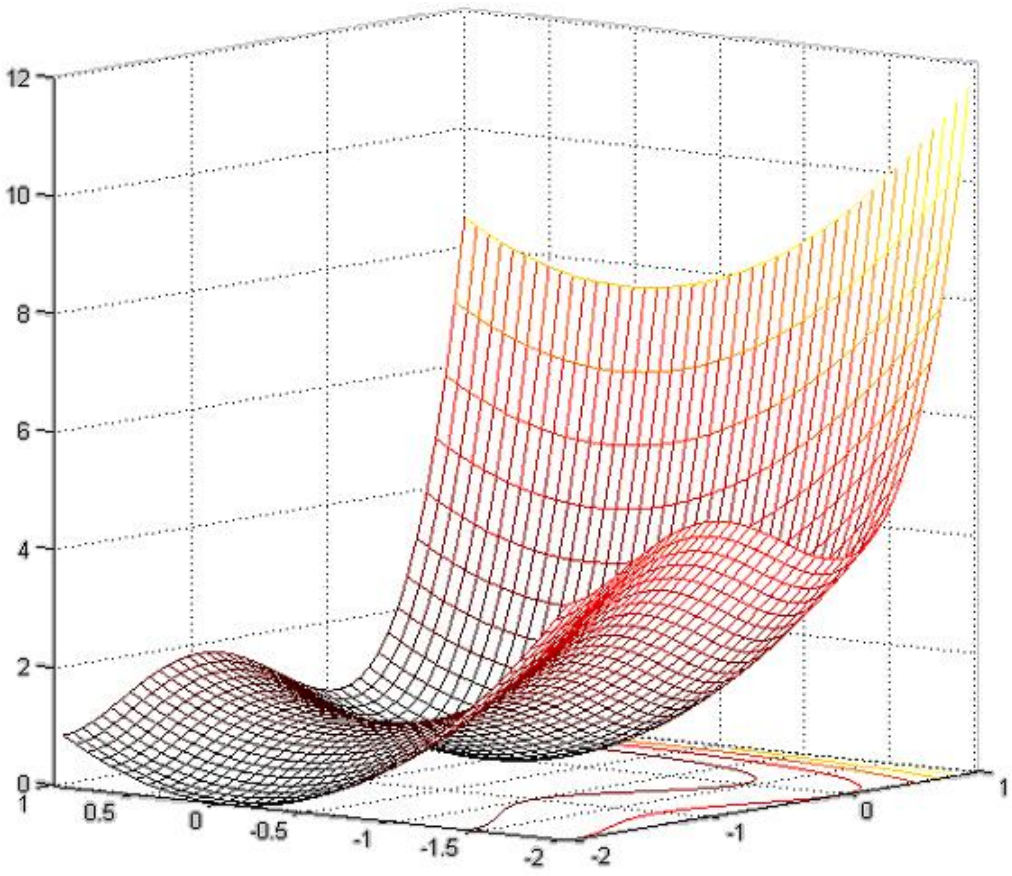


Рис. 3.2: Вид функции $z(x, y) = x^4 + 4x^3 + 4x^2 + y^2$ (Tre4) вблизи начала координат

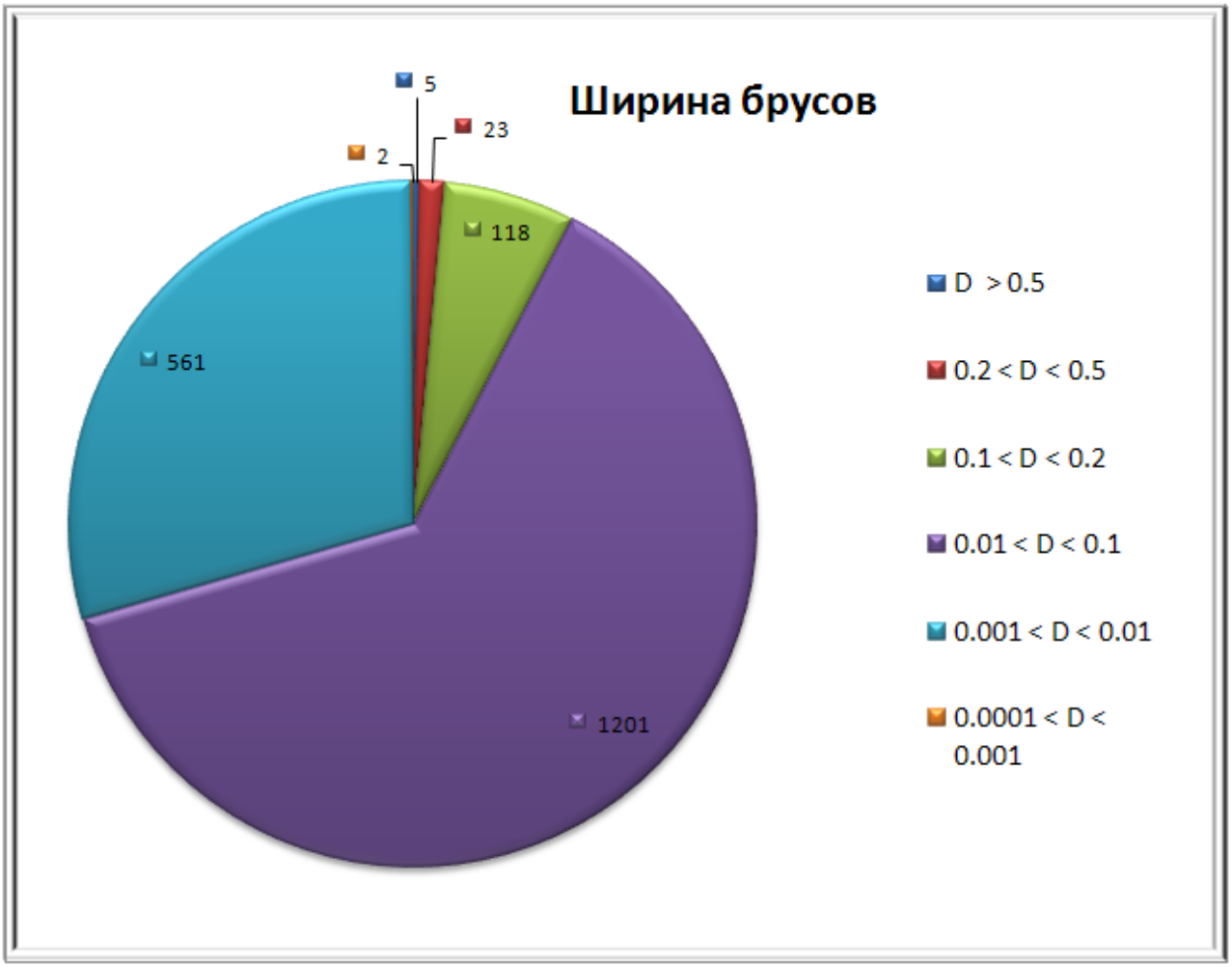


Рис. 3.3: Распределение ширины брусов в рабочем списке алгоритма «случайное интервальное дробление» через 0.01 секунды после старта

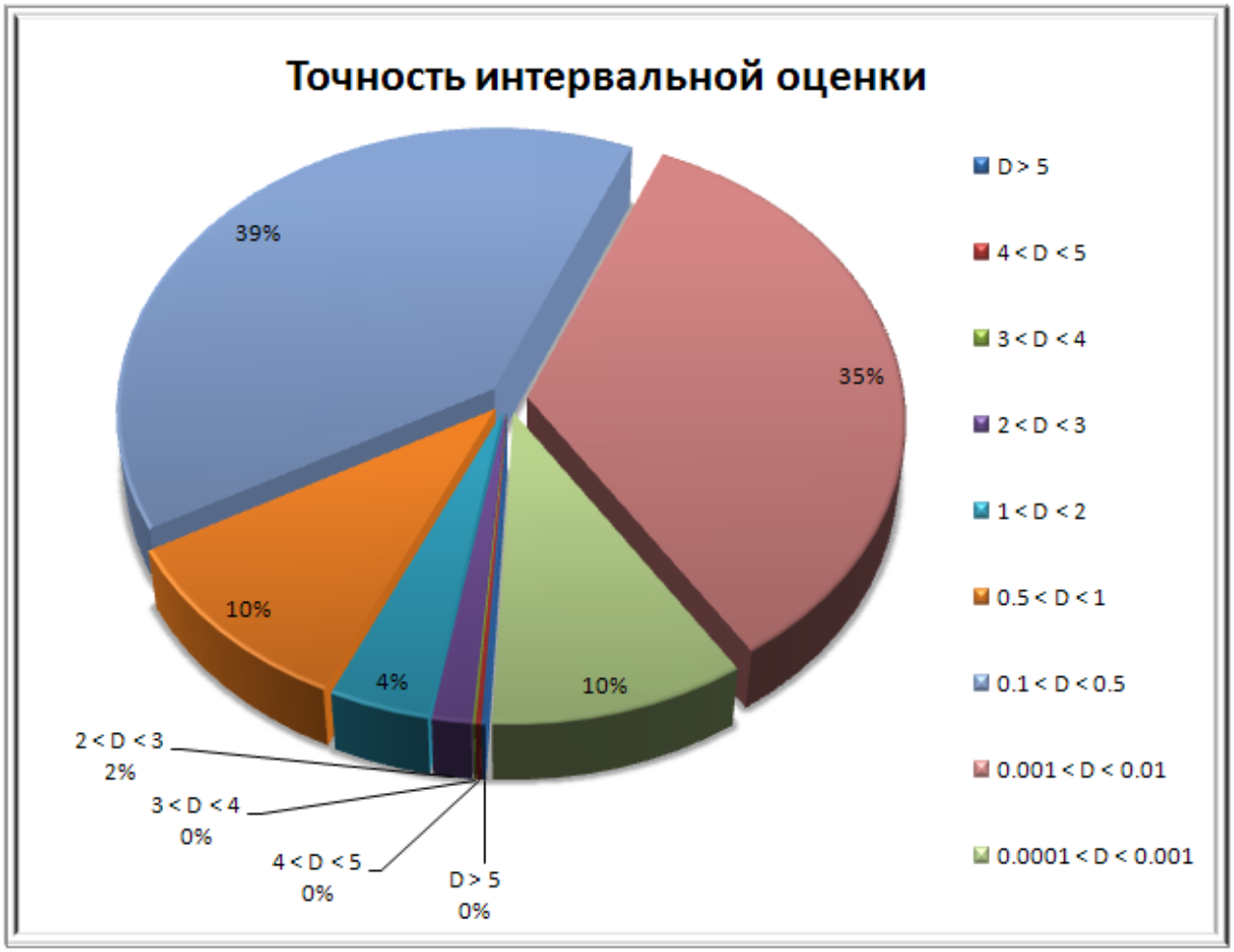


Рис. 3.4: Распределение ширины интервальной оценки целевой функции в рабочем списке алгоритма «случайное интервальное дробление» через 0.01 секунды после старта

дробленных брусов и портят всю картину. С увеличением количества итераций сколько-либо значимого улучшения значения оптимума не происходит — ведь с каждым дроблением количество брусов становится больше, а вероятность выбрать самый широкий брус — меньше. В частности, в нашем эксперименте с функцией (1.15) оценка оптимума не была существенно улучшена даже за тысячу секунд.

Графики на рис. 3.5 показывают усреднённые по ста (для каждой из размерностей) экспериментам размеры максимальной стороны в рабочем списке в зависимости от количества итераций, совершённых методом интервального случайного поиска. Обратите внимание на логарифмический масштаб по времени.

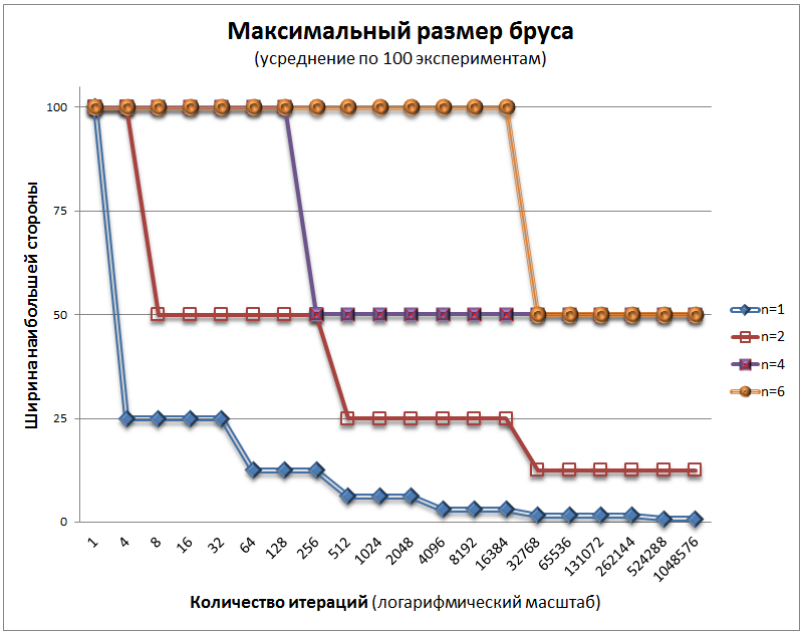


Рис. 3.5: Численное моделирование сходимости метода случайного дробления для функций малых размерностей

Если в качестве меры бруса опираться не максимальный размер, а, скажем, на средний, то ситуация (см. рис. 3.6) принципиально не изменится.

Также хочется отметить ещё одну особенность метода. Без процедур отбраковки на каждой следующей итерации размер рабочего списка увеличивается. При этом выбор бруса происходит из всего списка. Это означает, что рано или поздно объём обрабатываемых данных превысит размер любого кэша. После чего данные будут вытесняться в память и скорость выборки будет существенно замедляться. Это иллюстрирует рис. 3.7, на котором приведена зависимость времени одной итерации (в некоторых условных внутренних единицах измерения) в зависимости от количества итераций. Подробнее см.

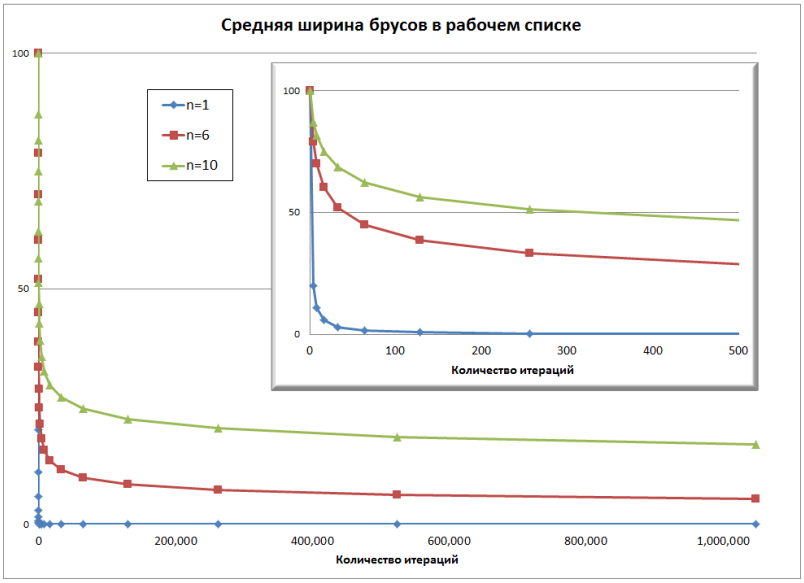


Рис. 3.6: Численное моделирование сходимости метода случайного дробления для функций малых размерностей

раздел 2.3. Сравнительная медленность первых пяти итераций объясняется тем, что Java-машина ещё не всё скомпилировала.

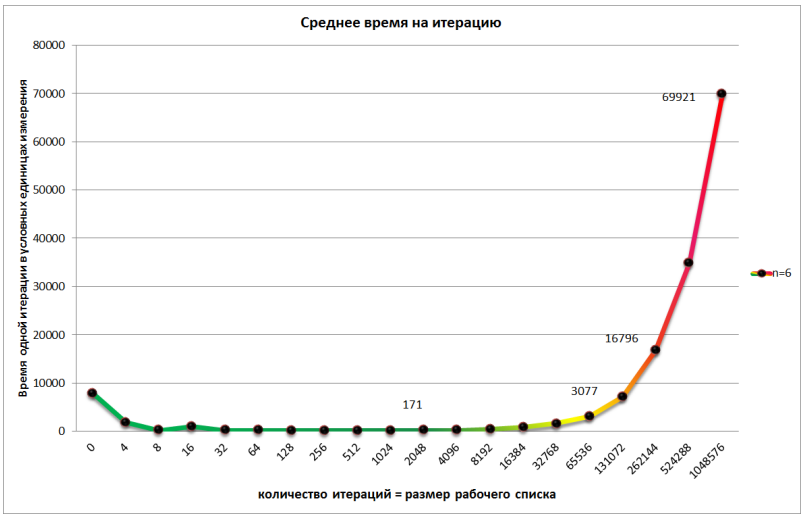


Рис. 3.7: Усреднённое время одной итерации интервального случайного дробления

Таким образом, наши вычислительные эксперименты пока показывают, что точность оценки глобального оптимума рассматриваемым методом довольно быстро выходит на асимптоту и при дальнейших итерациях значительно не улучшается. Алгоритм случайного интервального дробления не тратит время на такие второстепенные задачи, как упорядочивание рабочего списка или поиск наиболее перспективного бруса, а лишь дробит и дробит.

Как следствие отсутствия управляющей логики, он так же быстро «погрязает» в порожденном множестве подбрусов. Можно ли помочь простейшему стохастическому алгоритму интервальной глобальной оптимизации не «захлебнуться» результатами своего труда? Дополнение его процедурами отбраковки позволило отсеивать более половины брусов и локализовать минимум с абсолютно точностью 10^{-4} за 0.38 секунды. Правда, чтобы повысить точность до 10^{-5} пришлось потратить ещё две секунды. Важно понимать, что использование процедур отбраковки позволит локализовать оптимум методом случайного дробления далеко не на любой целевой функции.

В отсутствии осмысленной логики выбора ведущего бруса любой способ отбрасывания бесперспективных брусов способствует «целеустремленности» алгоритма. В целом, с одной стороны мы имеем жёстко детерминированную алгоритмику классического метода «ветвей и границ», а с другой, слишком свободную вычислительную схему «случайного интервального дробления». Проблемы обоих очевидны. С одной стороны, это застаивание интервальной оценки и ложные оптимумы для классического адаптивного интервального дробления и неадаптивность для «случайного интервального дробления», с другой.

Попробуем это исправить и модифицируем процедуру выбора ведущего бруса так, чтобы более широкие брусы выбирались с большей вероятностью.

3.2 Случайное интервальное дробление с приоритетом

Метод, который мы назвали «случайный интервальное дробление с приоритетом» является естественным развитием алгоритма «случайного интервального дробления», описанного в предыдущем пункте. Единственное, но весьма существенное отличие — мы модифицировали процедуру выбора очередного кандидата на дробление. Выбор по-прежнему происходит случайно, но более широкие брусы имеют бóльшую вероятность быть раздробленными.

Технически это можно реализовать несколькими способами.

Например, естественно поддерживать рабочий список отсортированным по размеру брусов и использовать генератор случайных чисел с неравномерной плотностью вероятности (в этом случае добавляются дополнительные расходы на вставку подбрусов в упорядоченный список).

Альтернативный образ действий состоит в том, чтобы отслеживать и помнить размер наиболее широкого (широких) бруса (брусов) и дробить только те брусы, которые «не слишком меньше» этих самых широких. Для наглядности этот метод проиллюстрирован псевдокодом на листинге 7. Основная

сложность и, следовательно, дополнительные расходы при таком подходе выпадают на отслеживание наиболее широких брусов и их количества. Кроме того, недостатком является то обстоятельство, что с увеличением размера рабочего списка (а это неизбежно происходит при оптимизации достаточно сложной целевой функции), количество неуспешных попыток найти достаточно широкий брус будет всё больше. Соответственно, увеличится и длительность каждой итерации. Кроме того, более эффективным представляется организовать приоритет брусов по ширине интервальной оценки или по комбинации нижней границы, ширины оценки и области определения.

Для начала будем предпочтительно дробить брусы с более широкой интервальной оценкой целевой функции, чтобы преодолеть основной недостаток алгоритма случайного дробления. Реализовать такой подход также можно несколькими способами.

В реализованной и исследованной нами версии мы храним в рабочем списке все брусы отсортированными по ширине интервальной оценки, делённой на ширину начальной оценки области значений в сочетании со значением нижней границы интервальной оценки, делённой на значение нижней границы начальной оценки. Кроме того, применялся генератор случайных чисел с кусочно-равномерным распределением вероятности:

$$\mathsf{P} = \sum_{i=1}^{M} \frac{1}{k_i} \cdot \frac{1}{n_i} = 1$$

Иными словами, на каждой итерации с вероятностью $\frac{1}{k_1}$ выбирался один из первых n_1 брусов в рабочем списке, с вероятностью $\frac{1}{k_2}$ — брус с номером от n_1 до n_2. Причём $\frac{1}{k_1} > \frac{1}{k_2}$ и т. д. Типичные значения, используемые нами в вычислительных экспериментах при размере рабочего списка более 1000 брусов были следующими: $M = 5, n_1 = 20, n_2 = 100, n_3 = 300, n_4$ — половина n_5, n_5 — количество всех брусов. Реализовано это следующим образом: сначала случайным образом выбирался диапазон, где будет производиться поиск, а затем уже из этого диапазона выбирался брус для дробления случайным образом.

При такой реализации добавляются дополнительные расходы на добавление подбрусов в упорядоченный список. Зато мы повышаем локальность данных, что положительно сказывается на производительности алгоритма. Напомним, что в предыдущем разделе было показано, что случайный выбор бруса из достаточно объемного рабочего списка — весьма медленная процедура, гораздо предпочтительнее как можно больше работать с локальными, расположенными в памяти компактно, данными.

Листинг 7. Вариант алгоритма «Случайного интервального дробления с приоритетом»

```
. . .

// Исходный брус помещаем в рабочий список.
m_boxQueue.add(X);

// m_widthMap - это объект, считающий сколько брусов имеют тот
// или иной размер.
// Регистрируем исходный брус.
m_widthMap.add(X->size());

// Параметры алгоритма:
aspect = 10;          // во сколько раз меньше наиболее широкого
                      // может быть брус, чтобы его ещё можно
                      // было раздробить.

attempts = 500;       // максимальное количество попыток найти
                      // <<достаточно широкий>> брус

    . . .


// основной цикл
do {
      // Увеличиваем счётчик попыток.
      i++;

      // Извлекаем случайный брус из рабочего списка.
      Box* workBox = m_boxQueue.getRandom();

      // В случае, если этот брус слишком мал, предпринимаем
      // следующую попытку, если ещё не исчерпан лимит попыток
      if (i < attempts && workBox->size() < m_widthMap.maxWide()/aspect )
            continue;

      // Брус принят, обнуляем счетчик попыток.
      i = 0;

      // На один брус с таким размером станет меньше.
      m_widthMap.del( workBox->size() );

      // Дробим брус каким-либо способом.
      _temporaryLine = algorithm->splitBox(workBox);
```

```
        // Пробегаем по получившимся подбрусам
        // и регистрируем их размер в m_widthMap
        for (SomeIterator it = _temporaryLine.begin();
                          it < _temporaryLine.end(); ++it)
        {
             m_widthMap.add( (*it)->size() );
        }

        // Добавляем новые брусы в рабочий список
        m_boxQueue.merge(_temporaryLine);

        // Цикл повторяется.
    }  while ( ++step_counter < max_step )
    // для простоты приводится лишь один критерий останова:
    // уточнение оптимума будет продолжаться max_step итераций.

. . .
```

Докажем, что метод случайного интервального дробления с приоритетом сходится к глобальному оптимуму почти всегда (почти всюду).

Определение 3.1 *Говорят, что последовательность случайных величин* $\{\xi_n\}$ *сходится почти всегда к* ξ *при* $n \to \infty$, *и пишут:* $\xi_n \xrightarrow{п.в.} \xi$, *если* $\mathsf{P}\{\omega : \xi_n(\omega) \to \xi(\omega) \text{ при } n \to \infty\} = 1$. *Или же если* $\xi_n(\omega) \to \xi(\omega)$ *при* $n \to \infty$ *для всех* $\omega \in \Omega$, *кроме, возможно,* $\omega \in A$, *где* A — *множество меры нуль* [69].

Другими словами, если мы хотим доказать, что описываемый метод сходится почти всегда, нам необходимо доказать, что мера множества всех исходов, при которых алгоритм не сошёлся к глобальному оптимуму равна нулю.

Теорема 3.1 *Метод случайного интервального дробления с приоритетом сходится к глобальному оптимуму почти всегда (почти всюду).*

Доказательство. Пусть $f(x)$ — непрерывная функция на брусе $\boldsymbol{x} = \{a \leqslant x \leqslant b\} \supseteq \mathbb{R}^n$, $\boldsymbol{f}(\boldsymbol{x})$ — её интервальное расширение, Q_k — разбиение $\boldsymbol{x}$ на непересекающиеся подбрусы, получаемое на k шаге алгоритма. При этом верно, что

$$|Q_{k+1}| - |Q_k| \leqslant 1, \tag{3.1}$$

где посредством $|\cdot|$ обозначено количество элементов множества. Если рассматривать вариант метода не использующий процедур отбраковки и дробящий брусы надвое, то для (3.1) справедливо строгое равенство

$$|Q_{k+1}| - |Q_k| = 1. \tag{3.2}$$

При этом, по построению метода, на каждом шаге k существует такое множество подбрусов $N \subset Q_k$, $|N| \geqslant 0$, что для каждого подбруса $\boldsymbol{n} \in N$ и любого подбруса $\boldsymbol{m}$ из Q_{k-1} справедливо следующее векторное покомпонентное неравенство

$$\operatorname{wid} \boldsymbol{n} < \operatorname{wid} \boldsymbol{m}.$$

Отметим, что ситуация, когда $|N| = 0$ может достигаться лишь при работающих процедурах отбраковки, причём если $|N| = 0$, то $|Q_{k+1}| < |Q_k|$, так как какой-то брус был раздроблен, но все получившиеся подбрусы удовлетворили критериям отбраковки и были изъяты из рабочего списка.

Определим среднюю ширину брусов на k-ом шаге как вектор

$$S_k = \frac{\sum_{\boldsymbol{Q}_k \in Q_k} \operatorname{wid} \boldsymbol{Q}_k}{|Q_k|}, \tag{3.3}$$

где $\boldsymbol{Q}_k^i$ — i-ый подбрус из разбиения Q_k. Таким образом, с учётом (3.1) числитель S_k из (3.3)

$$\sum_{i=1}^{k} \operatorname{wid} \boldsymbol{Q}_k^i \leqslant \sum_{i=1}^{k-1} \operatorname{wid} \boldsymbol{Q}_{k-1}^i \leqslant \operatorname{wid} \boldsymbol{x},$$

(строгое равенство $\sum_{i=1}^{k} \operatorname{wid} \boldsymbol{Q}_k^i \equiv \text{const}$ достигается в случае, если процедуры отбраковки не работают или неэффективны). В то же время, знаменатель рассматриваемого выражения — неубывающая по k функция, так как $|Q_k| \leqslant |Q_{k+1}|$, и строгое равенство $|Q_k| = |Q_{k+1}|$ достижимо лишь на отдельных шагах алгоритма и только в том случае, когда процедуры отбраковки работают и весьма эффективны.

Соответственно, каждая компонента вектора S_k из (3.3) является невозрастающей функцией от k, более того,

$$S_k = \frac{\sum_{i=1}^{k} \operatorname{wid} \boldsymbol{Q}_k^i}{|Q_k|} \to 0 \qquad \text{при } k \to \infty. \tag{3.4}$$

Таким образом, мы показали, что средний размер брусов в процессе работы алгоритма уменьшается. Теперь рассмотрим, как ведёт себя максимальный размер бруса.

Интервальный алгоритм поиска с приоритетом на каждой итерации выбирает для дробления и уточнения интервальной оценки случайный брус. Поэтому даже если брус имеет очень большие размеры, не гарантируется, что он будет обязательно раздроблен. Но используемый приоритет дробления позволяет утверждать, что вероятность того, что он не будет раздроблен стремится к нулю. Иными словами, мера всех таких исходов алгоритма, когда в рабочем списке можно найти брус размером более ϵ, равна нулю.

Другими словами для любого $\epsilon > 0$

$$\mathsf{P}\{\text{wid } \boldsymbol{x}_n > \epsilon\} \to 0 \qquad \text{при } n \to \infty. \tag{3.5}$$

Как мы уже говорили, размер бруса связан с точностью интервальной оценки целевой функции на нём. Таким образом, при любой организации приоритета (по ширине бруса или же по ширине интервальной оценки на нём, и т. п.) вероятность того, что точность интервальной оценки при бесконечном количестве шагов окажется недостаточной, равна нулю: для любого $\delta > 0$

$$\mathsf{P}\{\text{wid } \boldsymbol{f}_n^* > \delta\} \to 0 \qquad \text{при } n \to \infty.$$

Как следствие, мера множества всех таких исходов работы алгоритма, когда точность интервальной оценки окажется большей наперёд заданной константы δ, равна нулю. С учётом определения 3.1 это эквивалентно тому, что алгоритм сходится на всём пространстве возможных исходов за исключением множества исходов меры нуль.

Эвристическое дробление со случайным выбором и дробление в случайной пропорции

Одна из первых идей, которая была опробована при работе над улучшением эффективности поиска глобального оптимума интервальными методами, была идея неравномерного дробления.

Фактически, это не что иное, как развитие метода «эвристического дробления». Метод, описанный в §2.6, прекрасно зарекомендовал себя на целевых функция для которых смена направления дробления происходит не слишком часто. Например, это справедливо для случая, показанного на рис. 2.14 на стр. 103). При частой смене направления дробления алгоритм практически не отличается от постоянного дробления одной только большей стороны. Худший вариант — постоянные переключения, но достаточно редкие для того, что бы эвристики направленного дробления уже успели сработать.

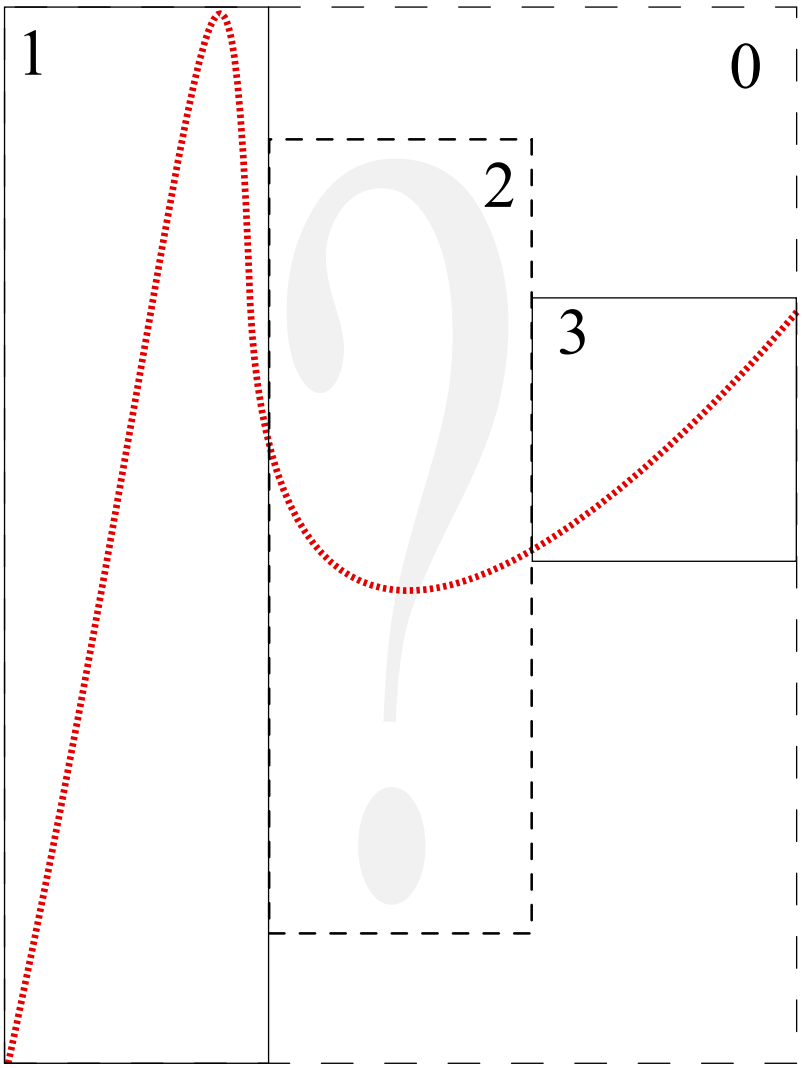


Рис. 3.8: Иллюстрация к методу трисекции

Для борьбы с подобными ситуациями мы, во-первых, добавили процедуру подсчёта переключений, и, во-вторых, разнообразили алгоритм возможностью случайного выбора стороны для дробления тогда, когда эвристика неустойчива. Развивая этот подход далее, мы перешли к неравномерному дроблению, т. е. рассечению ведущего бруса на две неравные части, конкретный размер которых зависит от предыстории.

Трисекция

В алгоритмах поиска, основанных на стратегии ветвей и границ, трисекция традиционно имеет определённые преимущества перед бисекцией, т. е. простым бинарным поиском. Применительно к нашей задаче трисекцию можно организовать следующим образом. Запомним границы интервальной оценки на очередном брусе, а затем раздробим его не на два, а на три подбруса. Но вычислим интервальное расширение целевой функции лишь на двух из них. Затем сравним их с сохранённой оценкой начального подбруса. Тут возможны следующие варианты (см. рис. 3.8).

Если на одном из новых подбрусов достигаются предыдущие границы интервального расширения, то экстремум функции, скорее всего, находится там, и этот брус и будет раздроблен на следующем шаге. Иначе разумно предположить, что оптимум лежит в третьем подбрусе, на котором интервальная оценка не вычислялась, и далее надо дробить именно его.

К сожалению, в интервальном анализе не всё так просто. Вышеописанный способ работал бы прекрасно, если бы границы интервального расширения функции всегда были бы точны. Применение этого приёма оправдано на завершающих этапах работы алгоритмов, когда брусы достаточно малы, и, следовательно, оценки целевой функции по ним весьма точны.

Вследствие упомянутой избыточности интервального расширения, рассмотренный выше способ дробления и выбора ведущего бруса можно использовать в качестве средства рандомизации классической детерминистской схемы на ранних этапах работы алгоритма оптимизации. В этом случае имеет смысл сравнивать границы интервальных расширений на равенство с некоторым допуском, который может быть некоей эвристической константой, либо же автоматически подбираться в процессе работы алгоритма.

3.3 Интервальный алгоритм имитации отжига

Фактически, с интервального алгоритма имитации отжига и началось развитие интервальных стохастических (рандомизированных) методов глобальной оптимизации. Впервые идея отказаться от жесткого детерминизма классических методов интервальной оптимизации с целью получения принципиально новых алгоритмов была предложена С.П. Шарым и в 2005 году представлена на XII Байкальской международной конференции «Методы оптимизации и их приложения» на примере интервальной версии алгоритма имитации отжига [70].

За основу взят достаточно популярный классический (точечный) метод (известный также как алгоритм Метрополиса [1, 21, 102]), моделирующий физические процессы отжига или кристаллизации и хорошо зарекомендовавший себя для многоэкстремальных проблем с большим количеством возможных решений [21, 104]. Упрощённый псевдокод этого алгоритма приведен в табл. 3.1.

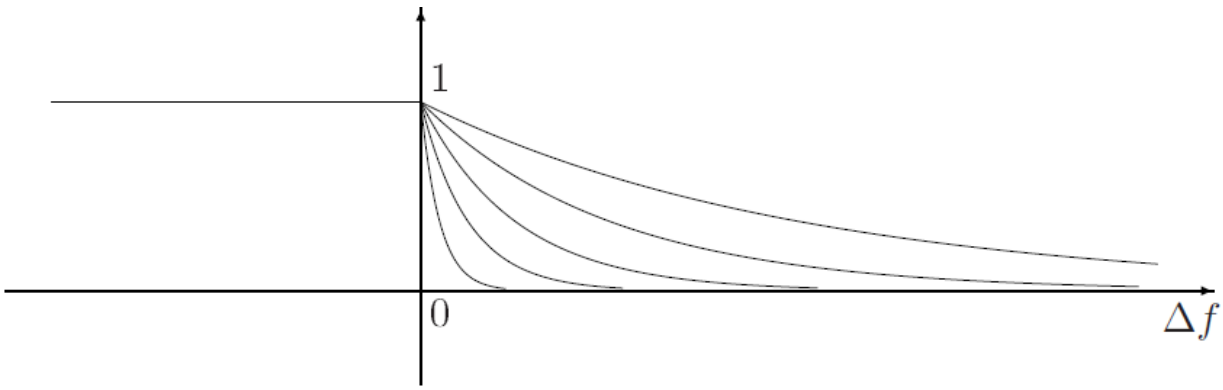


Рис. 3.9: Графики функций $\mathsf{P}_T(\boldsymbol{y}, \boldsymbol{z})$ для различных температур T.

Алгоритм оперирует таким понятием как «температура». Чем она выше, тем больше вероятность того, что на определённом шаге будет выбран брус, не доставляющий рекордную на данный момент оценку оптимума, то есть, тем больше «рысканье» алгоритма по области определения. По мере работы алгоритма температура понижается и «шатания» прекращаются.

В процессе работы алгоритма величина T постепенно уменьшается от некоторого начального значения T_0 до конечного T_{fin}. Можно для определённости считать последовательность значений T геометрической прогрессией с некоторым знаменателем α, $0 < \alpha < 1$, хотя в общем случае выбор стратегии уменьшения T, обеспечивающей сходимость метода к глобальному оптимуму, является, строго говоря, не столь простым [1]. При этом вероятность дробле-

ния выбранного бруса $\boldsymbol{z}$, аналогичная «принятию» очередного приближения в классическом методе «имитации отжига», задаётся формулой

$$\mathsf{P}_T(\boldsymbol{y},\boldsymbol{z}) \;=\; \begin{cases} 1, & \text{если} \quad \Delta\boldsymbol{f} \leqslant 0, \\ \exp\left(-\dfrac{\Delta\boldsymbol{f}}{kT}\right), & \text{если} \quad \Delta\boldsymbol{f} \geqslant 0, \end{cases} \tag{3.6}$$

где $\Delta\boldsymbol{f} = \underline{\boldsymbol{f}(\boldsymbol{z})} - \underline{\boldsymbol{f}(\boldsymbol{y})}$ — приращение оценки оптимума, обеспечиваемое брусом нового приближения. Семейство графиков этой функции для различных температур T приведено на рис. 3.9.

Правило $\mathcal{S}(\boldsymbol{y})$ в алгоритме табл. 3.1 — это случайный выбор бруса $\boldsymbol{z}$ из рабочего списка. Оно зависит от ведущего бруса $\boldsymbol{y}$ (а также ведущей оценки), и может быть организовано самым различным образом в зависимости от наличия априорной информации о целевой функции.

Отметим, что помимо процедуры табл. 3.1 интервальная имитация отжига может быть представлена ещё «доказательным» вариантом, в котором по ходу исполнения алгоритма сохраняется свойство вычисляемой оценки не превосходить глобальный оптимум. В чём же смысл введения элементов стохастики в такую вычислительную схему? Тем самым мы рандомизируем её, обеспечивая в среднем более равномерное приложение вычислительных усилий на ранних этапах работы алгоритма. Частично это способствует и преодолению эффекта «застаивания» интервальной оценки.

На рис. 3.9, изображающем графики функций $\mathsf{P}_T(\boldsymbol{y},\boldsymbol{z})$, область отрицательных $\Delta\boldsymbol{f}$ недоступна для доказательной версии «интервальной имитации отжига». Действительно, если $\boldsymbol{y}$ — ведущий на данном шаге алгоритма брус, то теперь $\underline{\boldsymbol{f}(\boldsymbol{y})}$ не больше оценки по любому другому брусу списка, и $\Delta\boldsymbol{f}$ всегда неотрицательна.

Интервальный алгоритм имитации отжига имеет важное преимущество перед своим классическим (точечным) вариантом. В отличие от классического алгоритма моделированного отжига, который обеспечивает нахождение глобального оптимума лишь в вероятностном смысле, результатом работы нашего интервального алгоритма имитации отжига будет гарантированная оценка глобального оптимума. Более того, этот глобальный оптимум будет найден независимо от способа изменения температуры T. Классическая точечная версия, запущенная для функции с множеством локальных оптимумов, может возвращать различные результаты в зависимости от способа изменения «температуры» [128].

Сформулируем сделанные утверждения более формально и докажем их.

Утверждение 3.1. В процессе работы интервального алгоритма имитации

Таблица 3.1: Простейший интервальный алгоритм «имитации отжига»

Вход

Брус $\boldsymbol{x} \subset \mathbb{R}^n$. Начальное T_0 и конечное T_{fin} значения «температуры». Интервальное расширение $\boldsymbol{f} : \mathbb{I}\boldsymbol{x} \to \mathbb{IR}$ целевой функции f.

Выход

Оценка f^* глобального минимума функции f на $\boldsymbol{x}$.

Алгоритм

присваиваем $\boldsymbol{y} \leftarrow \boldsymbol{x}$ и $T \leftarrow T_0$;
назначаем целочисленную величину N_T
— «количество испытаний на один температурный уровень» ;
вычисляем $\boldsymbol{f}(\boldsymbol{y})$ и инициализируем список $\mathcal{L}$ записью $\{ (\boldsymbol{y}, \underline{\boldsymbol{f}(\boldsymbol{y})}) \}$;
DO WHILE $(T > T_{fin})$
  DO FOR $j = 1$ TO N_T
    случайно выбираем из $\mathcal{L}$ запись $(\boldsymbol{z}, \underline{\boldsymbol{f}(\boldsymbol{z})})$ по правилу $\mathcal{S}(\boldsymbol{y})$;
    DO (с вероятностью $\mathsf{P}_T(\boldsymbol{y}, \boldsymbol{z})$)
      рассекаем $\boldsymbol{z}$ по самой длинной компоненте пополам
        на брусы-потомки $\boldsymbol{z}'$ и $\boldsymbol{z}''$;
      вычисляем $\boldsymbol{f}(\boldsymbol{z}')$ и $\boldsymbol{f}(\boldsymbol{z}'')$;
      удаляем запись $(\boldsymbol{z}, \underline{\boldsymbol{f}(\boldsymbol{z})})$ из списка $\mathcal{L}$;
      помещаем записи $(\boldsymbol{z}', \underline{\boldsymbol{f}(\boldsymbol{z}')})$ и $(\boldsymbol{z}'', \underline{\boldsymbol{f}(\boldsymbol{z}'')})$ в список $\mathcal{L}$;
      обозначаем через $(\boldsymbol{y}, \underline{\boldsymbol{f}(\boldsymbol{y})})$ ту из записей $(\boldsymbol{z}', \underline{\boldsymbol{f}(\boldsymbol{z}')})$
        и $(\boldsymbol{z}'', \underline{\boldsymbol{f}(\boldsymbol{z}'')})$, которая имеет меньшее значение
        второго поля ;
    END DO
  END DO
  уменьшаем значение температуры $T \leftarrow \alpha T$
END DO
$f^* \leftarrow \underline{\boldsymbol{f}(\boldsymbol{y})}$;

отжига из табл. 3.2 при любом законе предельного перехода $T \to 0$ существуют такие натуральные числа i_{max} и n, что любой брус $\boldsymbol{x}'$ из рабочего списка, удовлетворяющий равенству $\underline{\boldsymbol{f}(\boldsymbol{x}')} = \min_{1 \le i \le S} \underline{\boldsymbol{f}(\boldsymbol{x}_i)}$, начиная с шага с номером $i_{\max}$, подвергается дроблению не реже чем одного раза в продолжение любых n последовательных шагов алгоритма.

Доказательство. Интервальный алгоритм имитации отжига на каждой итерации разбивает очередной брус $\boldsymbol{x}$ на более мелкие подбрусы $\boldsymbol{x}_1$ и $\boldsymbol{x}_2$, такие

Таблица 3.2: Доказательная версия интервального алгоритма имитации отжига

Вход

Брус $\boldsymbol{x} \subset \mathbb{R}^n$. Заданная точность $\epsilon > 0$.
Интервальное расширение $\boldsymbol{f} : \mathbb{I}\boldsymbol{x} \to \mathbb{IR}$ целевой функции f.
Начальное T_0 и конечное T_{fin} значения «температуры».

Выход

Оценка снизу f^* глобального минимума функции f на $\boldsymbol{x}$.

Алгоритм

присваиваем $\boldsymbol{y} \leftarrow \boldsymbol{x}$ и $T \leftarrow T_0$;
назначаем целочисленную величину N_T
— «количество испытаний на один температурный уровень»;
вычисляем $\boldsymbol{f}(\boldsymbol{y})$ и инициализируем список $\mathcal{L}$ записью $\{ (\boldsymbol{y}, \underline{\boldsymbol{f}(\boldsymbol{y})}) \}$;
DO WHILE $\left(T > T_{fin} \text{ и } \operatorname{wid}(\boldsymbol{f}(\boldsymbol{y})) \geq \epsilon \right)$
DO FOR $j = 1$ TO N_T
случайно выбираем из $\mathcal{L}$ запись $(\boldsymbol{z}, \underline{\boldsymbol{f}(\boldsymbol{z})})$ по правилу $\mathcal{S}(\boldsymbol{y})$;
DO (с вероятностью $\mathsf{P}_T(\boldsymbol{y}, \boldsymbol{z})$)
рассекаем $\boldsymbol{z}$ по самой длинной компоненте пополам на брусы $\boldsymbol{z}'$ и $\boldsymbol{z}''$;
вычисляем $\boldsymbol{f}(\boldsymbol{z}')$ и $\boldsymbol{f}(\boldsymbol{z}'')$;
удаляем запись $(\boldsymbol{z}, \underline{\boldsymbol{f}(\boldsymbol{z})})$ из списка $\mathcal{L}$;
помещаем записи $(\boldsymbol{z}', \underline{\boldsymbol{f}(\boldsymbol{z}')})$ и $(\boldsymbol{z}'', \underline{\boldsymbol{f}(\boldsymbol{z}'')})$ в список $\mathcal{L}$ в порядке возрастания второго поля;
END DO
обозначаем ведущую запись списка $\mathcal{L}$ через $(\boldsymbol{y}, \underline{\boldsymbol{f}(\boldsymbol{y})})$
END DO
уменьшаем значение температуры $T \leftarrow \alpha T$
END DO
$f^* \leftarrow \underline{\boldsymbol{f}(\boldsymbol{y})}$;

что

$$\boldsymbol{x} = \boldsymbol{x}_1 \cup \boldsymbol{x}_2. \tag{3.7}$$

При этом из свойства (2.3) следует, что

$$\begin{gathered} \boldsymbol{f}(\boldsymbol{x}) \supseteq \boldsymbol{f}(\boldsymbol{x}_1) \\ \text{и} \\ \boldsymbol{f}(\boldsymbol{x}) \supseteq \boldsymbol{f}(\boldsymbol{x}_2). \end{gathered} \tag{3.8}$$

Выбор бруса для дробления из рабочего списка осуществляется псевдослучайным образом. Но выбранный брус будет отвергнут, если он не удовлетворяет соотношению (3.6). Исходя из этого, вероятность принятия какого-то конкретного бруса $\boldsymbol{x}$ может быть представлена как функция $\mathsf{P}_{\boldsymbol{x}}(T, \boldsymbol{f}(\boldsymbol{x}))$, зависящая от «температуры» T и оценки значений целевой функции на этом брусе. При этом для $T = +\infty$ абсолютно любой брус удовлетворяет соотношению (3.6) и, фактически, рассматриваемый алгоритм представляет собой реализацию метода случайного интервального дробления, описанного ранее. Иными словами $\mathsf{P}_{\boldsymbol{x}}(+\infty, \boldsymbol{f}(\boldsymbol{x})) = 1/n$, где n — размер рабочего списка, и совершенно не зависит от $\boldsymbol{f}(\boldsymbol{x})$. В то же время, для $T = 0$ может быть раздроблен только брус, дающий рекордную на данном шаге оценку. Таким образом, в этом случае реализуется вычислительная схема интервального адаптивного дробления. Если через S обозначить размер рабочего списка $\mathcal{L}$, то вероятность того, что интервальная оценка будет уточнена на брусе $\boldsymbol{x}$, выглядит следующим образом:

$$\mathsf{P}_{\boldsymbol{x}}(0, \boldsymbol{f}(\boldsymbol{x})) = \begin{cases} 1, \text{ если } \underline{\boldsymbol{f}(\boldsymbol{x})} = \min_{1 \le i \le S} \underline{\boldsymbol{f}(\boldsymbol{x}_i)}, \\ 0, \text{ иначе.} \end{cases}$$

Получается, что при значении T, меньшем какого-то порогового T_1 (зависящего от способа вычисления интервальной оценки и конфигурации целевой функции), интервальный алгоритм имитации отжига является реализацией алгоритма случайного интервального дробления с приоритетом (для которого ранее в тексте доказывается теорема о его сходимости к глобальному оптимуму почти всегда). С другой стороны, при $T \to 0$ наш алгоритм сколь угодно мало отличается от детерминистской вычислительной схемы адаптивного интервального дробления. Следовательно, можно утверждать, что если $T \to 0$, то при количестве итераций i, превышающем некоторое значение i^*, зависящее от целевой функции, способа вычисления её интервального расширения и т. п., в рабочем списке в течении нескольких итераций не сможет находиться такой брус $\boldsymbol{x}_0$, что $\underline{\boldsymbol{f}(\boldsymbol{x}_0)} = \min_{1 \le j \le S} \underline{\boldsymbol{f}(\boldsymbol{x}_j)}$ нераздробленным. Иными словами, для двух шагов алгоритма i_1 и i_2, таких, что $i_1 + n = i_2$, имеют место следующие соотношения

$$\min_{1 \le j \le S_{i_1}} \underline{\boldsymbol{f}(\boldsymbol{x}_j)} = \underline{\boldsymbol{f}(\boldsymbol{x}_0^{i_1})} \qquad \text{и} \qquad \min_{1 \le j \le S_{i_2}} \underline{\boldsymbol{f}(\boldsymbol{x}_j)} = \underline{\boldsymbol{f}(\boldsymbol{x}_0^{i_2})},$$

$$\text{причём } \boldsymbol{x}_0^{i_1} \neq \boldsymbol{x}_0^{i_2},$$

где n зависит от i_1 и способа стремления T к нулю. При этом чем больше

номер i_1, тем меньше n, а начиная с какого-то шага можно утверждать, что $n = 1$.

Теорема 3.2 *Интервальный алгоритм имитации отжига, представленный в табл. 3.2, сходится к глобальному оптимуму.*

Доказательство. Для любых шагов алгоритма q и w, таких что $q \leqslant w$,

$$\frac{\boldsymbol{x}_1^q + \boldsymbol{x}_2^q + \ldots + \boldsymbol{x}_n^q}{S_q} \geqslant \frac{\boldsymbol{x}_1^w + \boldsymbol{x}_2^w + \ldots + \boldsymbol{x}_m^w}{S_w}, \tag{3.9}$$

где S_q и S_w — размер рабочего списка брусов на q-ой и w-ой итерациях соответственно.

Из ранее доказанного Утверждения 3.1 следует, что не может существовать брус $\boldsymbol{x}_0$, являющийся ведущим, который при этом остаётся нераздробленным всё время работы алгоритма. С учётом этого факта, а также из (3.8) и (3.9), вытекает, что

$$\begin{gathered} \min_{1 \leqslant i \leqslant S_q} \underline{\boldsymbol{f}(\boldsymbol{x}_i)} \leqslant \min_{1 \leqslant i \leqslant S_w} \underline{\boldsymbol{f}(\boldsymbol{x}_i)} \\ \text{и} \\ \max_{1 \leqslant i \leqslant S_q} \overline{\boldsymbol{f}(\boldsymbol{x}_i)} \geqslant \max_{1 \leqslant i \leqslant S_w} \overline{\boldsymbol{f}(\boldsymbol{x}_i)}. \end{gathered} \tag{3.10}$$

Как следствие, алгоритм сходится именно к глобальному оптимуму.

Нами был проведён ряд вычислительных экспериментов с интервальным алгоритмом имитации отжига [47, 54]. Тестовые задачи брались, в основном, из [90]. Эксперименты показали преимущество нового метода в скорости нахождения глобального оптимума над классическим (детерминированным) алгоритмом интервальной глобальной оптимизации, в случае, если целевая функция f имела много локальных экстремумов и сложный рельеф. Если же f имела не очень сложную структуру, то разработанный метод заметно проигрывал детерминистскому.

Так, на рис. 3.10 продемонстрировано сравнение работы интервального алгоритма имитации отжига с работой классического алгоритма интервального адаптивного дробления на целевой функции (1.41) «Шестигорбый верблюд» при отключенных процедурах отбраковки. Хорошо видно, что метод имитации отжига нашёл глобальный оптимум с необходимой точностью на порядок быстрее. В то же время, применение этого метода с теми же настройками, что и в предыдущем случае, для поиска глобального минимума функции Растригина (1.42) дало чрезвычайно плохие результаты (см. рис. 3.11).

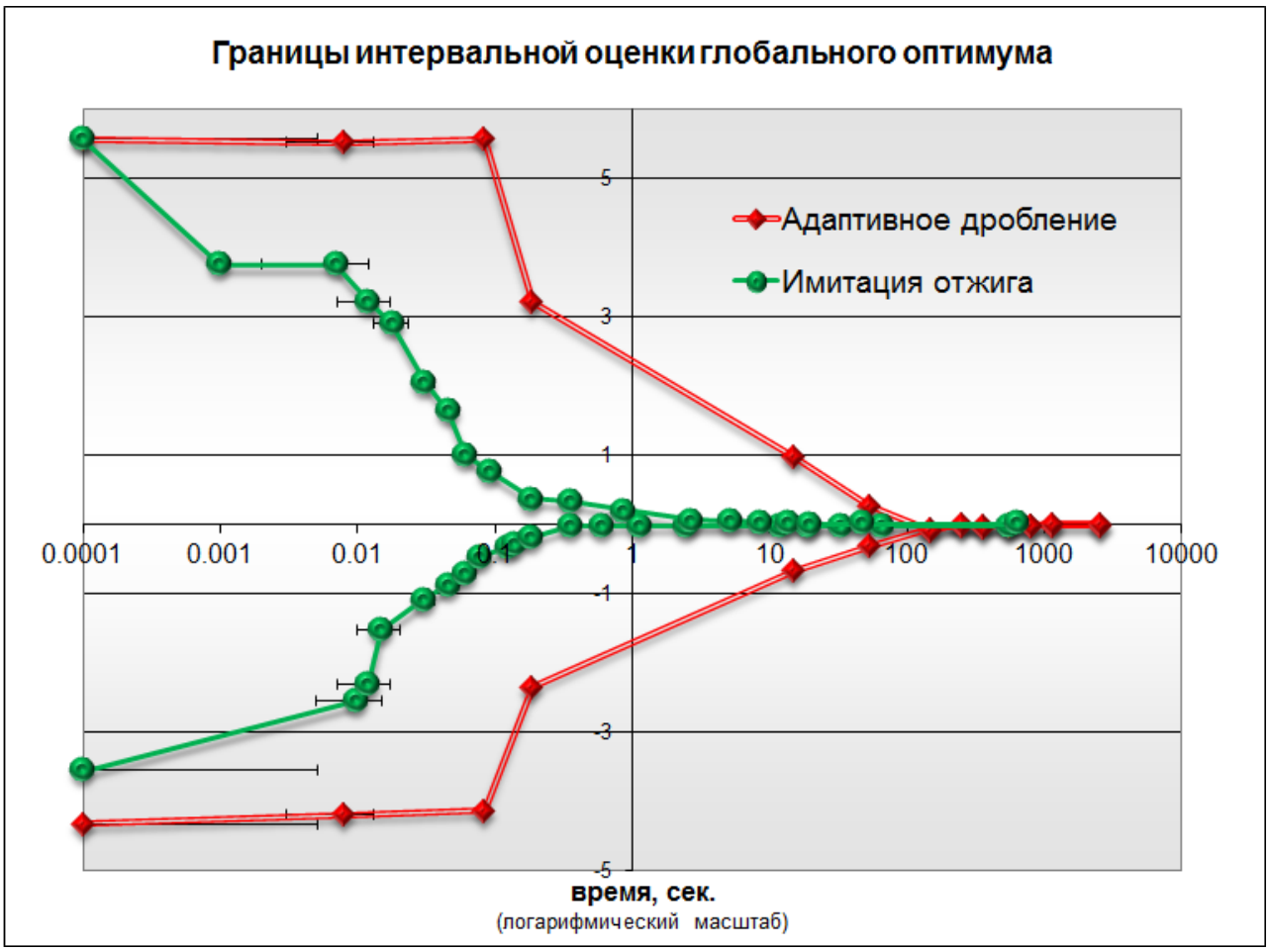


Рис. 3.10: Ширина оценивания оптимума интервальными алгоритмами имитации отжига и адаптивного дробления в зависимости от времени работы

Объяснение явления даётся на рис. 3.12, на котором проиллюстрирован непосредственно процесс дробления. Напомним, что при всей многоэкстремальности, неудобности и сложности для глобальной оптимизации функции Растригина 10 (её вид вблизи начала координат приводится на стр. 99), адаптивное интервальное дробление справляется с ней отлично. Можно вернуться к рис. 2.14 на стр. 103, на котором показано, как, ни разу не ошибаясь, метод адаптивного интервального дробления двигается к цели. Что касается метода «имитации отжига», то в нём заложено рысканье по области определения, что вместе с неоптимальными настройками приводит к метаниям и многочисленным бессмысленным дроблениям (см. рис. 3.12).

Мы привели этот пример, чтобы ещё раз подчеркнуть, что поведение алгоритма существенным образом зависит от параметра T — температуры, причём как от начального его значения, так и способа его изменения со временем. В классическом (точечном) варианте недостаточно высокая начальная

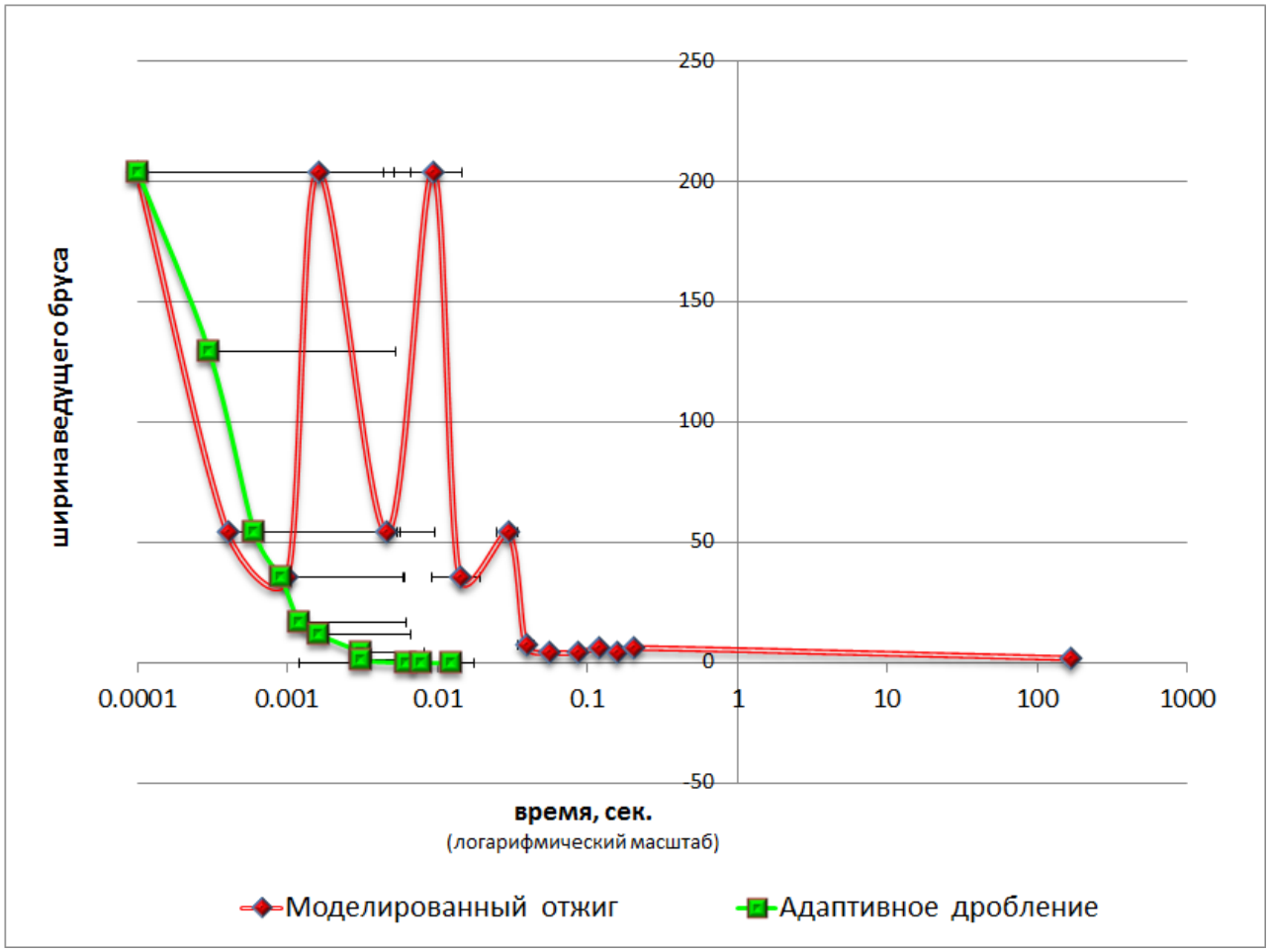


Рис. 3.11: Ширина интервальной оценки на ведущем брусе в зависимости от времени для алгоритма имитации отжига с неудачными настройками и классической интервальной бисекции при поиске оптимума десятой функции Растригина

температура или слишком быстрое охлаждение повышают вероятность того, что алгоритм сможет найти лишь локальный оптимум. В интервальном случае всё несколько лучше. Если система «охлаждается» очень быстро, то мы получаем практически обычный алгоритм адаптивного интервального дробления. В противном случае, если температура высока и почти не понижается, то получающийся алгоритм чрезвычайно похож на алгоритм случайного интервального дробления.

Для разных типов целевых функций разные стратегии изменения температуры дают лучшие результаты. Мы разработали стратегию управления «температурой», достаточно хорошо справляющуюся с различными целевыми функциями в отсутствие априорной информации об их характере. Этот способ неплохо зарекомендовал себя на различных типах задач и может использоваться как универсальный.

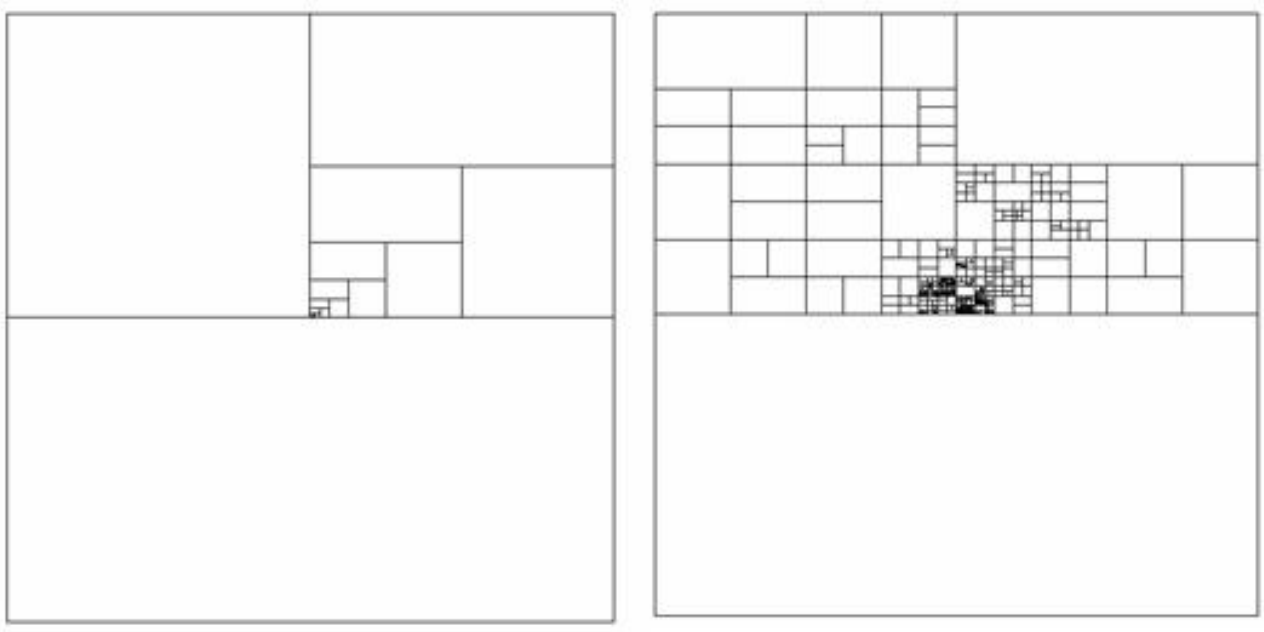


Рис. 3.12: Процесс поиска минимума функции «R10».
Слева: дробление методом адаптивного интервального дробления.
Планомерное и безошибочное движение к цели.
Справа: неудачные настройки метода имитации отжига
провоцируют множество разрозненных дроблений

Вначале температура назначается весьма высокой, но быстро снижается до тех пор, пока доля отвергнутых неведущих брусов не достигает значения примерно 20 процентов. Затем температура достаточно плавно понижается до тех пор, пока не будет отвергаться более половины брусов, после чего скорость охлаждения опять увеличивается. Если после понижения температуры до нуля ответ не был получен алгоритмом за какое-то разумное время, мы рекомендуем заново «подогреть» систему.

3.4 Интервальный генетический алгоритм

Следующим этапом нашей работы стало обобщение для интервального случая алгоритмов «биологического» типа, ещё называемых иногда «бионическими». Это эвристические алгоритмы поиска, так или иначе использующие для решения задач оптимизации случайный подбор, комбинирование и вариации входных параметров на основе механизмов, напоминающих биологическую эволюцию. Идеи применить биологические механизмы или, скорее, принципы их функционирования для создания новых алгоритмов возникли с небольшими вариациями практически одновременно у нескольких авторов. В 1966 году Л. Фогель, А. Оуэнс, М. Уолш предложили схему эволюции

логических автоматов, решающих задачи прогноза [68]. В 1975 году вышла основополагающая книга Дж. Холланда «Адаптация в естественных и искусственных системах» [95], в которой был предложен законченный вариант генетического алгоритма. Примерно в это же время немецкие ученые Рехенберг и Швефель начали разработку так называемой эволюционной стратегии [120, 122]. Эти работы заложили основы прикладного эволюционного моделирования или эволюционных алгоритмов.

Уже первые работы показали, что идея использовать принципы биологической эволюции оказалась плодотворной. В настоящее время множество разновидностей биологических алгоритмов используются в том числе и для решения задачи глобальной оптимизации [13, 99].

Алгоритмы данного типа можно условно разделить на

- *эволюционные*,
- *поведенческие*,
- и *генетические*[1].

В этой классификации для генетических алгоритмов характерно следующее. Выделяются ключевые для решения задачи свойства объектов, их признаки кодируются в «гены», которые, в свою очередь, объединяются в «генотип». Один из самых простых вариантов формирования генотипа — кодирование ключевых признаков в виде набора битовых массивов, достаточной длины для представления всего набора возможных значений каждого признака. Затем, изменяя по одному биту (по одной «*хромосоме*»), оценивать как подобные изменения сказываются на признаках объекта в целом и селектировать лучшие особи.

Поведенческие алгоритмы случайного поиска моделируют поведение живых организмов в неизвестной среде, когда имитируется присущая живым организмам способность стремиться к комфортным условиям и уклоняться от некомфортных. Для целей интервальной глобальной оптимизации нам показался наиболее удобным эволюционный подход. Опишем его подробнее.

Общее описание алгоритма

Каким-либо образом, чаще всего случайно, задается начальная «*популяция*» — некое множество объектов. Они оцениваются с использованием «*функции приспособленности*», в результате чего каждому объекту присваивает-

[1] Часто слово «генетический» используется для обозначения любого алгоритма биологического типа

ся определённое значение (*приспособленность*), которое определяет вероятность выживания организма, представленного данным объектом. После этого с использованием полученных значений приспособленности выбираются объекты, допущенные к «*размножению*» (производится *селекция*). Вообще говоря, к объектам могут применяться и «*генетические операторы*» (в большинстве случаев это «*скрещивание*» — *crossover* и «*мутация*» — *mutation*). Так или иначе, каким-то образом создаётся следующее «*поколение*», особи которого также оцениваются, затем снова производится селекция, формируется следующее поколение и т. д. Описанным образом моделируется «*эволюционный процесс*», продолжающийся несколько жизненных циклов (*поколений*) до тех пор, пока не будет выполнен критерий останова алгоритма. Таким критерием может быть 1) исчерпание количества поколений, отпущенных на эволюцию, 2) исчерпание времени, отпущенного на эволюцию, 3) достижение минимального размера ведущего бруса (сугубо интервальный критерий).

На Листинг 8 приведены основные этапы эволюционного алгоритма.

Листинг 8. Эволюционный алгоритм.

1. Создание начальной популяции
2. Вычисление функций приспособленности для особей популяции (оценивание)
   (Начало цикла)
   (a) Выбор индивидов из текущей популяции (селекция)
   (b) Формирование нового поколения
   (c) Разнообразие поколения за счёт мутаций
   (d) Вычисление функций полезности для всех особей
   (e) Если достигнуто условие останова, то решение найдено (конец цикла), если нет, то цикл повторяется.
   (Конец цикла)

Рассмотрим теперь отдельные этапы алгоритма подробнее.

Отбор

На стадии отбора нужно из всей популяции выбрать определённую её долю, которая является «доминантой» на этом этапе эволюции. Есть разные

способы проводить отбор, но в любом случае вероятность выживания особи h должна зависеть от значения функции приспособленности. При этом количество (доля) наиболее приспособленных s обычно является параметром алгоритма. В классических (точечных) реализациях часто большинство из особей не прошедших отбор (иными словами — недостаточно приспособленных) гибнут, т. е. исключаются из популяции на следующем шаге. По итогам отбора из N особей популяции H должны быть отобраны sN особей, которые выживут и дадут потомство в следующую популяцию H'.

Размножение

Размножение в разных алгоритмах определяется по-разному. Как правило, в современных точечных генетических алгоритмах преобладает «половое размножение»: чтобы произвести потомка, нужны несколько родителей (как правило два). Главное требование к размножению — потомок или потомки должны иметь возможность унаследовать черты родителей, «смешав» их каким-либо достаточно разумным способом. Вообще говоря, для того, чтобы провести такую операцию размножения, нужно выбрать $(1-s)p/2$ пар особей из H и провести с ними размножение, получив по два потомка от каждой пары (если размножение определено так, чтобы давать одного потомка, нужно выбрать $(1-s)p$ пар), и добавить этих потомков в H'. В результате H' будет состоять из N особей. В классических (точечных) реализациях обычно особи для размножения выбираются из всей популяции H, а не из доминант, хотя последний вариант тоже имеет право на существование. Это делается для того, что бы предотвратить недостаток разнообразия в особях. Иначе, достаточно быстро выделяется один генотип, который представляет собой локальный максимум, а затем все элементы популяции проигрывают ему отбор, и вся популяция наполняется копиями этой особи. Есть разные способы борьбы с этим нежелательным эффектом. Один из них — выбор для размножения не самых приспособленных, но вообще всех особей. Предлагаемая интервальная версия лишена этого недостатка (т. е. всегда находятся двусторонние оценки именно глобального оптимума), тем не менее алгоритм предусматривает возможность размножения и нескольких особей, признанных «неприспособленными».

Мутации

Для этапа «мутаций» классического генетического алгоритма справедливо многое из уже описанного для шага «размножения». Существует некоторая

доля мутантов m, являющаяся параметром алгоритма, и шаге мутаций нужно выбрать $m \cdot N$ особей, а затем изменить их в соответствии с заранее определенными операциями мутации. Заметим, что предлагаемый нами интервальный алгоритм не производит мутаций.

В качестве иллюстрации всего вышесказанного, на листинге 9 приведен пример простого варианта алгоритма генетического поиска.

Листинг 9. Пример тривиального генетического алгоритма со сменой поколений но без скрещивания. Для наглядности в качестве меры приспособленности выбрано числовое значение, хранимое каждой «особью».

```
int a[N];

...

for (;;)
{
    // мутация в случайную сторону каждого десятого элемента:
    for (int i = 0; i < N; i+=10)
        if (rand()%2 == 1)
            a[i] += 1;
        else
            a[i] -= 1;
    // сортируем популяцию по возрастанию.
    sort(a, a+N);
    // таким образом, самые приспособленные оказываются
    // в конце массива.

    // скопируем лучших в первую половину
    copy(a+N/2, a+N, a);
    // иными словами неприспособленные особи погибли,
    // на их место пришло потомство лучших особей

    // посмотрим среднее состояние популяции:
    cout << accumulate(a, a+N, 0) / N << endl;
    // с каждой итерацией оно всё лучше.

}
```

Для целей доказательной (гарантированной) глобальной оптимизации с

использованием инструментария интервального анализа достаточно лишь немного изменить описанную схему (см. Листинг 10).

В предлагаемом интервальном эволюционном алгоритме исходная область поиска по-прежнему разбивается на непересекающиеся подобласти — брусы. Для обеспечения гарантированности результатов мы не отбрасываем из рассмотрения никакую подобласть области поиска, пока не будет показано, что она гарантированно не может содержать оптимум. Что означает, что в отличии от большинства точечных реализаций, в предлагаемом алгоритме в популяции присутствуют как молодое потомство, так и все неприспособленные особи предыдущего поколения. Исключение особи из рассмотрения (гибель, в терминах генетических алгоритмов) происходит лишь в двух случаях: особь либо рождаются нежизнеспособной (не выдерживает интервальных критериев отбраковки, применяемых сразу после дробления) или особь погибает в результате «*эпидемии*» — срабатывания уточненного критерия отбраковки по значению при дальнейшем исполнении алгоритма.

В качестве меры приспособленности особи можно выбрать нижнюю границу (для определённости рассматриваем поиск минимума) интервальной оценки целевой функции на соответствующем брусе. Чем лучше мера приспособленности (чем меньше нижняя граница в привычных терминах), тем больше шансов у данной особи размножиться (данному брусу быть раздробленным) и тем многочисленнее будет потомство (брус может быть раздроблен на большее количество подбрусов).
В нашей реализации этой модели (см. Листинг 10) есть несколько коэффициентов:

— вероятности P_n и P_m;
— максимальные количества потомков L_n и L_m;
— минимальное количество потомков U_n и U_m;
— величина N — сколько объектов, начиная с самого приспособленного могут оставить потомство;
— брусы дробятся на равные части, или разбиение происходит в случайной пропорции, «разновесные» дети;
— количество «особей» в начальной популяции.

В качестве дополнительного пояснения на листинге 11 приведён псевдокод этого алгоритма. Как можно видеть из него, в модели есть несколько «подстроечных параметров»:

Листинг 10. Основа интервального генетического алгоритма.

Произвольным образом задаётся начальная популяция

(Как и в классическом случае, при запуске работы алгоритма каким-либо способом требуется задать начальную популяцию. Чаще всего она задается случайно, так как даже если созданная популяция окажется совершенно неконкурентоспособной, генетический алгоритм всё равно достаточно быстро переведет её в жизнеспособную популяцию. Этот шаг вообще можно пропустить, приняв брус исходной области поиска за единственную существующую особь или произвести несколько случайных дроблений исходной области определения.)

Начальная популяция передаётся в основной цикл

Основной цикл

{

1. **Вычислить значения функции приспособленности новорожденных особей**. (Этот шаг включает вычисление интервального расширения целевой функции по новым подбрусам, как необходимый для определения приспособленности подбруса)
2. N **из наиболее приспособленных брусов с вероятностью** P_n **оставляют от** L_n **до** U_n **потомков**
3. M **из неприспособленных брусов с вероятностью** P_m **оставят от** L_m **до** U_m **потомков**
4. **Потомки проверяются на жизнеспособность** (применяются интервальные критерии отбраковки, описанные ранее)
5. **Если критерий отбраковки был улучшен, возможно случается эпидемия** (улучшенные критерии применяются ко всем особям)

}

Листинг 11. Псевдокод интервального генетического алгоритма (в стиле ООП)

```
class CGeneticAlgorithm : public CAlgorithm {

    ...

// параметры алгоритма
protected:

        // количество "особей"  в начальной популяции
        unsigned _initialHeadCount;

        // максимальное количество потомков
        unsigned _maxProlificacy;
        // минимальное количество потомков
        unsigned _minProlificacy;

        // сколько брусов, начиная с самого
        //  приспособленного, могут оставить потомство
        unsigned _energyBarrier;

        ...

        // создание случайной популяции
        // вообще говоря, без этого можно и обойтись
        virtual void init() throw (CInitException) {

                ...

                CBox *pWorkBox = 0;

                // исходная область определения дробится на
                // _initialHeadCount брусов случайным образом
                for (int i = 0; i < _initialHeadCount/2; i++) {
                        pWorkBox = getRandomBox();
                        addBox(pWorkBox->split());
                }
                ...
        }

        // основной цикл
        virtual void iteration() throw (CIterationException) {

                ...

                // из общей "популяции" выбираются лучшие
                box_container* pKindergarten = selectBest();
```

```
\small{
                // они дают потомство
                pKindergarten->splitWithProlifacy();

                // применение критериев отбраковки:
                // проверка "жизнеспособности"
                pKindergarten->applayRestrictions();

                // "молодые" особи приходят в популяцию
                addBox(pKindergarten);

                ...

        }

    ...

};
}
```

```
    _maxProlificacy;  // максимальное количество потомков
    _minProlificacy;  // минимальное количество потомков
    _energyBarrier;   // сколько объектов, начиная с самого
                      //  приспособленного могут оставить потомство
    _isKidsEqual;     // true  = брусы раздрабливаются на равные части,
                      //  false = разбиение происходит в случайной
                      //  пропорции, <<разновесные>> дети
```

При этом поведение алгоритма и успешность его работы зависят от этих параметров весьма существенно.

Заметим, что если скорректировать условия таким образом, чтобы потомки были равноправны и их было обязательно двое, но оставить потомство мог лишь «вожак», то есть самый приспособленный. В более привычных терминах это будет звучать как «на каждой итерации дробится брус с наименьшей нижней оценкой». В этом случае реализуется детерминистская вычислительная схема классического интервального адаптивного дробления, в основе которой лежит "метод ветвей и границ предложенный Муром, дополненный Скелбоу, обогащенный Хансеном и уже много раз обсуждавшийся в этом тексте.

В качестве меры приспособленности в алгоритме мы использовали нижнюю границу интервальной оценки функции на подбрусе (оценку оптимума

на подбрусе снизу). С тем же успехом можно опираться и на ширину интервальной оценки или же на размер бруса.

Несмотря на то, что все три метода очень похожи, и критерии, которыми они руководствуются, направлены на достижение одной и той же цели, работают они по-разному. Таким образом, следующим логичным шагом должно стать объединение этих методов в один, использующий преимущества каждого.

Блок-схема объединенного алгоритма показана на рис. 3.13. Функция приспособленности бруса (при поиске минимума) приобрела следующий вид:

$$\boldsymbol{F}(\boldsymbol{b}) = \sum_{i=1}^{n} \left(\alpha_i \cdot \operatorname{wid} \boldsymbol{b}_i \right) + \beta \cdot \underline{\boldsymbol{f}(\boldsymbol{b})} + \gamma \cdot \operatorname{wid} \boldsymbol{f}(\boldsymbol{b}). \tag{3.11}$$

Здесь

$\boldsymbol{f}(\boldsymbol{b})$ — интервальная оценка целевой функции f на брусе $\boldsymbol{b}$;

$\underline{\boldsymbol{f}(\boldsymbol{b})}$ — нижняя граница интервальной оценки (оценка минимума снизу);

$\operatorname{wid} \boldsymbol{f}(\boldsymbol{b})$ — ширина (точность) интервальной оценки;

$\operatorname{wid} \boldsymbol{b}_i$ — ширина i-ой стороны бруса;

$\alpha_i, \beta_i, \gamma_i$ — соответствующие весовые коэффициенты.

Необходимо заметить, что поведение алгоритма очень сильно зависит от величин коэффициентов.

В заключение сформулируем и докажем, что интервальный генетический алгоритм сходится к глобальному оптимуму.

Теорема 3.3 *Интервальный генетический алгоритм сходится к глобальному оптимуму.*

Доказательство. Если мы сможем показать, что в процессе работы рассматриваемого алгоритма

1) глобальный оптимум не может быть пропущен и

2) границы интервальной оценки оптимума уточняются,

то мы докажем искомое утверждение. Таким образом, доказательство теоремы 3.3 распадается на доказательство двух утверждений.

Утверждение 3.2. В ходе работы алгоритма границы интервальной оценки оптимума монотонно сужаются.

Доказательство. Алгоритм принимает целевую функцию $f(x)$ и область поиска $\boldsymbol{x}$ в качестве входных аргументов. Таким образом, исходные границы

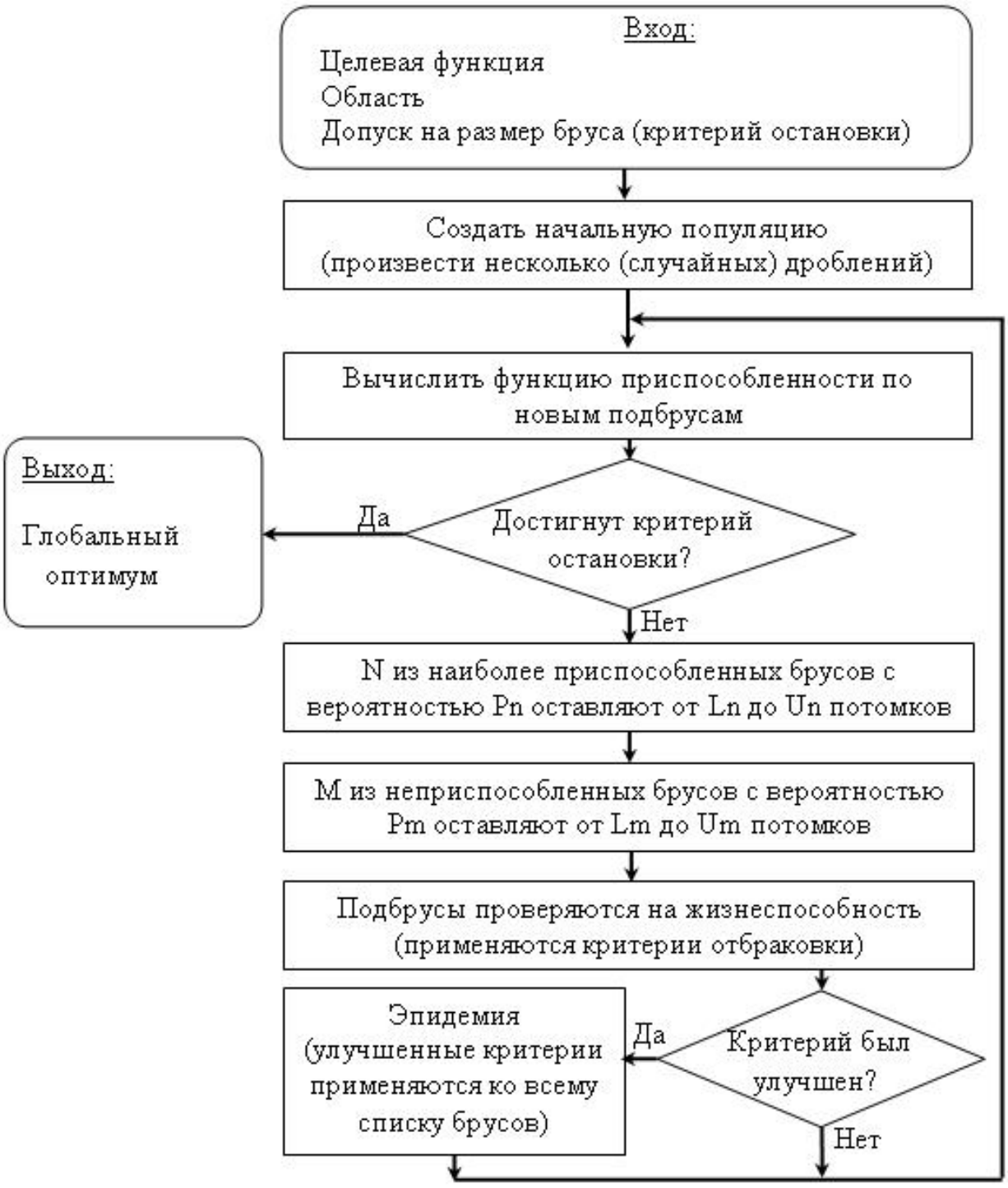


Рис. 3.13: Блок-схема интервального генетического алгоритма

оптимума это $[\underline{\boldsymbol{f}(\boldsymbol{x})}, \overline{\boldsymbol{f}(\boldsymbol{x})}] = \boldsymbol{E}^{(0)}$. Как было показано ранее, эти границы могут быть сколь угодно избыточными.

На каждой итерации алгоритма дробится не менее одного бруса (не менее чем одна особь рождает потомство). Что означает, что мы сможем пронумеровать все брусы в рабочем списке (особи в популяции), так, что для любых двух последовательных шагов алгоритма n и $n+1$ найдётся какое-то q и такие $w > 0$ и $z > 0$, что будет, во-первых, верно следующее соотношение:

$$\boldsymbol{f}(\boldsymbol{x}_i^n) = \boldsymbol{f}(\boldsymbol{x}_i^{n+1}), \quad \text{для } 1 \leqslant i \leqslant q,$$

где через $\boldsymbol{x}_i^n$ и $\boldsymbol{x}_i^{n+1}$ обозначены соответствующие особи n и $n+1$-го шага. И, во-вторых, для каждого $\boldsymbol{f}(\boldsymbol{x}_i^n)$, где $1 \leqslant i \leqslant w$, найдётся от 1 до z таких брусов, что каждый из них будем меньше исходного: $\exists i^* : \boldsymbol{f}(\boldsymbol{x}_{i^*}^n) \leqslant \forall_{i=1}^z \boldsymbol{f}(\boldsymbol{x}_i^{n+1})$. При этом $q+w = S_n$, и $q+z = S_{n+1}$, где через S обозначен размер рабочего списка на соответствующей итерации.

Оценка глобального оптимума (для определённости минимума без ограничения общности) на $n-$ой итерации алгоритма может быть выражена следующим образом. $\boldsymbol{E}^{(n)} = [\min_{1\leq i\leq S_n} \underline{\boldsymbol{f}(\boldsymbol{x}_i)}, \min_{1\leq i\leq S_n} \overline{\boldsymbol{f}(\boldsymbol{x}_i)}]$. Очевидно, что нижняя и верхняя граница достигаются на каких-то подбрусах i' и i'' (возможно совпадающих) из рабочего списка и для дальнейшего улучшения оценки глобального оптимума $\boldsymbol{E}$ необходимо их (адаптивное) дробление на более мелкие подбрусы.

Будет раздроблен тот или иной брус на очередном шаге разработанного интервального генетического алгоритма, зависит от значения функции приспособленности (3.11). Так как при её вычислении учитывается значение интервальной оценки, а также её ширина и размер бруса, то хотя бы при одном из коэффициентов $\alpha_i, \beta_i, \gamma_i$, отличном от нуля в формуле (3.11), ведущие брусы $\boldsymbol{x}'$ и $\boldsymbol{x}''$ окажутся среди доминантных и будут выбраны для дробления.

Из сказанного следует, что в ходе работы интервального генетического алгоритма границы интервальной оценки оптимума действительно уточняются и утверждение доказано.

Утверждение 3.3. Интервальный генетический алгоритм не упускает из рассмотрения глобальные оптимумы.

Доказательство. Рассмотрим этапы предложенного метода. На вход алгоритм получает интересующую нас область поиска $\boldsymbol{x}$ и целевую функцию f.

Нулевым шагом интервального генетического алгоритма является создание начальной популяции. Для этого исходная область определения функции

(брус) $\boldsymbol{x}$ разбивается на n подобластей так, что $\boldsymbol{x} = \sum_{i=1}^{n} x_i^0$, где x_i^0 — i-ая особь начальной популяции, соответственно, на этом шаге алгоритм не может упустить из рассмотрения область, в которой находится глобальный оптимум.

Далее, в основном цикле некоторое количество наиболее приспособленных особей и какие-то из неприспособленных особей оставляют некоторое количество потомков. Конкретное поведение здесь зависит от настроек алгоритма, но в любом случае на этом этапе алгоритм не может упустить из рассмотрения области, в которых находятся глобальные оптимумы.

После потомки проверяются на жизнеспособность. В этот момент применяются интервальные критерии отбраковки. Их применение безопасно, так как интервальные критерии из рассмотрения удаляют лишь те подобласти исходной области поиска, для которых можно строго показать, что на них глобальный оптимум находиться не может. В случае, если критерий отбраковки по значению был улучшен, начинается эпидемия, то есть все особи проверяются на соответствие изменённым критериям, и не удовлетворяющие этим критериям удаляются из популяции. При этом также гарантируется, что будут отброшены лишь те подобласти, для которых достоверно известно, что глобальный оптимум не может на них содержаться.

Таким образом, положение и значение глобального оптимума не могут быть пропущены и утверждение доказано. Следовательно, доказана и теорема 3.3.

3.5 Универсальный (мультиметодный) алгоритм

Итак, подведем промежуточный итог. Нами реализованы и исследованы различные подходы для поиска глобального оптимума, использующие стохастические техники наряду с методами интервального анализа:

- случайное интервальное дробление,
- случайное интервальное дробление с приоритетом,
- интервальный алгоритм имитации отжига и
- несколько вариантов интервального биологического алгоритма
- варианты дробления в случайной пропорции и направлении,
- бисекция с перекрытием.

Для разных целевых функций эффективными оказываются разные алгоритмы из этого списка. Какой алгоритм справится лучше с каждой конкретной проблемой — вопрос по-прежнему открытый. Мы провели некоторые исследования, чтобы ответить на этот вопрос. Общий вывод следующий — чем меньше точность интервальной оценки целевой функции, чем избыточнее интервальное расширение, тем более оправдано применение стохастических методик. Результаты экспериментов показывают, что если в аналитическом выражении целевой функции присутствует возведение аргументов более чем в четвёртую-пятую степень или если количество вхождений какой-либо переменной более трёх-четырех раз, то рандомизированный подход, как правило, справляется с такой задачей быстрее.

Идея с предварительными тестами хороша, но на них тратится время, что соответственно увеличивает суммарное время решения задачи. Естественный выход из сложившегося затруднения состоит в том, чтобы вообще отказаться от предварительного тестирования и делать динамический выбор того или иного метода на основании истории их работы с интересующей нас задачей, т. е. тестирование проводить параллельно с решением и на его основе.

Конкретизация этой идеи может быть следующей. Запускаем какой-то (любой) алгоритм глобальной оптимизации. Через короткий промежуток времени останавливаем его и на существующем рабочем списке брусов даём поработать другому алгоритму, т. е., фактически, на ходу подменяем алгоритмы оптимизации. При этом отслеживаем относительную эффективность каждого из алгоритмов — сколько итераций было сделано, насколько улучшилась оценка оптимума, как изменился размер рабочего списка и т. п. Имеет смысл по каждому методу отслеживать как глобальную статистику, то есть результаты за всё время работы алгоритма, так и локальную — результаты

каждого конкретного сеанса работы. Сравнивая успешность методов, можно динамически регулировать квант времени, выделяемый каждому из алгоритмов, обеспечивая оптимальность применения методов. Таким образом, нами реализуется «мета-алгоритм», адаптивный алгоритм второго порядка (по аналогии с терминологией, принятой в функциональных языках для функций, оперирующих функциями) или мультиметодный подход, как в работах Тятюшкина и Горнова [14, 15].

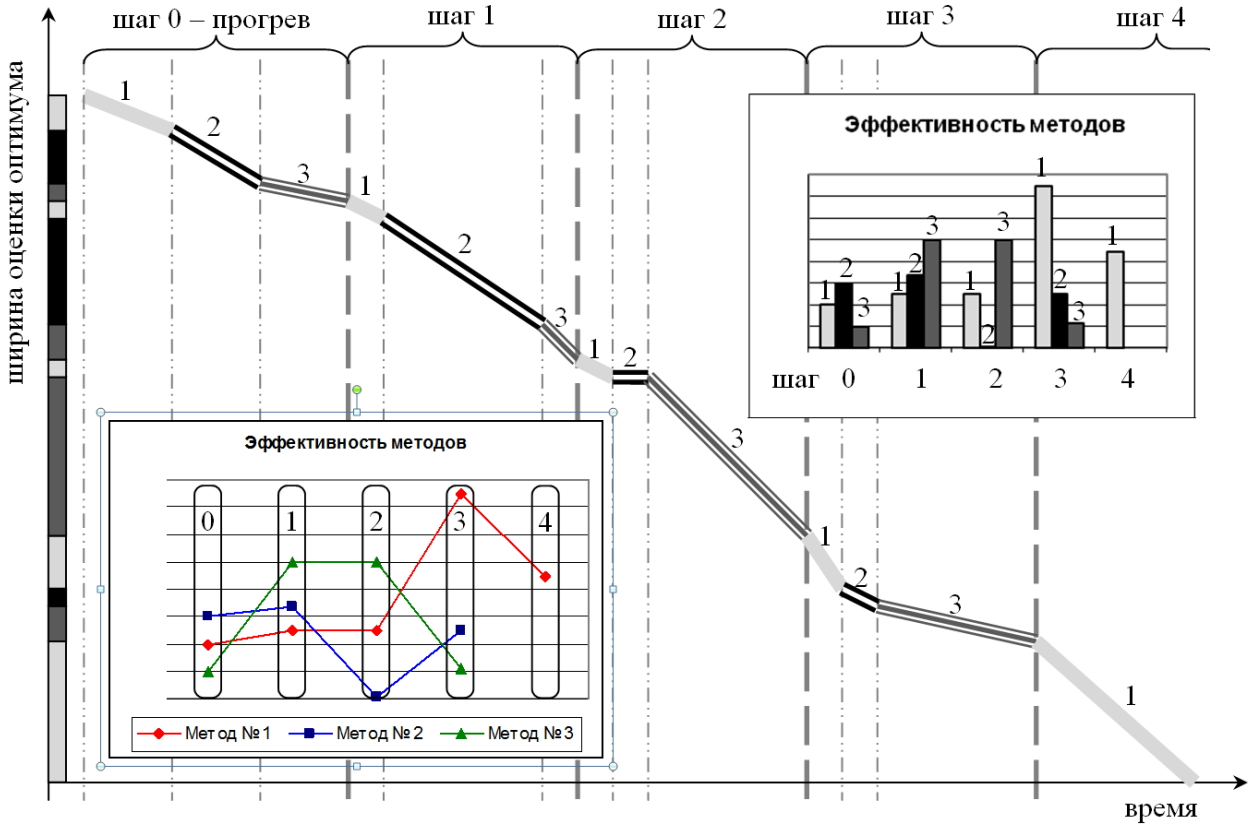


Рис. 3.14: Пример переключения между алгоритмами

На рис. 3.14 показан пример работы такого алгоритма для случая трёх методов глобальной оптимизации. Для простоты показана схема, в которой решение, какой метод будет признан перспективным для следующего шага, принимается только на основе информации с предыдущего шага. То есть, анализ суммарной успешности методов за всё время работы не проводится. Различные методы на графиках помечены различными цветами. На начальном этапе (стадия прогрева) алгоритмам предоставляется некоторое, одинаковое для всех, время работы. По результатам выбирается наиболее успешный, то есть тот, которому удалось улучшить оценку оптимума более других. На следующей итерации ему будет выделено больше времени для работы.

В показанной реализации всем остальным алгоритмам будет предоставлено одинаковое небольшое время для работы. На следующем шаге относительная успешность алгоритмов снова сравнивается и выделяемое время корректируется.

Эффективность такого алгоритма можно оценить следующим образом. Пусть наш мета-метод объединяет в себе n методов глобальной оптимизации. Время работы каждого отдельного метода на i-той итерации мета-метода будем обозначать как $t_i^j,\ j \in \{1 \dots n\}$. Причем это время может быть либо t_{long}, если алгоритм зарекомендовал себя как успешный, либо t_{small} — в противоположном случае. То, насколько j-му алгоритму удалось улучшить точности оценки оптимума на i-ом шаге, обозначим как d_i^j.

Тогда за m шагов мы получим следующую ширину оценки глобального экстремума D:

$$D = D_0 - \sum_{j=0}^{n-1} d_0^j - \sum_{i=1}^{m} \sum_{j=0}^{n-1} d_i^j, \tag{3.12}$$

где D_0 — точность начальной оценки, d_0^j — уточнения, сделанные каждым из алгоритмов на стадии прогрева. Соотношение между итерациями и временем работы t тоже просто:

$$t = t_0 \cdot n + (\,(t_{small} \cdot (n-1) + t_{long}) + t_{overhead} \cdot n) \cdot iteration, \tag{3.13}$$

где t_0 — время первой, «прогревочной» итерации; $t_{overhead}$ — время на подмену алгоритмов, сбор и анализ результатов; $iteration$ — количество итераций.

Для получения оценки снизу рассмотрим наихудший случай. Положим, что на каждом шаге лишь один алгоритм был существенно полезен. То есть, пусть на каждой итерации i существует лишь один алгоритм j_0, такой что

$$d_i \ll d_i^{j_0} \qquad \text{для всех} \quad j \neq j_0. \tag{3.14}$$

Более того, пусть на каждой итерации это будет другой алгоритм. То есть не существует два таких шага i_k и i_{k+1}, что бы соотношение (3.14) было верно. Для простоты пренебрежём начальной итерацией (стадией прогрева), будем считать что время на подмену алгоритмов, сбор и сравнение результатов ничтожно мало по сравнению со временем работы алгоритмов, а также упростим (3.14): положим, что на каждом шаге все методы, кроме одного вообще не уточняют оценку оптимума. В этом, наихудшем для себя варианте, наш мультиметодный алгоритм будет минимально эффективен (см. рис. 3.15). При этом не надо забывать, что он будет гораздо эффективнее каждого отдельного метода.

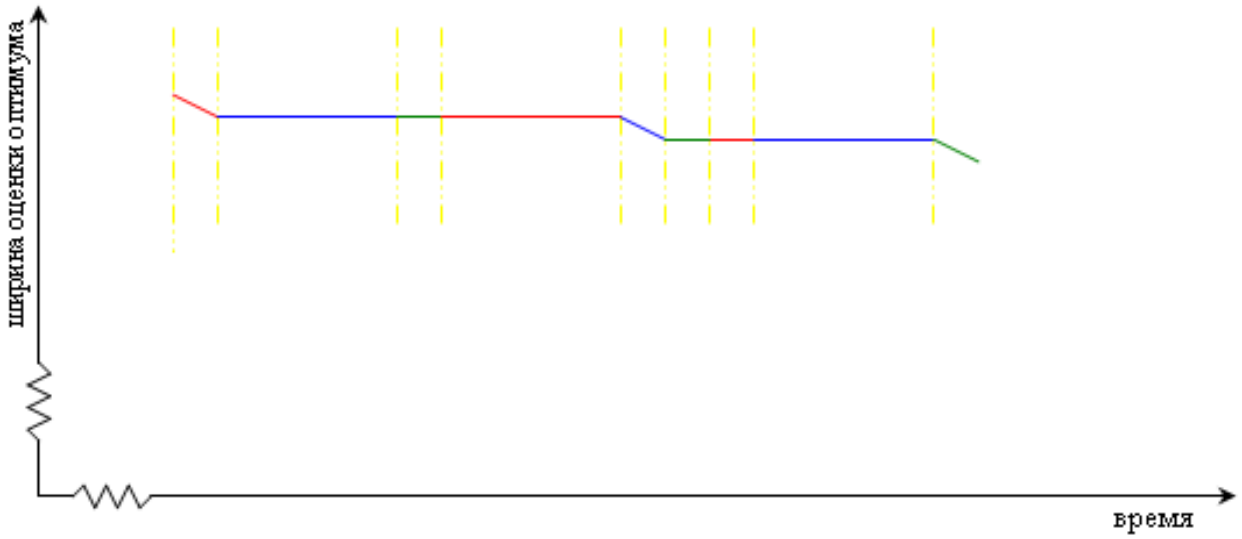


Рис. 3.15: Иллюстрация наихудшего варианта работы мультиметодного алгоритма

Другой вариант, когда есть один «хороший» метод, оказывающийся успешным на каждой итерации и множество «плохих» методов, которые вообще (рассмотрим максимально плохой случай) не улучшают оценки. В этом случае описанный выше мультиметодный алгоритм потратит на

$$d_0 \cdot (n-1) + t_{small} \cdot (n-1) + t_{overhead} \cdot n \tag{3.15}$$

времени больше, чем тот «хороший» метод. Проблема лишь в том, что в большинстве случаев заранее не известно, какой метод будет «хорошим» для предложенной целевой функции. К тому же эффективность мета-метода можно повысить, анализируя успешность алгоритмов не только на предыдущем шаге, как это было изображено на рис. 3.14, но учитывая информацию о всех предыдущих итерациях и выделяя не постоянные промежутки времени t_{small} или t_{long}, а t_i, являющиеся функциями от локальной и глобальной успешности. Главное не переусложнить механизм арбитража настолько, что увеличившееся $t_{overhead}$ сведет на нет все достижения.

3.6 Параллельный интервальный алгоритм глобальной оптимизации

В 1965 году Гордон Мур, один из основателей компании Intel, опубликовал в журнале «Electronics» статью, впоследствии ставшую весьма цитируемой, а сделанное в ней предположение получило название «закона Мура» [109]. Следует отметить, что закон Мура является наблюдением-экстраполяцией,

то есть когда-то он может перестать выполняться. Но вот уже более сорока лет технологии развиваются, технические процессы совершенствуются и инженеры-физики успешно следуют ему, каждые два года увеличивая примерно вдвое число транзисторов на единице площади кристалла.

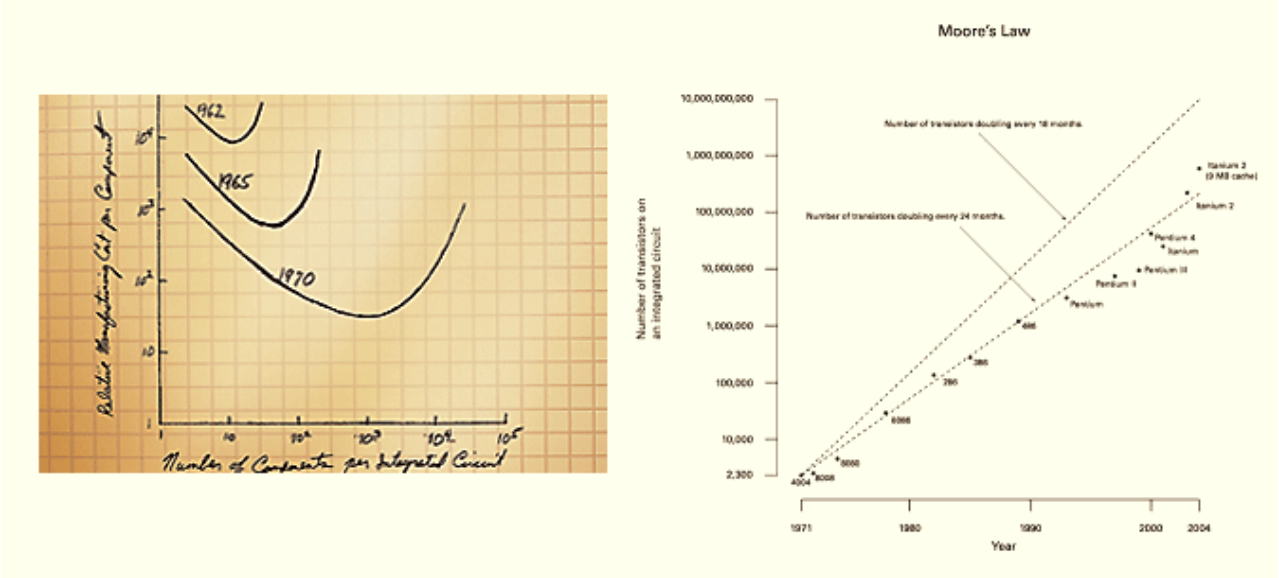


Рис. 3.16: Закон Гордона Мура.
Слева: Оригинальная схема, нарисованная Гордоном Муром в 1965 году.
Справа: Увеличение количества транзисторов в процессоре на примере архитектуры Intel от времени. Логарифмическая вертикальная шкала.

Но такое развитие не может быть бесконечным. Уже сейчас практически достигнут фундаментальный предел. Последующее уменьшение размеров и увеличение плотности элементов при тех же подходах станет невозможным из-за туннельных эффектов, квантовых эффектов да и тепловыделения.

Также невозможно бесконечно уменьшать толщину проводников и количество электронов, необходимых для фиксирования наличия сигнала. Ограничивается это волновой природой электрона и кристаллической решеткой кремния, выбранного за основу всех процессоров. Совершенствовать математические схемы выполнения операций для наименьшей задержки прохождения сигнала тоже можно лишь до определенного предела. Так, если предположить, что процессор работает на частоте 5 ГГц, то период одного такта составляет $2 \cdot 10^{-10}$ секунд. Так как скорость распространения электрического поля равна скорости света, то за это время электрическое поле успевает распространиться лишь на 6 см. Учитывая также время срабатывания логических элементов, получаем, что времени одного такта недостаточно для распространения сигнала по геометрической диагонали процессора, даже при наличии прямого пути.

Возможности дальнейшего увеличения тактовой частоты процессоров тоже не безграничны. Дело в том, что одновременно с частотой резко возрастает количество тепла, которое выделяет работающий процессор. Так, у последних моделей Pentium 4 тепловыделение не менее 100 ватт [34].

Архитекторы-схемотехники отказались от дальнейшего увеличения частоты, будучи не в силах побороть возникающие при этом высокочастотные помехи и всё те же тепловые эффекты. Но нашли другой не менее эффективный способ увеличивать производительность, размещая сотни простых процессоров-ядер на одном кристалле. Правда, этот способ действительно другой, непривычный для программиста.

Изменения в «железе» требуют пересмотра привычной архитектуры программных систем. Если код написан в расчёте на последовательное исполнение, из всех ядер будет работать только одно, что не приведёт ни к какому увеличению производительности.

В настоящее время недостаточно предложить алгоритм решения задачи. Необходимо, чтобы алгоритм был приспособлен для эффективной параллельной работе в многомашинной и многопроцессорной среде.

Мета-алгоритм, описанный в предыдущей главе, отлично подходит для распараллеливания. Действительно, для самой простой реализации, достаточно поделить процессоры между методами глобальной оптимизации и отключить систему распределения времени. К сожалению, эффективность такой версии будет крайне невысока.

При параллельной работе независимых алгоритмов, оперирующих общей структурой данных весьма вероятен конфликт доступа. Например, несколько алгоритмов одновременно решат раздробить один и тот же брус. Наиболее неприятна при этом ситуация, когда какой-то алгоритм собрался работать с брусом, а другой к этому времени уже понял, что этот самый брус бесперспективен и удаляет его. Аналогичная конфликтная ситуация: один алгоритм ещё не закончил процедуру вставки нового подбруса в рабочий список, а другой уже начал его переупорядочивание. Единственный способ обеспечить корректную работу такого параллельного алгоритма — синхронизация алгоритмов и обеспечение атомарности доступа к рабочему списку. Это означает, что прежде чем начать работать с общими данными, алгоритм должен проверить — не работает ли уже с ними какой-то другой метод. Только дождавшись освобождения общего ресурса, он может начинать операции чтения или модификации. Понятно, что такие блокировки превращают алгоритм в последовательный и сводят на нет все преимущества параллельной архитектуры.

Например, если на четырехпроцессорной платформе Intel Xeon 5100 «Woodcrest» мы решим для ускорения поиска глобального экстремума запустить в параллель четыре одинаковых алгоритма, например, «имитации отжига» над общим рабочим списком, то прирост скорости, вместо ожидаемых четырех раз по сравнению с одним алгоритмом на одном процессоре, составит всего двадцать процентов. Потери эффективности, а следовательно, и вычислительного времени происходят из-за синхронизации, когда при одновременной попытке работы с разделяемым ресурсом — рабочим списком брусов, все потоки кроме одного вынуждены простаивать, ожидая, когда он освободит доступ.

Повысить скорость работы программы можно путём улучшения способа доступа к разделяемым данным, а можно вообще отказаться от общего рабочего списка. Разделим исходную область определения целевой функции на подобласти по числу доступных процессоров, раздадим каждому из них свой исходный брус и запустим по независимому алгоритму глобальной оптимизации со своей начальной областью на каждом процессоре. Полезно в такой алгоритм добавить регулярную процедуру обмена текущими верхними и нижними оценками глобального экстремума. Понятно, что делать это надо периодически, но не слишком часто. Но как реализовать этот обмен наиболее эффективно?

Очевидное решение — через какое-то наперёд заданное и одинаковое для всех алгоритмов количество итераций они обмениваются данными. Но для этого каждый из них будет вынужден на время прекратить счёт и ждать, пока все остальные тоже досчитают до этой точки и будут готовы к обмену данными. Даже если у нас в параллель будет работать один и тот же детерминистский алгоритм, на разных подобластях исходной области определения целевая функция ведет себя по-разному, интервальные оценки целевой функции по-разному избыточны, да и абсолютные значения оценок различны. Это означает, что процедуры отбраковки, вставки новых подбрусов (либо же поиск ведущего бруса или периодическая сортировка рабочего списка — в зависимости от алгоритма) будут выполняться разное время. Как следствие, в точке обмена данными все параллельные процессы, кроме самого медленного, неизбежно будут простаивать.

Описанная несбалансированность лишь усилится в случае одновременной работы разнородных алгоритмов. Особенно актуально сделанное наблюдение в случае разработанных нами интервальных стохастических реализаций, для которых время работы каждой итерации зависит от удачности случайного выбора, и, следовательно, может сильно варьироваться. Это неминуемо при-

ведёт к потере общей эффективности из-за увеличения доли вынужденных простоев. В результате диаграмма работы потоков может приобрести вид, показанный на рис. 3.17.

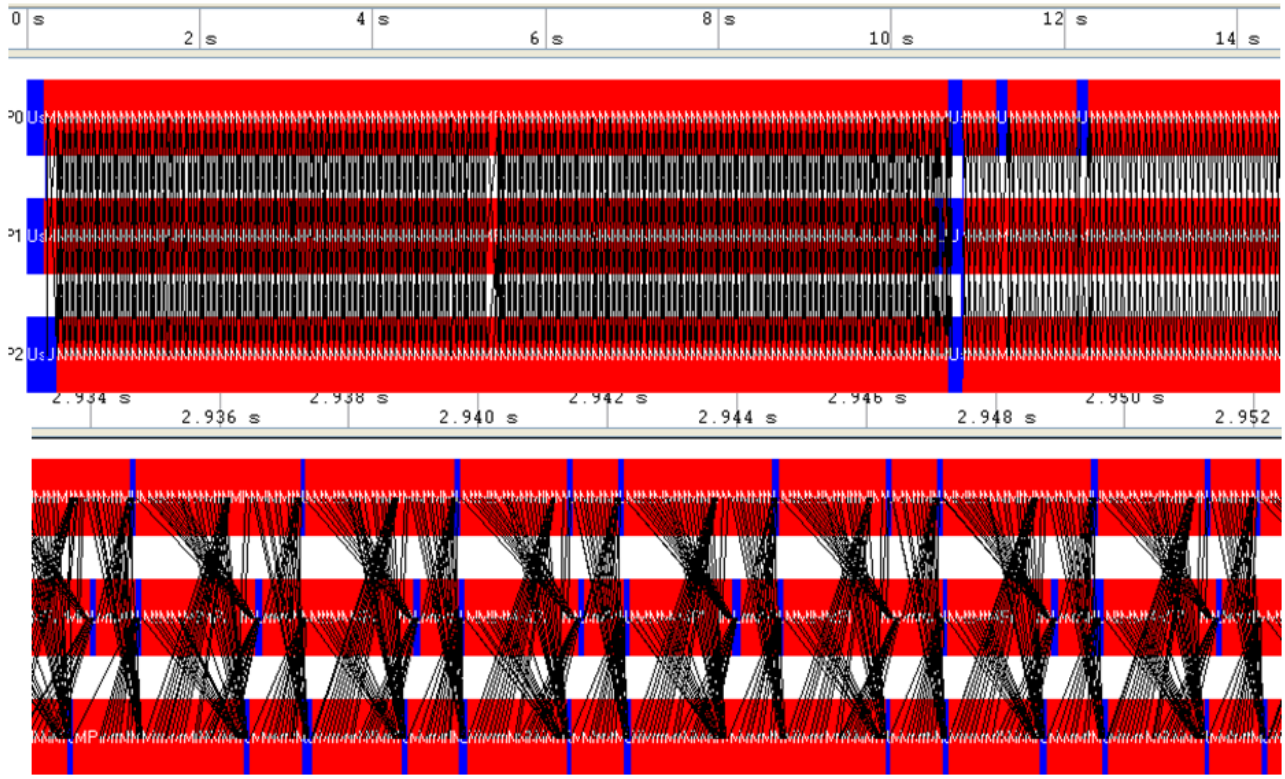


Рис. 3.17: Диаграмма потоков неоптимально распараллеленной программы. Синим показано время работы непосредственно вычислительного кода, красным обозначено время, потраченное на синхронизацию и обмен данными между потоками. Вверху приведен общий вид, внизу — отрезок диаграммы в увеличенном масштабе.

Избежать ожидания самого медленного процесса можно лишь при асинхронной работе. При такой схеме взаимодействия никакой метод не обменивается данными ни с одним другим напрямую. Если в соответствии со своей внутренней логикой какой-то метод решает, что своими текущими результатами стоит обменяться с другими методами, он просто делает эти данные публичными. Иными словами, есть некая область памяти, про которую известно всем алгоритмам. Каждый из алгоритмов может прочитать из неё текущее значение, и, если его оценки лучше, то обновить рекорд. Иначе прочитанные значения используются в качестве пороговых для процедур отбраковки.

Единственное необходимое условие корректной работы состоит в том, что все обращения к общим данным должны быть атомарными, то есть должно гарантироваться, что ни один метод не сможет получить к данным доступ, если другой ещё не закончил свою работу с ними. Но это обязательное требо-

вание не приведёт к сколько-нибудь серьезным потерям производительности, если дополнительно рассинхронизовать методы, т. е. сделать так, чтобы обмен рекордами происходил в разное время.

Архитектуры современных вычислительных машин предполагают многопроцессорность и многоядерность. Это означает, что при обращении к одной и той же переменной из потоков, выполняющихся на разных сокетах, требуется аппаратная синхронизация быстрой памяти (кэшей) обоих сокетов. Для устранения этого вместо общей переменной был создан специальный поток, циклически забирающий рекорды у вычислительных алгоритмов[35].

После получения очередных границ глобального оптимума и исполнения процедур отбраковки может оказаться, что рабочий список у какого-то процесса пуст. Это означает доказательство того, что на подобласти, доставшейся этому потоку, глобального экстремума быть не может. Ситуация эта не редкая, и в таком случае целесообразно поделить рабочий список. Для этого можно ввести дополнительный интерфейс, останавливающий потоки и запрашивающий данные. Но по нашим оценкам, это нецелесообразно. Реализованная нами асинхронная схема при правильно подобранных временах синхронизации демонстрирует очень хорошую скорость. Обмен данными происходит следующим образом. Поток, прочитавший новый рекорд, после завершения процедур отбраковки проверяет размер рабочего списка. Если список пуст, выставляется соответствующий флаг, который проверяется всеми алгоритмами после каждого чтения глобального рекорда. Первый обнаруживший такой запрос процесс гасит флаг и делится своими данными с простаивающими процессами. Численные замеры производительности с применением различных комбинаций алгоритмов на разных целевых функциях показали, что наиболее удачным является следующий способ разделения данных.

Если рабочий список достаточно объёмный, от него отщепляется лишь такое количество брусов, которые поместятся в кэш процессора (здесь надо учитывать, что кэш последнего уровня делится между ядрами и помнить о возможном пересечении адресов и эффектах от так называемого снуп-фильтра). Подробнее об этом можно прочитать, например в [123]. Если же количество брусов невелико, они будут единовременно раздроблены до получения примерно нужного количества.

3.7 Вычислительные эксперименты

Разработанный нами параллельный мультиметодный алгоритм показал свою высокую вычислительную способность. Так, глобальный минимум функции

«Price 5» (1.36), см. рис. 1.3 и 1.4 на стр. 48, был локализован за 0.42 секунды; функции «Griewank 2» (1.43) (см. стр. 53) за 0.61 секунды; четвертой функции Розенброка (1.44) (см. стр. 56) за 0.72 секунды; функции Треккани (1.15), см. стр. 33 и 111, за 0.67 секунд. При выполнении замеров производительности методов использовалась опция -XXpreOptProfile с указанием предварительно собранного профиля для этой задачи, благодаря чему Java–машина, производила все необходимые оптимизации кода до запуска приложения.

Следующие тестовые функции были взяты из работ [90, 92, 105, 115].

Для функции Шафера

$$f(x) = 0.5 + \frac{\sin^2\left(\sqrt{x_1^2 + x_2^2}\right) - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2} \tag{3.16}$$

оптимум был локализован за 1.98 секунды;

для изменённой функции Шафера №1

$$f(x) = 0.5 + \frac{\sin^2(x_1^2 + x_2^2) - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2} \tag{3.17}$$

оптимум был локализован за 1.27 секунды;

для изменённой функции Шафера №2

$$f(x) = 0.5 + \frac{\sin^2(x_1^2 - x_2^2) - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2} \tag{3.18}$$

глобальный минимум был найден за 0.9 секунды,

для функции под названием «решётка для яиц» (Egg holder function)

$$f(x) = -(x_2 + 47)\sin\left(\sqrt{|x_2 + x_1/2 + 47|}\right) + \sin\left(\sqrt{|x_1 - (x_2 + 47)|}\right)(-x_1), \tag{3.19}$$

которая упоминается в работе [105] как трудная для оптимизации, глобальный минимум был найден за 29.34 секунды,

для функции под названием «синус оборачивающий синус» (Sine envelope sine wave function)

$$f(x) = \sum_{i=1}^{m-1}\left(\frac{\sin^2\left(\sqrt{x_{i+1}^2 + x_i^2}\right) - 0.5}{\left(0.001(x_{i+1}^2 + x_i^2) + 1\right)^2} + 0.5\right) \tag{3.20}$$

при $m = 2$ глобальный оптимум был найден за 36.25 секунд из-за многократных переключений алгоритмов,

зато для функции Чичинадзе (Chichinadze)

$$f(x) = x_1^2 - 12x_1 + 11 + 10\cos(\pi x_1/2) + 8\sin(5\pi x_1) - (1/5)^{0.5} e^{-0.5(x_2-0.5)} \quad (3.21)$$

глобальный минимум был найден за 1.3 секунды,
для функции McCormick

$$f(x) = \sin(x_1 + x_2) + (x_1 - x_2)2 - 1.5x_1 + 2.5x_2 + 1 \quad (3.22)$$

глобальный минимум был найден за 0.28 секунды,
для 13 функции Леви (Levy13)

$$f(x) = \sin^2(3\pi x_1) + (x_1 - 1)^2\big(1 + \sin^2(3\pi x_2)\big) + (x_2 - 1)^2\big(1 + \sin^2(3\pi x_2)\big) \quad (3.23)$$

оптимум был найден за 0.81 секунды,
для функции Zettle

$$f(x) = (x_1^2 + x_2^2 - 2x_1)^2 + 0.25x_1 \quad (3.24)$$

оптимум был найден за 2.06 секунды,
для функции Стиблинского-Танга (Styblinski-Tang)

$$f(x) = \frac{1}{2}\sum_{i=1}^{2}(x_i^4 - 16x_i^2 + 5x_i) \quad (3.25)$$

оптимум был найден за 1.77 секунды,
для функции Леона

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad (3.26)$$

оптимум был найден за 1.58 секунды.

Кроме того, реализованный алгоритм был опробован в том числе на обобщённой функции Розенброка (3.27) (называемой также в отечественной литературе «функцией Скокова»):

$$f(x) = \sum_{i=1}^{n-1}(100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2). \quad (3.27)$$

Ниже приводятся результаты нашего метода именно для этой функции, так как в работе [18] эта функция упоминается и используется в качестве «классического тестового примера для алгоритмов оптимизации». Там же [18] приводятся следующие результаты:

Таблица 3.3: Зависимость времени работы параллельного метода неравномерных покрытий предложенного авторами [18] и приводимое в той же работе от числа процессоров для обобщённой функции Розенброка с числом переменных 5 и точностью 0.001 на Суперкомпьютере МВС-6000IM (256 процессоров Intel®Itanium®2)

число процессоров	1	2	4	8	16	32
время работы (сек.)	28.02	14.23	7.13	3.94	2.51	1.82

Реализованный нами алгоритм, работающий на одном компьютере серии Intel Xeon X7460 «Dunnington» 2.6 GHz, показал следующие результаты:

количество потоков	1	8	16
время работы (сек.)	78.8	2.02	1.52

При увеличении размерности время работы изменялось следующим образом:

размерность задачи	5	10	15	20	25	30	35
время работы (сек.)	2	7	167	610	1058	3784	18066

что, в основном, вызвано неэффективной работой с оперативной памятью и некоторыми избыточными синхронизациями и будет исправлено в следующих версиях алгоритма.

В другом вычислительном эксперименте для целевой функции Экли

$$f(x) = 20 \cdot \exp\left(\frac{1}{5\sqrt{n}} \sum_{i=i}^{n} x_i^2\right) - \exp\left(\frac{1}{n} \sum_{i=i}^{n} \cos(2\pi x_i)\right) + \exp(1)$$

при $n = 35$ глобальный минимум был найден параллельным мультиметодным алгоритмом всего за 1.49 сек.

Для целевой функции де Йонга

$$f(x) = \sum_{i=1}^{n} x_i^2$$

и второй («повёрнутой») функции де Йонга

$$f(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2,$$

являющимися популярными тестовыми задачами глобальной оптимизации [16], при размерности $n = 200$ глобальный оптимум был найден параллельным мультиметодным алгоритмом за 2.33 и 7.81 секунд соответственно. Графики этих целевых функций можно увидеть на рис. 3.18.

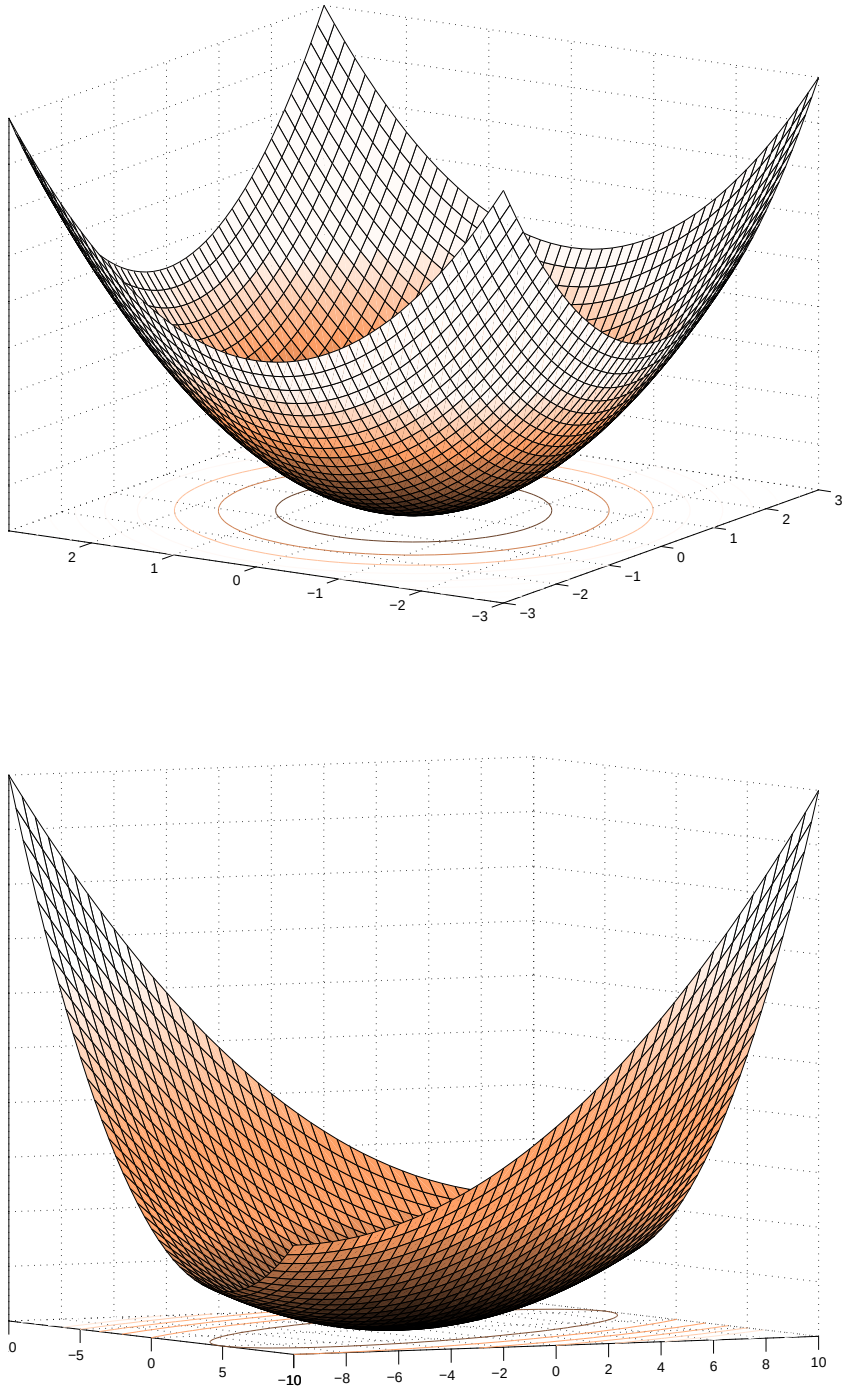


Рис. 3.18: Графики функций де Йонга.

Отметим, что в силу выпуклости последних функций, их оптимизация на простых областях никаких проблем не представляет и может быть выполнена, к примеру, градиентным спуском. Но они часто используются в качестве тестов именно для неклассических методов оптимизации.

3.8 Практическое применение

В этом параграфе мы опишем применение разработанных выше в диссертации алгоритмов оптимизации в программах предсказания погоды и моделирования климата.

Предсказание погоды и моделирования климата в настоящее время — востребованные и бурно развивающиеся области на стыке физики, математики и программирования. Современные погодные и климатические модели являются результатами многолетних трудов специализированных институтов.

Основные физические процессы, учет которых обычно реализуется в гидродинамических моделях прогноза погоды и общей циркуляции атмосферы, схематически представлены на рис. 3.19.

Неадиабатические физические процессы, связанные с турбулентными и конвективными движениями, фазовые превращения воды, радиационные и облачные эффекты существенно влияют на развитие атмосферной циркуляции [32]. При этом учитывается наличие мелкодисперсных частиц, таких как пыль, принимается во внимание газовый состав атмосферы, учитывается влияние примесей [80].

Во всем мире центры предсказания погоды — одни из потребителей современных суперкомпьютеров и высокопроизводительных компьютерных кластеров. Ведь требуется одновременно и быстро рассчитывать прогнозы погоды разных сроков, для разных территорий с разным пространственно-временным разрешением. Постоянно идет обработка данных в масштабе континента, строится общий фон. В масштабах страны строится грубая модель. Вокруг городов насчитывается более подробная локальная схема. Типичная задача для аэропорта — расчет в реальном времени высоконадежного прогноза метеоусловий, пусть краткосрочного — всего на несколько ближайших часов, но с высоким разрешением по всем трем координатам на участке хотя бы $20 \times 10 \times 5$ километров. Тут же расчеты для военных, обслуживание самолетных трасс, стратегических объектов и так далее. Требуется высокопроизводительный и дорогостоящий кластер на несколько сот, а подчас и тысяч процессоров. В случае неоптимальности алгоритма её придется компенсировать «железом» — устанавливать больше вычислительных блоков, а значит и па-

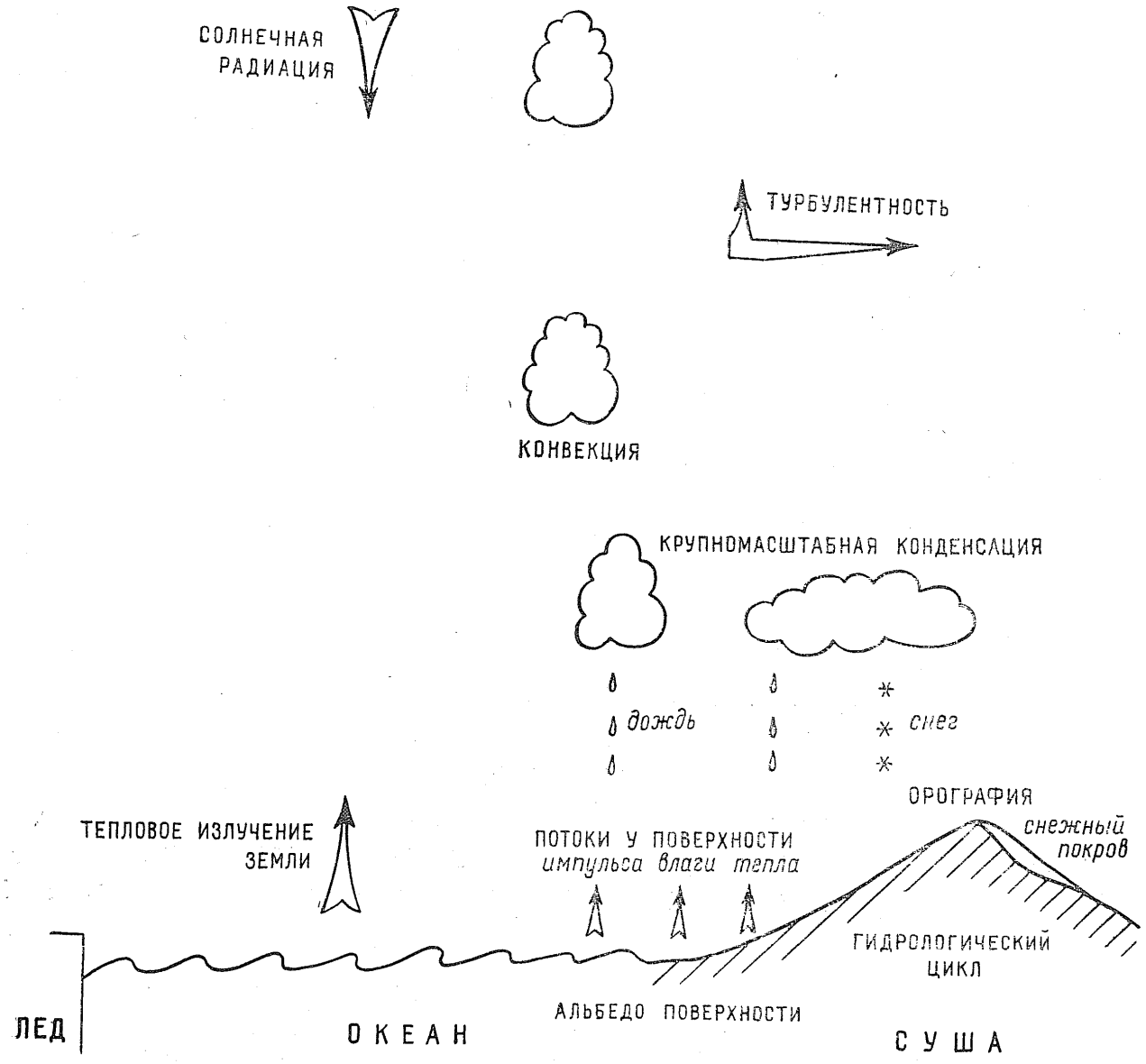


Рис. 3.19: Схематическое представление основных физических процессов в атмосфере

мяти, и специального внутрикластерного коммуникационного оборудования, а затем и дополнительных систем охлаждения и питания. Начиная с какого-то момента наращивание вычислительной мощности кластера ведет к экспоненциальному удорожанию системы. Кроме того, также непропорционально возрастает стоимость поддержания работоспособности системы и возрастает вероятность сбоя. Использование неоптимальных алгоритмов или моделей означает недосчитанные или слишком грубые прогнозы, либо же неминуемо приведет к наращиванию вычислительных мощностей. Таким образом, улучшение существующих математических моделей, поиск более совершенных алгоритмов наряду с оптимизацией реализации уже существующих — ничуть не менее важная задача, чем разработка новых погодных моделей. А,

возможно, и даже более приоритетная на данном этапе.

Мы работали с новой моделью, называемой `mwf_t1024rlb`, предназначенной для составления кратко и средне срочных прогнозов в локальной области. Перед нами стояла задача, по-видимости никак не связанная с интервальным анализом, а именно — оптимизировать алгоритм, ускорить работу программы, рассчитывающей прогноз погоды по этой модели.

Пользуясь специальными программными инструментами для анализа производительности программ, мы шаг за шагом, двигаясь от функции к функции, скрупулезно изучали код и оптимизировали программу, до тех пор, пока в одной из компонентах кода, ответственном за расчёт атмосферы не обнаружилось функция поиска минимума.

Программа проектировалась и создавалась для работы на больши́х распределенных системах и, соответственно, использовала параллельную модель обработки данных. Применительно к задаче поиска глобального минимума, это было реализовано следующим образом. Сначала весь объем данных копировался по вычислительным узлам, затем на каждом из них запускалось несколько независимых процедур мультистарта. По окончании вычислительные узлы обменивались найденными рекордами и уже из них выбиралось минимальное значение. Это весьма странная реализация. Ведь с увеличением количества вычислительных узлов время работы функции не уменьшается, но увеличивается. Более того, чем на меньшем количестве вычислительных узлов запущенна программа, тем больше вероятность найти лишь локальный минимум.

Для улучшения производительности достаточно добиться масштабируемости функции, иными словами, изменить функцию так, чтобы при работе на бо́льшем количестве вычислительных узлов время работы уменьшалось. При этом желательно, чтобы качество результата не зависело от того, на каком количестве вычислительных узлов программа была запущенна. В принципе, достаточно запускать на всех вычислительных узлах не одно и то же постоянное количество процедур локального поиска, а варьировать его в зависимости от общего количества доступных процессоров (см. листинг 12).

В порядке эксперимента мы переписали существующую функциональность её интервальным аналогом основанным на представленных выше разработках. Как и ожидалось, это сразу переместило функцию из «горячих», то есть таких функций, на исполнение которых тратится основное время работы программы, глубоко в конец списка потребителей процессорного времени и, соответственно, принесло несколько дополнительных процентов производительности. А это, как очевидно из предыдущего абзаца, важное достижение.

Листинг 12. Более подходящий способ реализации параллельного поиска глобального оптимума (на языке Фортран).

```
 . . .

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Внешние переменные

! Желательное количество задач локальной оптимизации,
!  которое кажется разумным для данной задачи.
integer :: opt_tasks

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Внутренние переменные

! Количество процессоров в группе
integer :: comm_size

! Переменные MPI: коммуникатор и код ошибки
integer :: comm, ierr

! Текущий и глобальный оптимум
real :: opt
real :: g_opt

 . . .


do 1, (opt_tasks / com_size) + 1
    opt = StartMultiStart(domain)
    g_opt = min(opt, g_opt)
end do

! Сбор глобального минимума
call MPI_Reduce(g_opt, opt, 1, MPI_REAL, MPI_MIN, 0, comm, ierr)
! После вызова этой функции у процесса с ранком 0
!  в переменной <<opt>> будет сохранено минимальное
!  из найденных значений.
```

Подчеркнем, что основной прирост был достигнут, в частности, из-за просчётов в реализации оригинальной схемы, а именно, из-за неуменьшающегося времени работы подпрограммы оптимизации при увеличении количества процессоров. Тем не менее, пример демонстрирует успешное применение интервальных методов глобальной оптимизации.

Для реализации успешной параллельной работы алгоритма мы пошли по пути, частично описанному в §3.6. В нём описывался алгоритм для платформ с общей памятью. Рассматриваемое приложение предназначено для работы одновременно на большом числе вычислительных узлов. Каждый узел в отдельности является многопроцессорной системой с общей памятью и все они объединены в кластер. Таким образом, теперь у нас появились два уровня параллелизации.

Мы разделили исходную область определения между вычислительными узлами, на каждом из которых работал параллельный мета-алгоритм. Дополнительный уровень параллелизации увеличил свободу действий. Мы использовали разные алгоритмы оптимизации на разных узлах, предусмотрели создание на разных вычислительных узлах одних и тех же алгоритмов, но использующих разные типы интервальных расширений. При этом мы отслеживали успешность каждого метода, и в соответствии с этой статистикой динамически увеличивали количество успешных на данном этапе алгоритмов в ущерб менее успешным. При большом количестве процессоров в кластере такая методика оказалась гораздо успешнее, чем просто разделение входных данных между экземплярами одного алгоритма глобальной оптимизации, работающего в параллель на всех вычислительных узлах. Результаты этой работы вошли в доклад на конференции «Use of High Performance Computing in Meteorology» [112].

Помимо описанного успешного применения разработанных нами новых подходов к решению задачи поиска глобального оптимума, мы работаем над решением ряда других практических проблем.

Например, одной из важных задач молекулярной биологии является синтез новых веществ, в частности белковых структур, с заданными свойствами. Белки играют очень важную роль практически во всех процессах живого организма, и то, с какими рецепторами и как именно будет взаимодействовать та или иная молекула, во многом определяется её конфигурацией. Таким образом, задача предсказания в какую форму «свернётся» та или иная поли-молекула выходит на первый план. Пространственная конфигурация молекулы определяется минимумом её энергии, соответственно, поставленная задача сводится к решению задачи о поиска глобального минимума в

многомерном пространстве. Это одно из возможных применений разработанных нами методов, но, пока, мы не приступили к решению этой интересной задачи.

В настоящее время мы решаем вариант задачи глобальной оптимизации, возникающей при лазерной эллипсометрии. Эллипсометрические измерения относятся к неразрушающим оптическим методам исследования поверхности и приповерхностные свойства структурно и химически неоднородных объектов. Круг задач в этом направлении непрерывно расширяется, изучаются все более сложные объекты, исследования которых требует создания всё более сложных математических моделей. Суть метода заключается в том, что по значениям некоторых оптических показателей образца, снятым в процессе его исследования, решается «обратная задача эллипсометрии» — восстанавливается вид функциональной зависимости свойств материала от толщины. Решение этой обратной задачи фактически сводится к минимизации функционала (3.28), содержащего неизвестные параметры системы.

$$F = \sum_{i=1}^{N} \Big((\frac{\Delta_i^c - \Delta_i^m}{\delta\Delta})^2 - (\frac{\psi_i^c - \psi_i^m}{\delta\psi})^2 \Big), \tag{3.28}$$

где Δ_i^c, ψ_i^c — рассчитанные на основе данной модели эллипсометрические параметры; Δ_i^m, ψ_i^m — эллипсометрические параметры, полученные экспериментальным методом, $\delta\Delta$ и $\delta\psi$ — экспериментальные ошибки измерения соответствующих величин. При этом типичными значениями для размерности N являются числа порядка тысячи, иногда десятков тысяч.

Итоги главы 3

Основные итоги главы 3 диссертации состоят в следующем:

1) Продемонстрирована вычислительная эффективность стохастических (рандомизированных) алгоритмов интервальной глобальной оптимизации в сравнении с детерминистскими аналогами на ряде общеупотребительных тестовых задач.

2) Модифицированы существующие стохастические (рандомизированные) алгоритмы интервальной глобальной оптимизации, такие как случайное интервальное дробление (предложенное ранее научным руководителем работы). Предложены и исследованы некоторые новые подходы, в частности, интервальное дробление в случайной пропорции и направлении.

3) Разработаны и экспериментально исследованы различные версии интервального генетического алгоритма для глобальной оптимизации функций.

4) Для предложенных алгоритмов сформулирован и доказан ряд утверждений и теорем о сходимости разработанных методов. В частности, доказано, что метод интервального случайного поиска с приоритетом сходится к глобальному оптимуму почти всегда (Сходимость «почти всегда» сильнее «сходимости по вероятности»); а интервальный алгоритм имитации отжига и интервальный генетический алгоритм сходятся к глобальному оптимуму.

5) Разработан и апробирован мультиметодный алгоритм интервальной глобальной оптимизации, созданный на основе стохастических интервальных подходов. Приведены оценки эффективности и времени работы метода, в том числе проанализирован худший для алгоритма случай. С помощью мультиметодного метода успешно решена задача оптимизации в программе моделирования погоды `mwf_t1024rlb`.

Кроме того, в главе 3 прорабатываются вопросы параллелизма предлагаемых методов и их реализации на современных вычислительных системах.

Заключение

Задачи глобальной оптимизации, в особенности доказательной глобальной оптимизации, являются актуальными и сложными. Использование методов интервального анализа позволяет конструировать для их решения вычислительные технологии, удовлетворительно справляющиеся со многими практическим задачами, но вычислительная эффективность получающихся алгоритмов во многих случаях остаётся недостаточной.

В диссертации проанализирована классическая вычислительная схема интервальных детерминистских алгоритмов глобальной оптимизации, основанных на адаптивном дроблении области поиска. В результате был выявлен ряд причин недостаточной вычислительной эффективности существующих алгоритмов, основанных на этой схеме. Предложены и исследованы пути их преодоления, такие как улучшение качества интервального оценивания, повышение эффективности процедур отбраковки, адаптация различных способов дробления и хранения брусов с учётом особенностей архитектуры современных компьютеров. В работе получила развитие идея использования стохастических переходов при выборе бруса для уточнения оптимума. В частности, в

работе реализованы и исследованы алгоритмы случайного дробления и случайного дробления с приоритетом, интервальный алгоритм имитации отжига и, кроме того, разработаны интервальный эволюционный и мультиметодный алгоритмы. На их основе сконструирован гибридный параллельный алгоритм интервальной глобальной оптимизации. Кроме того, в работе сформулированы и доказаны ряд утверждений и теорем о сходимости разработанных методов. На широком наборе общеупотребительных тестовых задач глобальной оптимизации проведены численные эксперименты по апробации новых вычислительных алгоритмов.

Объединение интервальных и стохастических (рандомизированных) подходов позволило создать принципиально новые, эффективные алгоритмы для решения задачи доказательной глобальной оптимизации — одной из востребованных проблем современной вычислительной и прикладной математики.

Литература

[1] Азенкотт Р. Процедура «отпуска» // *Труды семинара Н. Бурбаки за 1988 год.* – М.: Мир, 1990. – С. 235–251.

[2] Айда-Заде К.Р., Евтушенко Ю.Г. Быстрое автоматическое дифференцирование // *Математическое моделирование.* – 1989. – №1(1). – С. 121–131.

[3] Аоки М. *Введение в методы оптимизации.* – М.: Наука, 1977.

[4] Бауэр Ф.Л., Гооз Г. *Информатика. Вводный курс: В 2-х ч. Ч.1. Перевод с немецкого* – М.: Мир, 1990.

[5] Бахвалов Н.С., Жидков Н.П., Кобельков Г.Г. *Численные методы. 8-е изд.* – М.: Лаборатория Базовых Знаний, 2000.

[6] Беда Л.М., Королев Л.Н., Сухих Н.В., Фролова Т.С. *Программа для автоматического дифференцирования для машины БЭСМ.* // Ин-т точной механики и вычислительной техники АН СССР, Москва, 1959.

[7] Букатова И.Л. *Эволюционное моделирование и его приложения.* – М.: Наука, 1979.

[8] Бухбергер Б., Калме Ж., Калтофен Э., Коллинз Дж., Лауэр М., Лафон Ж., Лоос Р., Минньотт М., Нойбюзер Й., Норман А., Уинклер Ф., ван Хюльзен Я. *Компьютерная алгебра: Символьные и алгебраические вычисления.* – М.: Мир, 1996.

[9] Васильев Ф.П. *Методы оптимизации* – Мю: Факториал Пресс, 2009.

[10] Васильев Ф.П. *Численные методы решения экстремальных задач* – М.: Наука, 1988.

[11] Гаганов А.А. О сложности вычисления интервала значений полинома от многих переменных // *Кибернетика.* – 1985. – №4. – С. 6–8.

[12] Гашков С.Б. *Системы счисления и их применения.* — Издательство московского центра непрерывного математического образования. М., 2004.

[13] Гладков Л. А., Курейчик В. В, Курейчик В. М. и др. *Биоинспирированные методы в оптимизации: монография.* — М.: Физматлит, 2009.

[14] Горнов А.Ю. *Вычислительные технологии решения задач оптимального управления.* – Новосибирск: Наука, 2009.

[15] Горнов А.Ю. Тятюшкин А.И. Программная реализация мультиметодной технологии для задач оптимального управления // *Сборник трудов 3-й Междунар. конф. «Проблемы управления и моделирования в сложных системах»*, Самара, 4–9 сентября 2001 г. – Самара, 2001. – С. 301–307.

[16] Долгов Ю.Г. Метод глобальной оптимизации на основе метода ветвей и границ // *Труды Международной конференции по вычислительной математике МКВМ-2004. Рабочие совещания* / Ред.: Шокин Ю.И., Федотов А.М., Молородов Ю.И., Ковалёв С.П., Семёнов А.Л., Шарый С.П. – Новосибирск: Изд-во ИВМиМГ СО РАН, 2004. – С. 184-192.

[17] Евтушенко Ю.Г. *Численный метод поиска глобального экстремума функций (перебор на неравномерной сетке)* // *Ж. вычисл. матем. и матем. физики.* – 1971. – Т. 11, №6. – С. 1390—1403.

[18] Евтушенко Ю.Г., Посыпкин М.А. Параллельные методы решения задач глобальной оптимизации // *Труды четвёртой международной конференции PACO'2008 «Параллельные вычисления и задачи управления». М., 27–29 октября 2008.* – М.: Институт проблем управления им. В. А. Трапезникова. – С. 19—39.

[19] Евтушенко Ю.Г., Ратькин В.А. *Метод половинных делений для глобальной оптимизации функций многих переменных* // *Известия Академии Наук СССР. Техническая Кибернетика.* – 1987. – №1. – С. 119—127.

[20] Ершов А.Р., Хамисов О.В. Автоматическая глобальная оптимизация // *Труды XIV Байкальской международной школы-семинара «Методы оптимизации и их приложения», Иркутск, Байкал, 2–8 июля 2008 года. Т. 1 «Математическое программирование».* – Иркутск, ИСЭМ СО РАН, 2008. – С. 235–244.

[21] Жиглявский А.А., Жилинскас А.Г. *Методы поиска глобального экстремума.* – М.: Наука, 1991.

[22] Жилинскас А.А., Шалтянис В.Р. *Поиск оптимума: компьютер расширяет возможности.* – М.: Наука, 1989.

[23] Зюбин В.Е., Панов Н.В., Шарый С.П. Распределенная система для решения задач глобальной оптимизации функции методами интервального анализа с возможностью трехмерной визуализации // *Материалы Десятой Всероссийской научной конференции студентов-физиков и молодых ученых.* — Красноярск, 2004. – С.195.

[24] Калиткин Н.Н. *Численные методы.* – М.: Наука, 1978.

[25] Калмыков С.А., Шокин Ю.И., Юлдашев З.Х. *Методы интервального анализа.* — Новосибирск: «Наука», 1986.

[26] Канторович Л.В. О проведении численных и аналитических вычислений на машинах с программным управлением // *Изв. АН АрмССР, сер. физ.-мат. наук.* – 1957. – Т. 10, №2. – С. 3–16.

[27] Канторович Л.В. Об одной математической символике, удобной при проведении вычислений на машинах // *Доклады АН СССР.* – 1957. – Т. 113. – С. 738–741.

[28] Катковник В. *Линейные оценки и стохастические задачи оптимизации.* – М.: Наука, 1976.

[29] Коротченко А.Г. Об одном алгоритме поиска наибольшего значения одномерных функций. // *ЖВМ и МФ* – 1978. – Т .18, №3. – С.563-573.

[30] Кулиш У., Рац Д., Хаммер Р., Хокс М. *Достоверные вычисления. Базовые численные методы.* – Москва-Ижевск: Издательство «РХД», 2005.

[31] Кудрявцев Л.Д *Краткий курс математического анализа. Т.2. Дифференциальное и интегральное исчиления функций многих переменных. Гармонический анализ.* — М.: ФИЗМАТЛИТ, 2002.

[32] Курбаткин Г.П., Дегтярев А.И., Фролов А.В. *Спектральная модель атмосферы, инициализация и база данных для численного прогноза погоды.* – Санкт-Петербург: Гидрометеоиздат, 1994.

[33] Левитин А.В. *Алгоритмы: введение в разработку и анализ. Глава 6. Метод преобразования: Схема Горнера и возведение в степень.* – М.: «Вильямс», 2006.

[34] Леонтьев В. *Новейшая энциклопедия ПК-2006.* — ОЛМА-ПРЕСС, 2006.

[35] Лозбень М.Е., Панов Н.В. Параллельные алгоритмы интервальной глобальной оптимизации // *Труды Международной конференции «Современные проблемы прикладной математики и механики: теория, эксперимент и практика», посвященной 90-летию со дня рождения академика Н.Н. Яненко (Новосибирск, Россия, 30 мая - 4 июня 2011 г.). – №гос. регистр. 0321101160, ФГУП НТЦ «Информрегистр». – Новосибирск. – 2011. – http://conf.nsc.ru/files/conferences/niknik-90/fulltext/39370/52239/LozbenPanov.pdf*

[36] Лоэв М. *Теория вероятностей.* — М.: Иностранная литература, 1962.

[37] Лядова М.А., Панов Н.В. Применение интервального метода Ньютона и его модификации для решения задачи поиска глобального оптимума функций // *Труды Международной конференции «Современные проблемы прикладной математики и механики: теория, эксперимент и практика», посвященной 90-летию со дня рождения академика Н.Н. Яненко (Новосибирск, Россия, 30 мая - 4 июня 2011 г.). – №гос. регистр. 0321101160, ФГУП НТЦ «Информрегистр». – Новосибирск. – 2011. – http://conf.nsc.ru/files/conferences/niknik-90/fulltext/38161/47719/Lyadova1.pdf*

[38] Панков П.С. Алгоритмы доказательства устойчивых утверждений и глобальной оптимизации в ограниченной области. – Фрунзе, 1984. – 13 с. – Депонировано в ВИНИТИ, №5250-84Деп.

[39] Панов Н.В. Автоматическая верификация процессоров с использованием методов интервального анализа и удовлетворения ограничений // *Тезисы VI Всероссийской конференции молодых ученых по математическому моделированию и информационным технологиям (с участием иностранных ученых). 29–31 октября 2005 года, г.Кемерово, Россия.* — Кемерово: КемГУ, 2005. (Электронный вариант доступен на http://www.ict.nsc.ru/ws/show_abstract.dhtml?ru+130+9372)

[40] Панов Н.В. Адаптивный мета-алгоритм глобальной оптимизации // *Естественные и технические науки* – 2009. – №1 (39). – С. 315–318.

[41] Панов Н.В. Гибридные алгоритмы глобальной оптимизации // *Труды XV Байкальской международной школы-семинара «Методы оптимизации и их приложения». Т.2 Математическое программирование. Иркутск: РИО ИДСТУ СО РАН, 2011. С. 145–150.*

[42] Панов Н.В. Гибкость и гарантированность. Интервальные стохастические методы поиска оптимума // *IX Всероссийская конференция молодых ученых по математическому моделированию и информационным технологиям, Кемерово, 28–30 октября 2008 г. Программа и тезисы докладов.* — Кемерово: КемГУ, 2008. – С.101–102.

[43] Панов Н.В. Комбинирование точечных и интервальных методов поиска глобального оптимума // *XII Всероссийская конференция молодых ученых по математическому моделированию и информационным технологиям. Россия, г. Новосибирск, 3-6 октября 2011г. Дополненные тезисы докладов, с. 55-56.*

[44] Панов Н.В. Обзор интервальных методов глобальной оптимизации // *Тезисы докладов VII Всероссийской конференции молодых ученых по математическому моделированию и информационным технологиям.* — Красноярск. – 2006. – С. 25.

[45] Панов Н.В. Объединение стохастических и интервальных подходов для решения задач глобальной оптимизации функций // *Вычислительные технологии.* – 2009. – Т. 14, №5. – С. 49–65.

[46] Панов Н.В. Оценка области значений функций методами интервального анализа // *Вопросы современной науки и практики.* – Тамбов, 2009. – (Сборник научных трудов / Университет им. Вернадского: № 3(17)). – С. 78–86.

[47] Панов Н.В., Колдаков В.В. Программный комплекс для графического представления процесса и результатов работы интервальных алгоритмов // *Труды Пятой международной конференции памяти академика А.П. Ершова «Перспективы систем информатики».* — Новосибирск, ИПО Эмари, 2003. Т. «Международное совещание по интервальной математике и методам распространения ограничений». — С. 38–46.

[48] Панов Н.В., Шарый С.П. Интервальный эволюционный алгоритм поиска глобального оптимума // Известия Алтайского государственного университета. (ISSN 1561-9443). 2011. №1(69). Т.2. С. 108-113.

[49] *Шарый С.П., Панов Н.В.* Пакет визуализации интервальных методов глобальной оптимизации // Студент и научно-технический прогресс: Материалы XLI Международной научной студенческой конференции. Серия «Физика: Автоматизация научных исследований и машинной графики». Новосибирск: НГУ. 2003. С. 21.

[50] Панов Н.В., Шарый С.П. Развитие стохастических подходов в интервальной глобальной оптимизации // *VIII Всероссийская конференция молодых учёных по математическому моделированию и информационным технологиям, Новосибирск, 27–29 ноября 2007 г., Тезисы докладов.* — Новосибирск, 2007. – С. 22.

[51] Панов Н.В. Развитие стохастических подходов в интервальной глобальной оптимизации. Интервальный генетический алгоритм // *Международная конференция «Современные проблемы математического моделирования и вычислительных технологий – 2008», Красноярск, 18–24 августа 2008 г. Тезисы докладов.* — Красноярск: СФУ. – С. 31–32.

[52] Панов Н.В. Решатель задач поиска глобального минимума и максимума функций // *Труды Международной конференции «Современные проблемы прикладной математики и механики: теория, эксперимент и практика», посвященной 90-летию со дня рождения академика Н.Н. Яненко* (Новосибирск, Россия, 30 мая - 4 июня 2011 г.). – №гос. регистр. 0321101160, ФГУП НТЦ «Информрегистр». – Новосибирск, 2011. – http://conf.nsc.ru/files/conferences/niknik-90/fulltext/38157/46913/Panov2.pdf

[53] Панов Н.В., Шарый С.П. Стохастические интервальные подходы в задачах глобальной оптимизации функций // *Тезисы докладов Всероссийской конференции по вычислительной математике «КВМ-2007».* – Новосибирск, 18–20 июня 2007 г. (Электронный вариант доступен на http://www-sbras.nsc.ru/ws/show_abstract.dhtml?ru+164+12106)

[54] Панов Н.В., Шарый С.П. Стохастические подходы в интервальных методах глобальной оптимизации // *Всероссийское (с международным участием) совещание по интервальному анализу и его приложениям ИНТЕРВАЛ-06, 1–4 июля 2006 года, Петергоф, Россия. Расширенные тезисы докладов.* — Санкт-Петербург: ВВМ, 2006. – С. 101–105.

[55] Первозванский А.А. *Поиск.* – М.: Издательство «Наука», главная редакция физико-математической литературы, 1970.

[56] Поляк Б.Т. *Введение в оптимизацию.* – М.: Наука, 1983.

[57] Протасов В. *Максимумы и минимумы в геометрии.* (Серия: «Библиотека «Математическое просвещение»). – М.: МЦНМО, 2005.

[58] Рубан, А. И. *Глобальная оптимизация методом усреднения координат.* – Красноярск: ИПЦ КГТУ, 2004.

[59] Сергеев Я.Д., Квасов Д.Е. *Диагональные методы глобальной оптимизации.* – М.: Физматлит, 2008.

[60] Сергеев Я.Д., Квасов Д.Е. Глобальная оптимизация и условия Липшица // *Сборник научно-популярных статей — победителей конкурса РФФИ 2008 года. Вып. 12. Часть I.* – М.: Октопус-Природа, 2009. – С. 19–29.

[61] Стрекаловский А.С., Орлов А.В. *Биматричные игры и билинейное программирование.* – М.: Физматлит, 2007.

[62] Стронгин Р.Г. *Поиск глобального оптимума. Серия «Математика и кибернетика», вып. 2.* – М.: Знание, 1990.

[63] Стронгин Р.Г. *Численные методы в многоэкстремальных задачах. Информационно-статистический подход.* – М.: Наука, 1978.

[64] Стронгин Р.Г., Гришагин В.А. Оптимизация многоэкстремальных функций при монотонно унимодальных ограничениях // *Известия АН СССР. Техническая кибернетика.* – 1984. – №4. – С. 203–208

[65] Харчистов Б.Ф. *Методы оптимизации. Учебное пособие.* — Таганрог: ТРТУ, 2004.

[66] Хювёнен Э., Сеппянен Й. *Мир Лиспа. В 2-х томах.* – М.: Мир, 1990.

[67] Фихтенгольц Г.М. *Курс дифференциального и интегрального исчисления. Т. 2.* – М.: ФИЗМАТЛИТ, 2001.

[68] Фогель Л., Оуэнс А., Уолш М. *Искусственный интеллект и эволюционное моделирование.* – М.: Мир, 1969.

[69] Чернова, Н. И. *Теория вероятностей.* – Новосибирск, 2007.

[70] Шарый С.П. Стохастические подходы в интервальной глобальной оптимизации // *Труды XIII Байкальской международной школы-семинара*

«Методы оптимизации и их приложения», Иркутск–Северобайкальск, 2–8 июля 2005 года. Т. 4 «Интервальный анализ». – Иркутск, ИСЭМ СО РАН, 2005. – С. 85–105.

[71] Шарый С.П. Рандомизированные алгоритмы в интервальной глобальной оптимизации // *Сиб. Журнал Вычисл. Матем.* – 2008. – Т. 11, №4. – С. 457–474.

[72] Шарый С.П. *Конечномерный интервальный анализ.* – Электронная книга, доступная на `http://www.nsc.ru/interval/index.php?j=Library/InteBooks/index`.

[73] Шарый С.П., Колдаков В.В., Панов Н.В. Интервальный симулированный отжиг для глобальной оптимизации функций // *Материалы конф. XLI Международной научной студенческой конференции «Студент и научно-технический прогресс».* — Новосибирск: НГУ. – 2003.

[74] Ширяев А.Н.. *Вероятность.* — М.: МЦНМО. – 2004.

[75] Шокин Ю.И. *Интервальный анализ.* — Новосибирск: Наука. Сиб. Отделение. – 1981.

[76] Aho A., Ulman J. *The design and Analysis of Computer Alghorithms.* – Reading, Mass.:Addison-Wesley, 1974.

[77] Akhmerov R.R. Interval-affine Gaussian algorithm for constrained systems // *Reliable Computing.* – 2005. – Vol. 11, No. 5. – P. 323–341.

[78] Ali M., Torn A., Viitanen S. Stochastic global optimization: problem, classes and solution techniques // *Journal of Global Optimization.* – 1999. – Vol. 14. – P. 437–447.

[79] Becker R.W., Lago G.V. A global optimization algorithm // *Proceedings of the 8th Allerton Conference on Circuits and Systems Theory*, 1970. P. 3–12.

[80] Bengtsson L. Medium range forecasting // *GARP Spec. Rep.* – 1985. – No. 43, 3/12–3/45.

[81] Caprani O., Madsen K., Nielsen H.B. *Introduction to interval analysis.* – Technical University of Denmark, DTU, 2002.

[82] Caprani O., Madsen K. Mean value forms in interval analysis // *Computing.* – 1980. – Vol. 25. – P. 147–154.

[83] Chee-Keng Y. *Fundamental Problems of Algorithmic Algebra.* – Oxford: Oxford University Press, 2000.

[84] Cohen J. *Computer Algebra and Symbolic Computation: Mathematical Methods.* – Peters, 2003.

[85] Ding-Zhu Du, Pardalos P., Weili Wu *Mathematical Theory of Optimization* – Kluwer Academic Publishers, Dordrecht, 2001.

[86] Diwekar U. *Introduction to Applied Optimization* – Springer, 2008.

[87] Dixon, L.C.W., Szego G.P. *Towards global optimization* – North-Holland, 1975.

[88] Dixon L.C.W., Szego G.P. (eds.) *Towards Global Optimisation 2* – North-Holland, Amsterdam, 1978.

[89] Frigo M., Leiserson C., Prokop H., Ramachandran S. *Cache-obligious algorithms* // Proceeding of the 40th Annual Symposium of Foundations of Computer Science (FOCS'99). — New York, 1999. IEEE Computer Society Press, Los Alamitos.

[90] Jansson C., Knüppel O. A Global Minimization Method: The Multi-Dimensional Case // *Berichte des Forschungsschwerpunktes Informations und Kommunikationstechnik.* – Hamburg: TU Hamburg, 1992.

[91] Zhigljavsky A., Zilinskas, A. *Stochastic Global Optimization* – Springer, 2008.

[92] Jung B.S., Karney B.W. Benchmark Tests of Evolutionary Computational Algorithms // *Environmental Informatics Archives (International Society for Environmental Information Sciences)* – 2004. – Vol. 2. – pp. 731-742.

[93] Hansen E. A generalized interval arithmetic // *Interval Mathematics / K. Nickel, ed.* – Berlin-Heidelberg: Springer-Verlag, 1975. – P. 7–18. – (Lecture Notes in Computer Science, 29)

[94] Hansen E., Walster G. *Global optimization using interval analysis.* – New York: Marcel Dekker, 2004.

[95] Holland J. *Adaptation in Natural and Artificial Systems.* – University of Michigan Press. 1975.

[96] Kahrimanian H. *Analytical Differentiation by a Digit Computer. M. A. Thesis.* – Temple University. Philadelphia. 1953.

[97] Kearfott R. *Rigorous Global Search: Continuous Problems.* – Dordrecht: Kluwer Academic Publishers, 1996.

[98] Kirkpatrick S., Gelatt C.D., Vecchi M.P. Optimization by simulated annealing // *Science.* – 1983. – Vol. 220. – P. 671–680.

[99] Fogel, David B *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* – IEEE Press, Piscataway, NJ. Third Edition, 2006

[100] Kreinovich V., Kearfott R.B. Beyond convex. Global optimization is feasible only for convex objective functions: a theorem // *Journal of Global Optimization.* – 2005. – Vol. 33, No. 4. – P. 617–624.

[101] Kushner H. A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. // *Transactions ASME, Ser. D, J. Basic Eng.*, – 1964. – Vol. 86, No. 1. – P. 97–106.

[102] Lambert M.S., Miriam T.T., Susan F.M. *Simulated Annealing: Global Optimization, Applied Mathematics, Global Optimum, Metropolis-Hastings Algorithm, Monte Carlo Method, Nicholas Metropolis, Quantum Annealing.* – Betascript Publishers, 2009.

[103] Markov S. Extended interval arithmeticinvolving infinite intervals // *Mathematica Balkanica (new series).* – 1992. – Vol. 6. – P. 269–304.

[104] Metropolis N., Rosenbluth M., et al. Equation of State Calculations by Fast Computing Machines // *J. Chem. Phys.* – 1953. – Vol.21. – P. 1087.

[105] Mishra, S.K. Global Optimization by Differential Evolution and Particle Swarm Methods: Evaluation on Some Benchmark Functions // *SSRN* – 2006. Available at SSRN: http://ssrn.com/abstract=933827

[106] Moore R.E. *Interval Analysis.* — New Jersey, Prentice-Hall, 1966.

[107] Moore R.E. *Methods and Applications of Interval Analysis.* – SIAM, Philadelphia, 1979.

[108] Moore R.E., Kearfott R.B., Cloud M. *Introduction to Interval Analysis.* – SIAM, Philadelphia, 2009.

[109] Moore G.E. Cramming more components onto integrated circuits // *Electronics Magazine.* – 1965.

[110] Neumaier A. *Interval metods for systems of equations.* – Cambridge: Cambridge Universiry Press, 1990.

[111] Nolan J. *M. A. Thesis. Analytical Differentiation by a Digit Computer* – MIT. Cambridge. 1953.

[112] Panov N. *Twelfth ECMWF Workshop «Use of High Performance Computing in Meteorology» October 30–November 3, 2006.* – Reading, RG2 9AX, United Kingdom. – 2006. – P. 27.

[113] Plauger P., Stepanov A., Lee M., Musser D. *C++ Standard Template Library.* – Prentice Hall PTR, 2000.

[114] Polak E. *Computational Methods in Optimization (Mathematics in Science and Engineering Ser).* – New York: Academic Press, 1971.

[115] Parsopoulos K.E., Vrahatis M.N. Recent Approaches to Global Optimization Problems Through Particle Swarm Optimization // *Computing* – 2002. – Vol. 1. – P. 235–306.

[116] Ratschek H. Inclusion functions and global optimization // *Mathematical Programming.* – 1985. – Vol. 33. – P. 300–317.

[117] Ratschek H., Rokne J. *Computer Methods for the Range of Functions.* – Horwood, Chichester, 1984.

[118] Ratschek H., Rokne J. *New computer methods for global optimization.* – Chichester, New York: Ellis Horwood, Halsted Press, 1988.

[119] Ratz D., Csendes T. On the selection of subdivision directions in interval branch-and-bound methods for global optimization // *Journal of Global Optimization.* – 1995. – Vol. 7, No. 2. – P. 183–207.

[120] Rechenberg I. *Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* – Stuttgart: Fromman-Holzboog, 1973. (1993 – 2^{nd} edn.).

[121] Sendov B. Some topics of segment analysis // *Interval Mathematics* / Ed. by K. Nickel. – New York: Academic Press, 1980. – P. 203–222.

[122] Schwefel H.-P. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie.* – Basel: Birkhaeuser, 1977.

[123] Shanly T., Colwell B. *The Unabridget Pentium 4.* – Addison-Wesley, 2005.

[124] Skelboe S. Computation of rational interval functions // *BIT.* – 1974. – Vol. 14. – P. 87–95.

[125] Stengle G. Error analysis of a randomized numerical method // *Numerische Mathematik.* – 1995. – Vol. 70. – P. 119–128.

[126] Stolfi J., de Figueiredo L.H. *Self-validated numerical methods and applications.* – Brazilian Mathematical Society, 1997.

[127] Stoutemyer D. *Automatic simplification.* – ACMSIGSAM Bill. 10/6 97-104, 1976.

[128] Tao Wang *Global optimization for constrained global programming.* – Ph.D. thesis at University of Illinois at Urbana-Champaign, 2001.

[129] Torn A.A. Global optimization as a combination of global and local search // *Proceedings of Computer Simulation Versus Analytical Solutions for Business and Economic Models.* – Gothenburg, 1972. – P. 191–206.

[130] Torn A.A. *Global optimization as a combination of global and local search. Ph.D. Thesis.* – Abo Akademi, Abo, 1974.

[131] http://www.basegroup.ru/genetic/math.htm